Multiobjective Hyperparameter Optimization for Deep Learning Interatomic Potential Training Using NSGA-II

Mark Coletti
colettima@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Ada Sedova
sedovaaa@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Rajni Chahal
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Luke Gibson
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Santanu Roy
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Vyacheslav S. Bryantsev
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

ABSTRACT

Deep neural network (DNN) potentials are an emerging tool for simulation of dynamical atomistic systems, with the promise of quantum mechanical accuracy at speedups of 10000×. As with other DNN methods, hyperparameters used during training can make a substantial difference in model accuracy, and optimal settings vary with dataset. To enable rapid tuning of hyperparameters for DNN potential training, we developed a scalable multiobjective optimization evolutionary algorithm for supercomputers and tested it on the Summit system at the Oak Ridge Leadership Computing Facility (OLCF). The multiobjective approach is required due to the coupling of two learned values defining the potential: the energy and force. Using a large-scale implementation of the NSGA-II algorithm adapted for training DNN potentials, we discovered several optimal multiobjective combinations, including best choices of activation functions, learning rate scaling scheme, and pairing of the two radial cutoffs used in the three dimensional descriptor function.

KEYWORDS

evolutionary computation, hyperparameter optimization, high performance computing, machine learning, neural networks, multiobjective optimization, neural network potentials, molecular simulation

ACM Reference Format:
Mark Coletti, Ada Sedova, Rajni Chahal, Luke Gibson, Santanu Roy, and Vyacheslav S. Bryantsev. 2023. Multiobjective Hyperparameter Optimization for Deep Learning Interatomic Potential Training Using NSGA-II. In *Proceedings of The Third International Workshop on Paralle and Distributed Algorithms for Decision Sciences (PDADS).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Simulation is an essential component of research in in the physical and chemical sciences. Key processes driving chemical phenomena like chemical bond breaking and formation are best simulated with methods which consider electronic structure, which is inherently quantum mechanical. First principles (FP) simulation methods (those that explicitly simulate electrons using quantum mechanics) are equipped to simulate such processes most accurately [23]. These simulation methods, however, are computationally expensive, while chemical processes often involve complex energy surfaces and computational treatment of time and length scales larger than these accurate simulation methodologies can directly produce. For our research on molten aluminum halides, which are of interest to efforts in separations and nuclear energy, computational studies can provide substantial insights. Molten aluminum halides have unique and complex properties, and unfortunately, empirical models which are substantially cheaper [26] cannot capture the chemical accuracy required for an understanding of chemical reactivity.

Extending simulations such as FP molecular dynamics (FPMD) to timescales that permit more direct comparisons with laboratory-measured values would require orders of magnitude speedups; limitations from Amdahl's law and hardware prohibit such speedups for even the cheapest FP methods, such as density functional theory (DFT) [26, 25]. Recently, breakthroughs in the use of machine learned interatomic potentials—the potential energy fields describing the interactions of atoms and their (negative) vector gradients, which are the forces—have promised to provide a transformative solution to this difficulty: trained on data from FP simulations, these surrogate models can provide the accuracy of FP calculations, with dramatic reductions in computational cost, sometimes exceeding 10000× [31, 17, 1, 5], representing a "dream come true" for molecular and atomistic simulation [28]. ML methods include Gaussian approximation potential (GAP)-based potentials [18] and deep neural network (DNN) potentials (DNNPs) [31, 1].

Hyperparameters involved in training DNNs can have a dramatic effect on the final model accuracy, and are often tuned manually, which can be a tedious task [21, 3]. Automated hyperparameter tuning algorithms are therefore desired, but finding the optimal set of hyperparameters for DNNs is extremely difficult due to the size of the parameter search space and associated lengthy training times; the commonly used grid-based searched has been shown to be prone to missing optimal values unless a very fine grid is used [2], which would be prohibitive considering the numbers of hyperparameters used in DNN training. Evolutionary algorithms (EAs) can be well-suited strategies for optimization problems over large search spaces [19], and have been used to optimize both DNN training and model hyperparameters [3, 19, 29]. EAs are inherently parallelizable in that fitness evaluations can happen concurrently, and which makes EAs scalable and suitable for HPC platforms; using an EA on an HPC platform allows for a larger population to improve optimization. For example, an EA used to tune the hyperparameters for a convolutional neural network used for settlement detection in satellite imagery found better models than a grid-based hyperparameter search on Oak Ridge National Laboratory's Summit supercomputer, and incorporated 551 parallel evaluation tasks per run [6].

The DeePMD-kit program for DNNP training and deployment [31], uses two connected neural networks to learn the energies and forces of an interatomic potential from FP simulation data, usually using DFT as the level of theory. With interfaces in Python and C++, the program makes use of TensorFlow for its neural network framework, as well as custom kernels to optimize calculation of its atomic-level descriptor functions [31, 17]. There are a number of training parameters that can be set to control model training in DeePMD-kit.[1] When using the Deep Potential Smooth Edition (DeepPot-SE) in DeePMD-kit, the model learns a smooth and continuously differentiable potential energy surface, mapping between a local environment within a radial cut-off of each atom to a per-atom energy, such that the sum of atomic energies for a particular three dimensional configuration of atoms corresponds to the total energy for that configuration from the reference DFT data. The gradients of the predicted energies are used to compute the atomic forces through backpropagation and both the reference energies and forces are included in the evaluation of the loss which is minimized during training of the model [17, 32]. For accurate molecular dynamics simulations, it is necessary to have accurate energies and forces, and the correct mathematical relationship between the two. *For this reason, it is not enough to minimize either the energy or force loss alone as the fitness objective for a hyperparameter optimization, and thus a multiobjective optimization approach is required.* While it has been mentioned that DeePMD-kit model training is not extremely sensitive to parameters, neither a detailed sensitivity analysis nor a hyperparameter optimization has been reported, and workers using this program consistently make use of the suggested defaults.

Here we describe the use of the Summit supercomputer at the Oak Ridge Leadership Facility (OLCF) to sample and rapidly optimize the DeePMD-kit model training hyperparameter space for a given dataset, making use of the inherent parallelism in EAs to scale across 100 Summit compute nodes to simultaneously minimize both model energy and force loss with a large-scale deployment of the NSGA-II algorithm for multiobjective optimization, while exploring the space of seven different training hyperparameters, and converge to several interesting candidate solutions on and around Pareto frontiers, while also providing for optimization of time to solution.

[1]https://docs.deepmodeling.com/projects/deepmd/en/master/train/train-input.html

2 METHODOLOGY

Our research objective was to optimize the hyperparameters for a deep-learning-based interatomic energy and force field potential, with our focus being on training potentials for molten aluminum halides. Parameters that are used in DeePMD-kit training and can affect model performance include the learning rate decay, start and stop learning rates, type of activation functions used for the fitting and embedding networks, and the radial cutoffs for inclusion of neighbors in the descriptors, among others described in detail below. Included implicitly in the optimization is also training runtime, with every model training limited to two hours; individuals that do not finish the required number of training steps are counted as "unfit" by the EA.

2.1 System and Software

2.1.1 The Summit supercomputer. The OLCF at the Oak Ridge National Laboratory (ORNL) is an open science computing facility that supports HPC research. The OLCF houses the Summit supercomputer, an IBM AC922 system consisting of 4608 large nodes each with six NVIDIA Volta V100 GPUs and two POWER9 CPU sockets providing 42 usable cores per node [22].

2.1.2 DeePMD training on Summit. The DeePMD program [30] version v2.1.4 was built on OLCF Summit on top of the Open Cognitive Environment (open-ce)[2] version of TensorFlow for ppcle64, using open-ce version 1.5.2-py39-0. Using Summit's 6-GPU nodes, distributed, data parallel training was deployed with Horovod [27], MPI for Python (mpi4py) [9], and MPI [13] libraries (IBM Spectrum MPI Version 10[3]). The embedding network and fitting network size were {25, 50, 100} and {240, 240, 240}, respectively. The tunable prefactors in the loss function were chosen as 0.02, 1000, 1, 1 for p_e^{start}, p_f^{start}, p_e^{limit}, and p_f^{limit}, respectively. These two sets of parameters were fixed and were not included in the EA optimization. Training on Summit's GPUs provides about 65× speedup per node vs. a CPU-only, threaded version, training a potential on approximately 250,000 frames from DFT FPMD in under 2 hours, compared to about 7 days.

2.1.3 Training dataset generation. Training data was generated using the Compute and Data Environment for Science (CADES) facility at ORNL, specifically the Scalable HPC Condos. Datasets consisted of FPMD simulations using the CP2K program [20]. The system consisted of a mixture of molten aluminum and potassium chloride at percentages of 66.7 and 33.3 %, respectively, with 160 atoms and a square box size of side length of 17.84 Å. A single FPMD simulation consisting of over 250,000 frames, simulated at 498 K, was converted to input data formats compatible with DeePMD (energy, force, box values in Numpy arrays) using in-house scripts. These arrays were split into separate datasets after shuffling, and a set of 25% of the frames was withheld for use as the validation set.

2.1.4 The LEAP library and implementation of NSGA-II for DeePMD. The Library for Evolutionary Algorithms in Python (LEAP) was used to implement the software used for our experiments [8], and used LEAP's implementation of NSGA-II [11] for multiobjective optimization support. However, we used an improved version of

[2]https://github.com/open-ce
[3]https://www.ibm.com/docs/en/SSZTET_EOS/eos/guide_101.pdf

ranked-based sorting that yielded a significant speed-up for NSGA-II [4]. We also took advantage of LEAP's support for distributed fitness evaluations that relied on the Dask parallel library for Python [10] to scale our experiments to run on Summit.

2.2 Experimental Setup

The following describes training hyperparameters that were optimized, the ranges of values used for each, and the implementation of the multiobjective optimization algorithm.

2.2.1 Representation. Each individual in a population was a seven-element real-valued vector that corresponded to the following training parameter variables:

start_lr Start learning rate.
stop_lr Stop learning rate.
rcut The hard atomic descriptor radial cutoff distance, in Angstroms.
rcut_smth The extent of the smoothing function for the radial cutoff, in Angstroms.
scale_by_worker The scaling function used when scaling the learning rate by worker during distributed data parallel training, with possible values of {"linear", "sqrt", "none"}.
desc_activ_func Descriptor activation function that maps to one of {"relu", "relu6", "softplus", "sigmoid", "tanh"}.
fitting_activ_func Fitting network activation function that also maps to one of {"relu", "relu6", "softplus", "sigmoid", "tanh"}.

While DeePMD has numerous other hyperparameters that are eligible for tuning, here we start with the above set as they were indicated as worthy of exploration based on initial sensitivity testing and simulation considerations, and to elucidate the effects of some key settings that were not well understood. In particular, the choices of activation functions are not discussed in detail in publications or documentation, and are therefore interesting to explore. Furthermore, the two different radial cutoff parameters—the radial cutoff distance and the smoothing function radius, work together to create a smooth potential, but it is not trivial to understand how to best adjust them for accuracy. While larger cutoff distances will produce more accurate potentials, this will also increase the time to solution for both training and inference. The value for the smoothing function is not as clearly related to accuracy of interactions.

The learning rate decays exponentially, based on the input start and stop learning rates, and the loss function. The loss function is a weighted sum of mean-squared errors of energy and forces, and is weighted by different prefactors which are themselves functions of the decaying learning rates, with the force prefactor dominating the the loss function at the start of training, and decreasing as the training proceeds, and the reverse for the energy loss prefactor. This forces the training to initially principally minimize the force error and then gradually include energy error as an objective as training proceeds.

For distributed data parallel training, large batch sizes can cause optimization problems, which can be ameliorated by scaling learning rate [15]. The default setting in DeePMD-kit for distributed training is a linear learning rate scaling, by the number of GPUs used, regardless of whether this number is small. Linear scaling may not be needed for small batch sizes, and therefore, it may be better to use a reduced scaling factor setting when only 6 GPUs are used for training, which seems to be sufficient for the neural network and dataset sizes used in this paper.

Table 1: Initialization parameters for the experiments: ranges in which randomly created individual gene values were generated, as well as the initial standard deviations used for the Gaussian mutation operator.

hyperparameter	initialization range	mutation standard deviations
start_lr	(3.51e-8, 0.01)	0.001
stop_lr	(3.51e-8, 0.0001)	0.0001
rcut	(6.0, 12.0)	0.0625
rcut_smth	(2.0, 6.0)	0.0625
scale_by_worker	(0.0, 3.0)	0.0625
desc_activ_func	(0.0, 5.0)	0.0625
fitting_activ_func	(0.0, 5.0)	0.0625

Tbl. 1 shows the ranges in which random values were generated when creating individuals for the initial population as well as the starting standard deviations used for Gaussian mutation. Because the overall objective was to minimize the energy and force potential losses during training, fitnesses for each individual were represented by a two element Numpy array.

2.2.2 Decoding individuals prior to evaluation. Normally, real-valued vector representations as shown here are entirely phenotypic in that they can be directly used during fitness evaluation. However, during fitness evaluation, the variables scale_by_worker, desc_activ_func, and fitting_activ_func needed to first be mapped to valid strings. This requires implementing a LEAP decoder that maps those genes (variables) to numerical values by taking the floor of the random float then taking the modulus of the resulting value against the number of possible string values to look up the appropriate string. For example, for a gene value of 5.78 for scale_by_worker that needs to map to one of {"linear","sqrt","none"}, `floor(5.78)%3` would yield 2, so the program would assign the string "none" for that gene. This allows for Gaussian mutation of real-valued genes to satisfy the constraints of mapping to string-valued parameters.

2.2.3 NSGA-II implementation. Although LEAP has a `nsga2()` function that handles the implementation details of NSGA-II, we used the lower-level functions that implement LEAP's `nsga2()` to write our own version of NSGA-II that would more conveniently allow for some tailoring of mutation. In particular, with each new generation, the vector of standard deviations, as shown in column three of Tbl. 1, was multiplied by .85. While originally, this process of annealing was within the context of the 1/5 success rule [16], we chose not to implement the 1/5 success rule to adjust the annealing rate, as sensitivity tests prior to our formal experiments indicated that this was not necessary. Listing 1 shows the LEAP reproduction operator pipeline we used that would create the offspring for each generation. Line 2 is the current prospective parent population that acts as the source with a corresponding sink in line 9 that "pulls" individuals as needed through the intervening operators, which are

```
1  offspring = \
2      pipe(parents,
3           ops.random_selection,
4           ops.clone,
5           mutate_gaussian(
6               std=context['std'],
7               expected_num_mutations='isotropic
    ',
8               hard_bounds=DeepMDRepresentation.
    bounds),
9           eval_pool(client=client,
10                    size=len(parents)),
11          rank_ordinal_sort(parents=parents),
12          crowding_distance_calc,
13          ops.truncation_selection(size=len(
    parents),
14                                   key=lambda x
    : (-x.rank,
15
       x.distance)))
```

Listing 1: The LEAP reproduction operator pipeline used to create offspring for each generation using LEAP's NSGA-II pipeline operators, rank_ordinal_sort and crowding_distance_calc.

Python generator functions. That is, we need as many evaluated offspring as parents, so `eval_pool()` pulls in as many offspring as parents, and once it has accumulated that many, it fans them out via the Dask-based parallel deployment scheme (discussed in detail in §2.2.5) to workers for parallel fitness evaluation. For each offspring, a parent is randomly selected, cloned, and then Gaussian mutation is applied to all the genes given the current set of standard deviations for each gene found in the global variable `context['std']`. This vector of standard deviations is multiplied by .85 after the offspring are returned from this pipeline to anneal the mutation per generation (`context` is a LEAP global Python dictionary used to maintain a run-time state accessible by pipeline operators). Lines 11–15 are the NSGA-II operators for doing the rank sorting and crowding distance calculation, followed by performing truncation selection based on rank and crowding distance. The offspring returned by the truncation selection will be the prospective parents for the next generation. `pipe()` is provided by the third party Python toolkit, toolz.[4]

2.2.4 *Steps taken for fitness evaluation.*

Evaluating individuals in parallel on the Summit supercomputer entailed a multistep workflow. The steps for evaluating a single individual was as follows:

(1) An individual's genome of the seven element real-valued array was decoded, including mapping floating point values to strings as described in §2.2.2.
(2) A space in which training would happen was created:
   (a) Each individual was automatically assigned a UUID when created.
   (b) A sub-directory was created named after that UUID, and DeePMD training was executed there.
(3) An `input.json` file that contained the DeePMD hyperparameters and run-time configuration was created:
   (a) A file containing JSON-formatted input template was read in.
   (b) Using the Python Standard Library `string.Template` mechanism, variable substitution was performed with that JSON-formatted template using the decoded gene values from the individual.
   (c) The updated `input.json` file was written to the UUID-named run directory.
(4) The fitness values were assigned based on training results:
   (a) A `subprocess()` call was used to invoke `dp`, the DeePMD training executable.
   (b) Successful DeePMD training would produce a model and training statistics in an output file, `lcurve.out`.
   (c) The last values of the rmse_e_val and rmse_f_val columns were read from `lcurve.out`, which correspond to the energy and force validation loss results, and returned in a numpy two-element array as the fitness.

There are occasions when an error will occur during evaluation. For example, the `subprocess()` call could exceed the two hour limit, thus raising a `TimeoutError` exception. It is also possible that the unique combination of hyperparameter values will cause training to fail. Moreover, a hardware fault or other type of node failure could also foil training. In all these cases, both values of the two-element fitness are assigned MAXINT. In this way, the optimization is also implicitly including runtime performance, as individuals that require too much time to train or cause other runtime problems will be eliminated. As an aside, the LEAP `DistributedIndividual` class, which we originally used, catches exceptions that are raised during evaluation and assigns an IEEE 754 NaN as the fitnesses. However, NSGA-II sorts all individuals by their fitnesses, and sorting values that include NaNs yields undefined behavior. Therefore we implemented a subclass of `DistributedIndividual` that overrode the default exception handling behavior and assigned MAXINT as fitnesses instead. Since both fitness objectives were minimization problems, this ensured that NSGA-II rank sorting worked correctly.

2.2.5 *Dask-related setup and configuration.*

We relied on LEAP's use of the Dask parallel library for Python to implement parallel evaluations for our Summit supercomputer runs. Dask has three types of components: a scheduler, workers, and a client. The client submits tasks to the scheduler, which assigns tasks to workers and ensures that their results are communicated back to the client. When a Summit batch job begins running, it is first assigned a batch node to begin executing the batch submission script. Within the batch script will be one or more `jsrun` statements that assign computing node resources also allocated to the running job to user specified executables that then begin running. We configured our batch script to run our experiments by launching the Dask scheduler and Dask workers on the batch node with each Dask worker assigned an entire Summit node to perform the DeePMD training and the fitness evaluation described in §2.2.4. A Dask client was to assign the fitness evaluation tasks, and also ran on the assigned batch node. For prior work on Summit using LEAP, we used a single `jsrun` call to launch all Dask workers directly on compute nodes [24, 7, 12]. In this case, because DeePMD uses `MPI_init` within Horovod-based distributed training in TensorFlow to coordinate training that occurs on each of the GPUs, the system is left in a state

[4]https://toolz.readthedocs.io/en/latest/api.html#toolz.functoolz.pipe

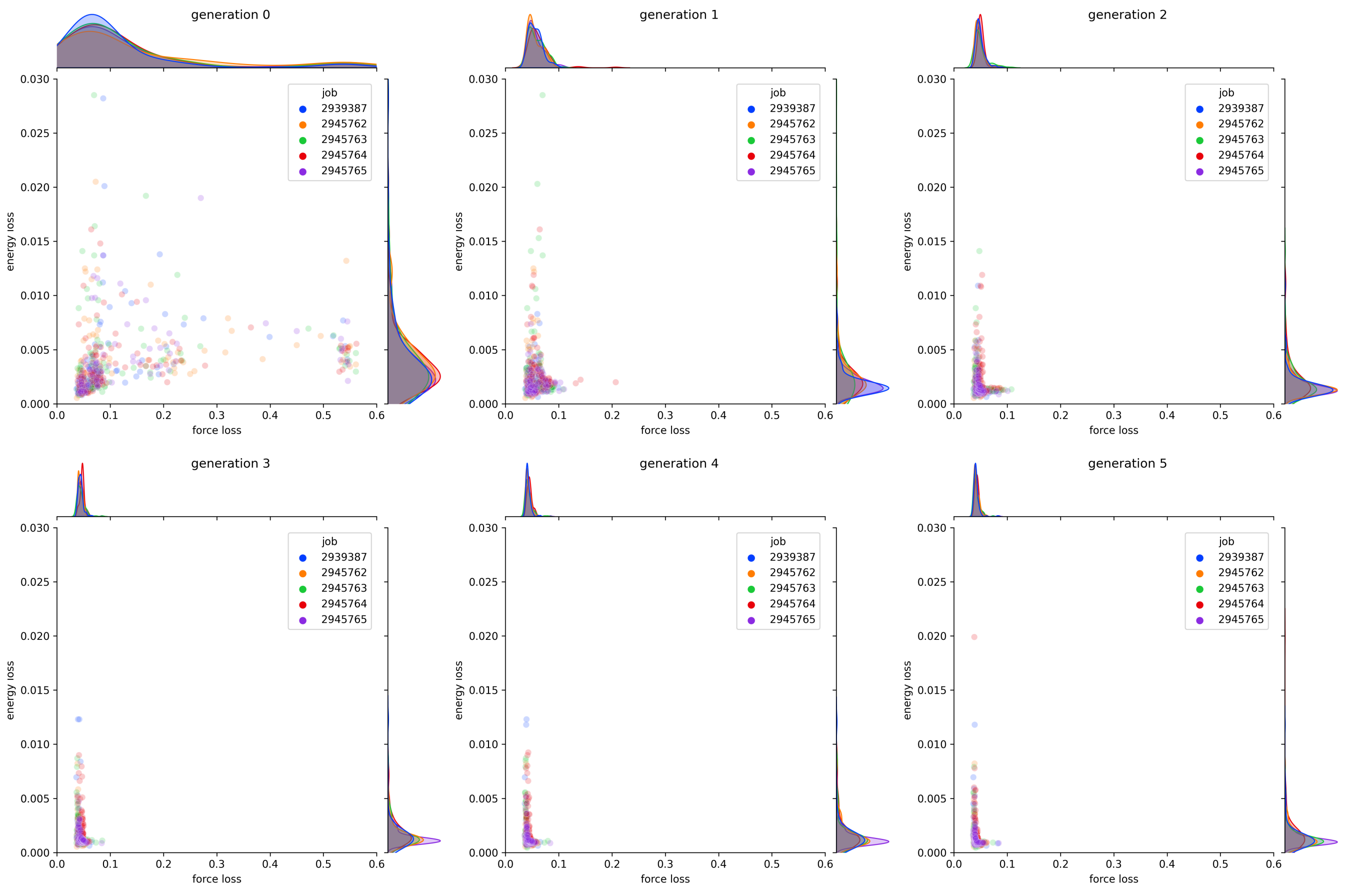

Figure 1: Level plots showing energy vs. force losses after each generation for all models over five independent EA runs, up to six generations. Some outliers are cropped from generation 0 for visual clarity. Units for energy loss are in eV/atom and force loss in eV/Å.

such that a subsequent fitness evaluation using `MPI_init` was not possible without a new `jsrun` command, therefore the workflow was reconfigured to start workers on the batch node instead of on a compute node, and with separate `jsrun` calls for the each DeePMD training launched with the subprocess call described in §2.2.4.

Other adjustments to facilitate use of Dask on Summit for this application included the specification of `\tmp` for use by Dask workers creating temporary directories to save state instead of the default home directory which is mounted as read-only during a Summit run. Also, Dask uses Bokeh[5] to maintain a dashboard showing the run state of the scheduler and workers; this dashboard is not practical given the sheer number of workers and the run environment on the supercomputer, we therefore disabled this dashboard during our runs. Dask also has “nannies" that are associated with a Dask worker; if the nanny observes that its worker has prematurely terminated, the nanny will restart the worker. Worker failures may be due to hardware failures, in which case a restart will not correct anything. We found it best to disable nannies, let workers fail, and have the scheduler reassign tasks to other workers in those scenarios.

[5]https://bokeh.org/

Other runtime parameters include the parent and offspring population sizes, set to the number of Summit nodes allocated to the job (100 for all experiments), and the maximum wall clock time allocated for the jobs, which was 12 hours. DeePMD users often report training for hundreds of thousands to one million steps. We found during sensitivity runs that 40,000 steps was sufficient to obtain minimized losses sufficiently converged to provide both acceptable chemical accuracy and a time to solution amenable to converging the EAs within our computational cost limitations. We therefore used this number of training steps for all runs.

3 RESULTS AND DISCUSSION

3.1 Convergence of the EA

Fig. 1 shows level plots displaying losses for energy (in eV/atom) vs. force (in eV/Å) combined over the five independent EA runs for the first 6 generations. Generation 0 was the initial random population and already shows a cluster of values close to the origin. Most individuals that were scattered away from the origin in the initial random population are eliminated within the first EA step that produced generation 1. From that generation forward there are smaller changes in the loss distributions, with distributions between the last three runs being similar, indicating convergence of the algorithm. Several outliers with force losses greater than 0.6 or energy loss greater than 0.03 found in generation 0 were culled for visual clarity in Fig. 1. A total of 3500 DeePMD model training runs were performed over seven generations. For practical use, five separate EA deployments should not be required; however this number is orders of magnitude smaller than a brute-force grid search with 10 grid points per parameter.

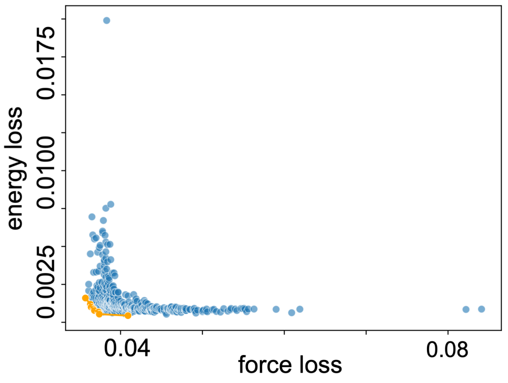

Figure 2: Pareto frontier, shown as connected orange dots, from the aggregated last generations of all six Summit runs, shows one distinct clusters of non-dominating individuals. Units for energy loss are in eV/atom and force loss in eV/Å.

3.1.1 The Final Pareto Frontier of Solutions. Fig. 2 shows the Pareto frontier calculated from the aggregate of the last generation of all five runs. The frontier comprises 8 points clustered close to the origin. Tbl. 2 displays the forces and energies for these solutions. It should be noted that the exact frontier may be too strict for selection of the set of optimal solutions, as small floating point differences in energies or forces will not be physically relevant in the simulations. Therefore, the region around this frontier can be considered for selection for optimized solutions, which can be further filtered. Using this frontier region, we are able to further choose desired parameter set solutions that may correspond to our specific needs, with the knowledge that the parameter space has been sufficiently explored and a set of non-dominating optimal solutions has been presented. We discuss chemistry-specific requirements for solutions below.

Table 2: Force and energy values for all solutions found exactly on the Pareto frontier

solution	force error (eV/Å)	energy error (eV/atom)
1	0.0357	0.0016
2	0.0363	0.0012
3	0.0364	0.0010
4	0.0367	0.0009
5	0.0368	0.0008
6	0.0373	0.0007
7	0.0374	0.0005
8	0.0409	0.0004

3.2 Considerations of Chemical Accuracy

For training a molecular potential such that errors are within the precision of the reference DFT, the trained network using should yield energy and force errors of below about 0.004 eV/atom and 0.04 eV/Å, respectively. For performing molecular dynamics, force accuracy is extremely important, because each time step in the dynamics time series uses force values to propagate the integrator, which integrates the equations of motions numerically. Force errors compound as the time series progresses, driving a simulation farther and farther away from correct values the longer it runs [14]. Inspection of the Pareto frontier shows that the multiobjective optimization offers one solution that would not be exactly suitable for use as a potential model with the above stated requirements for force error, by about 0.001. While this is potentially an insignificant difference, it may be beneficial to chose certain solutions from the Pareto frontier that best satisfy the chemical requirements.

It is also notable that, while there is a well-defined cluster of force losses below 0.06 eV/Å, there is no clear barrier distinguishing those above the desired 0.04 eV/Å from those below that value, and likewise along the y-axis for energy losses below 0.008 eV/atom. What this indicates is that there is no direct relationship between the boundaries around the minima of the hyperparameter search space and the desired accuracy according to chemical considerations, and therefore it could be relatively easy to produce a chemically inaccurate model. Therefore, if our scheme for multiobjective optimization is used for automating hyperparameter tuning without a further selection of solutions from the Pareto frontier using chemical criteria, it is possible that inaccurate models may result. It should be noted, however, that the root mean squared errors of forces from empirical models compared to reference condensed-phase DFT calculations can exceed 1.4 eV/Å [33].

In addition, the local region around this exact set of frontier points is also ripe for selecting optimal hyperparameter combinations, with the consideration of the chemical system and potentially other factors, such as time to solution, by selecting for chemically accurate results. Fig. 3 displays the hyperparameters for each solution found in our final solution dataset, the combined last generations from all runs, in a parallel coordinates plot. The blue colored lines indicate the chemically accurate solutions, using limits on energy and force errors of 0.004 eV/atom and 0.04 eV/Å, respectively. From this plot, it can be seen that when considering chemically accurate solutions, the radial cutoff setting tends to be in the higher ranges, with no accurate solution having an rcut below 8.5 Å. It is interesting to note that for these chemically complicated, charged molten salt systems, which may contain longer-range atomic interactions, larger values for these radial parameters seem to be selected.

The value for the smoothing distance varies across the range of possibilities, but with an increased density below 4.5 Å. These two parameters have a direct connection to the physics of the system, representing an acceptable distance away from each atom at which to ignore interactions from other atoms. It is not generally known *a priori* what important contributions to forces and energies are actually found at what distances around each atom in each system, and the relationship of the smoothing function decay to the energy and force errors is also not clear. Therefore, an automated, exhaustive, and intelligent scheme like the one presented here can

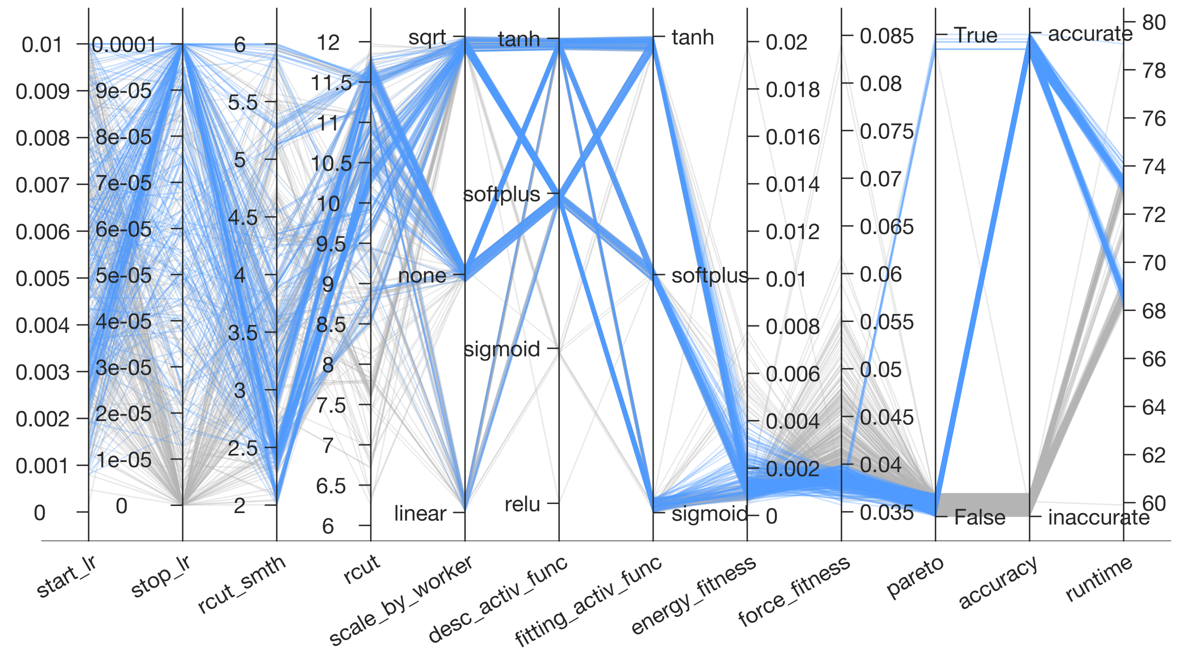

Figure 3: Parallel coordinates plot showing the parameters found for each of the final solutions in the last generation, with those providing acceptable chemical accuracy colored in blue, and others colored in grey. Runtime (in minutes) for the training, energy and force loss (fitness), and whether or not the solution is on the exact Pareto frontier are also indicated.

be especially useful for setting these parameters optimally. The relationship of the smoothing parameter with the model accuracy is complex; not all solutions using a lower value for the smoothing distance correspond to a high radial cutoff, and the activation functions and learning corresponding to these solutions also vary. It should be noted that we included the program default settings in the initialization ranges for all hyperparameters tested except the smoothing distance; for this one setting we used a lower bound of 2 Å, based on previous work with molten salt systems, as opposed to the default value of 0.5.

Table 3: Parameter values for three selected chemically-accurate solutions found in the last NSGA-II generations across the five runs, showing the solution with lowest force loss, lowest energy loss, and lowest runtime.

hyperparameter	solution 1	solution 2	solution 3
start_lr	0.0047	0.0058	0.01
stop_lr	0.0001	0.0001	2e-05
rcut	11.32	10.10	11.32
rcut_smth	2.42	2.11	2.43
scale_by_worker	none	none	none
desc_activ_func	tanh	softplus	tanh
fitting_activ_func	tanh	tanh	tanh
runtime (min.)	68.7	74.1	68.1
energy loss (eV)	0.0016	0.0005	0.0019
force loss (eV/Å)	0.0357	0.0374	0.0370

Runtimes for all training runs in the combined last generation solution set are under 80 minutes, and no runs for any generations crossed beyond this value. Therefore, no runtime optimization was necessary or performed for this dataset. However, in the previous generations, very short runtimes were found, corresponding to failed training tasks, for a total of 25 such instances, spread across all five jobs. No such failures were found in the last generation for any jobs, indicating that an optimization away from potentially fatal configuration errors may have occurred.

For chemically accurate solutions, start learning rate has a heavier distribution between 0.004 and 0.002; the default setting is 0.001. While stop learning rate default is 1e-08, all chemically accurate solutions are found above 1e-05 and cluster around 0.001. The learning rate scaling function results were also informative, and helped to validate our hypothesis that learning rate scaling with distributed data parallel training may not be required when the total number of GPUs is only 6 and the total batch size (training_batch_size × 6) does not reach into the hundreds as in large scale parallel training schemes. We can see that scaling by the square root of the number of workers and no scaling at all can provide excellent training results, and in fact, more chemically accurate solutions are obtained this way. Another result to note is that both relu activation functions for the fitting network have dropped out completely from the final solution, and the sigmoid activation function for the descriptor network is not included in any chemically accurate solutions. Softplus and sigmoid for the fitting activation function provided excellent results, and softplus also performs well as the activation function for the descriptor network. The default setting for both the fitting and descriptor network activation functions is the tanh function.

Tbl. 3 displays the hyperparameters for chemically accurate models with the lowest energy loss, force loss, and runtime found in the last generation across the five runs. The first two models are found in the Pareto frontier, while the lowest runtime corresponds to a model that is not in the frontier.

4 CONCLUSIONS AND FUTURE WORK

Here we showed the value of automated, parallel hyperparameter tuning using multiobjective evolutionary algorithms for training of DNN interatomic physical potentials for simulations in chemistry. For these model potentials, the relationship between energy and force must be incorporated in the EA fitness. We have shown that deployment of the NSGA-II algorithm works well to address this challenge. While we only optimized hyperparameters here, model fidelity may also be further improved by incorporating neural architecture searching on the two DeePMD neural networks, and the method can be extended to other new DNNP programs, which are rapidly emerging.

ACKNOWLEDGMENTS

Thanks to Steven Young and Guojing Cong of the Oak Ridge National Laboratory, Travis Johnston of Striveworks, and Josh Romero from NVIDIA for their insightful feedback and assistance. This work was supported by the Office of Materials and Chemical Technologies within the Office of Nuclear Energy, U.S. Department of Energy, and used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725

REFERENCES

[1] Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P Mailoa, Mordechai Kornbluth, Nicola Molinari, Tess E Smidt, and Boris Kozinsky. 2022. E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature Communications*, 13, 1, 2453.

[2] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13, 2.

[3] Erik Bochinski, Tobias Senst, and Thomas Sikora. 2017. Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms. In *2017 IEEE International Conference on Image Processing (ICIP)*, 3924–3928. DOI: 10.1109/ICIP.2017.8297018.

[4] Bogdan Burlacu. 2022. Rank-based non-dominated sorting. *arXiv preprint arXiv:2203.13654*.

[5] Rajni Chahal, Santanu Roy, Martin Brehm, Shubhojit Banerjee, Vyacheslav Bryantsev, and Stephen T Lam. 2022. Transferable deep learning potential reveals intermediate-range ordering effects in LiF–NaF–ZrF_4 molten salt. *JACS Au*.

[6] Mark Coletti, Dalton Lunga, Jeffrey K. Bassett, and Amy Rose. 2019. Evolving larger convolutional layer kernel sizes for a settlement detection deep-learner on summit. In *Workshop for Deeplearning for Supercomputers*. The International Conference for High Performance Computing, Networking, Storage, and Analysis.

[7] Mark A. Coletti, Shang Gao, Spencer Paulissen, Nicholas Quentin Haas, and Robert Patton. 2021. Diagnosing autonomous vehicle driving criteria with an adversarial evolutionary algorithm. In *Proceedings of the 2021 Genetic and Evolutionary Computation Conference Companion* (GECCO '21). Association for Computing Machinery, Lille, France, 301–302. DOI: 10.1145/3449726.3459573.

[8] Mark A. Coletti, Eric O. Scott, and Jeffrey K. Bassett. 2020. Library for evolutionary algorithms in python (leap). In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion* (GECCO '20). Association for Computing Machinery, Cancún, Mexico, 1571–1579. ISBN: 9781450371278. DOI: 10.1145/3377929.3398147.

[9] Lisandro Dalcin and Yao-Lung L. Fang. 2021. Mpi4py: status update after 12 years of development. *Computing in Science & Engineering*, 23, 4, 47–54. DOI: 10.1109/MCSE.2021.3083216.

[10] Dask Development Team. 2016. *Dask: Library for dynamic task scheduling*. https://dask.org.

[11] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: nsga-ii. *IEEE transactions on evolutionary computation*, 6, 2, 182–197.

[12] Alexander Fafard, Jan Van Aardt, Mark Coletti, and David L Page. 2020. Global partitioning elevation normalization applied to building footprint prediction. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13, 3493–3502.

[13] Message Passing Interface Forum. 2015. *MPI: a message passing interface standard: version 3.1; Message Passing Interface Forum, June 4, 2015*. University of Tennessee.

[14] D. Frenkel and B. Smit. 2001. *Understanding Molecular Simulation: From Algorithms to Applications. Computational science*. Elsevier Science. ISBN: 9780080519982. https://books.google.com/books?id=5qTzldS9ROIC.

[15] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

[16] 1997. *Handbook of evolutionary computation*. CRC Press. Chap. Evolution strategies, B1.3:2.

[17] Weile Jia, Han Wang, Mohan Chen, Denghui Lu, Lin Lin, Roberto Car, E Weinan, and Linfeng Zhang. 2020. Pushing the limit of molecular dynamics with ab initio accuracy to 100 million atoms with machine learning. In *SC20: International conference for high performance computing, networking, storage and analysis*. IEEE, 1–14.

[18] Ryosuke Jinnouchi, Ferenc Karsai, and Georg Kresse. 2019. On-the-fly machine learning force field generation: application to melting points. *Physical Review B*, 100, 1, 014105.

[19] Franklin Johnson, Alvaro Valderrama, Carlos Valle, Broderick Crawford, Ricardo Soto, and Ricardo Ñanculef. 2020. Automating configuration of convolutional neural network hyperparameters using genetic algorithm. *IEEE Access*, 8, 156139–156152. DOI: 10.1109/ACCESS.2020.3019245.

[20] Thomas D. Kühne et al. 2020. Cp2k: an electronic structure and molecular dynamics software package - quickstep: efficient and accurate electronic structure calculations. *The Journal of Chemical Physics*, 152, 19, 194103. DOI: 10.1063/5.0007045.

[21] Lizhi Liao, Heng Li, Weiyi Shang, and Lei Ma. 2022. An empirical study of the impact of hyperparameter tuning and model optimization on the performance properties of deep neural networks. *ACM Trans. Softw. Eng. Methodol.*, 31, 3, Article 53, 40 pages. DOI: 10.1145/3506695.

[22] ORNL Leadership Computing Facility. 2019. Summit: america's newest and smartest supercomputer. (2019). https://www.olcf.ornl.gov/for-users/system-user-guides/summit/summit-user-guide/#system-overview.

[23] Santanu Roy et al. 2021. A holistic approach for elucidating local structure, dynamics, and speciation in molten salts with high structural disorder. *Journal of the American Chemical Society*, 143, 37, 15298–15308.

[24] Eric O. Scott, Mark Coletti, Catherine D. Schuman, Bill Kay, Shruti R. Kulkarni, Maryam Parsa, and Kenneth A De Jong. 2021. Avoiding excess computation in asynchronous evolutionary algorithms. In *Proceedings of the 20th UK Workshop on Computational Intelligence*. Aberystwyth University, (to be printed).

[25] Ada Sedova, Russ Davidson, Mathieu Taillefumier, and Wael Elwasif. 2022. HPC molecular simulation tries out a new GPU: experiences on early AMD test systems for the Frontier supercomputer. In *Proceedings of 2022 Cray User Group Meeting, Monterey, CA, USA, May 2022*.

[26] Ada Sedova, John D. Eblen, Reuben Budiardja, Arnold Tharrington, and Jeremy C. Smith. 2018. High-performance molecular dynamics simulation for biological and materials sciences: challenges of performance portability. In *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 1–13. DOI: 10.1109/P3HPC.2018.00004.

[27] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*.

[28] Oliver T Unke, Stefan Chmiela, Huziel E Sauceda, Michael Gastegger, Igor Poltavsky, Kristof T Schütt, Alexandre Tkatchenko, and Klaus-Robert Müller. 2021. Machine learning force fields. *Chemical Reviews*, 121, 16, 10142–10186.

[29] Amala Mary Vincent and P Jidesh. 2023. An improved hyperparameter optimization framework for automl systems using evolutionary algorithms. *Scientific Reports*, 13, 1, 4737.

[30] Han Wang, Linfeng Zhang, Jiequn Han, and Weinan E. 2018. Deepmd-kit: a deep learning package for many-body potential energy representation and molecular dynamics. *Computer Physics Communications*, 228, 178–184. https://github.com/deepmodeling/deepmd-kit.

[31] Han Wang, Linfeng Zhang, Jiequn Han, and E Weinan. 2018. DeePMD-kit: a deep learning package for many-body potential energy representation and molecular dynamics. *Computer Physics Communications*, 228, 178–184.

[32] Tongqi Wen, Linfeng Zhang, Han Wang, E Weinan, and David J Srolovitz. 2022. Deep potentials for materials science. *Materials Futures*.

[33] Ying Yuan, Zhonghua Ma, and Feng Wang. 2021. Development and validation of a dft-based force field for a hydrated homoalanine polypeptide. *The Journal of Physical Chemistry B*, 125, 6, 1568–1581.