

Neural Network and ReaxFF comparison for Au properties - Supporting information

Jacob R. Boes¹, Mitchell C. Groenenboom², John A. Keith^{*2}, and John R. Kitchen^{†1}

¹Department of Chemical Engineering, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, USA

²Department of Chemical and Petroleum Engineering, University of Pittsburgh, Benedum Hall, 3700 O'Hara Street, Pittsburgh, PA 15261, USA

Contents

1	Introduction	2
2	Methods	3
2.1	Density Functional Theory	3
2.1.1	Generating the ASE database from DFT calculations	3
2.1.2	Using keywords to get information from the database	4
2.2	Reactive Force Field	7
2.2.1	Comparing Force Fields with and without 3-Body Terms	7
2.2.2	Monte-Carlo Force-Field Optimization Process (MCFFopt)	9
2.2.3	Adding ReaxFF energies to the database	11
2.2.4	Manuscript figure fig-reax-train	12
2.2.5	Manuscript figure fig-reax-vaild	13
2.3	Neural Network	14
2.3.1	Adding BPNN energies to the database	14
2.3.2	Manuscript figure fig-neural-train	14
2.3.3	Manuscript figure fig-neural-valid	16
3	Results	17
3.1	Bulk properties	17
3.1.1	Manuscript figure fig-bulk-eos	17
3.1.2	BCC and HCP equations of state	18



^{*}jakeith@pitt.edu

[†]jkitchen@andrew.cmu.edu

3.1.3	Full equations of state	20
3.1.4	Manuscript figure fig-vacancy-formation	21
3.1.5	Structural relaxation of ≈ 0.015 vacancies/atom	22
3.1.6	Manuscript figure fig-vacancy-diffusion	23
3.2	Surface calculations	24
3.2.1	Manuscript figure fig-full-diffusion	24
3.2.2	Manuscript figure fig-barrier-residuals	25
3.2.3	Figures of individual fcc(100) diffusion pathways	26
3.2.4	Manuscript figure fig-111-slipping	28
3.2.5	Additional slipping barriers	29
3.3	Cluster predictions	33
3.3.1	Manuscript figure fig-6atom-md	33
3.3.2	6-atom MD simulations with 2000 data points	35
3.3.3	Manuscript figure fig-38atom-minima	37
4	TOC	38

1 Introduction

This document contains supporting data for the work, "Neural Network and ReaxFF comparison for Au properties." Much of the code utilized to produce this work is recorded here for reproducibility and transparency. Much of the molecular simulation code is produced using tools from the Atomic Simulation Environment (ASE) python modules: <https://wiki.fysik.dtu.dk/ase/>. This includes all of the databasing techniques, which are conducted through ASEs database module: <https://wiki.fysik.dtu.dk/ase/ase/db/db.html#module-ase.db>.

Due to the large number of calculations, data utilized in this work is stored in SQLite format (.db files) for ease of use. These data files are embedded into the PDF and can be accessed through the following links: Calculations specific to the 6 atom MD simulations performed in the paper:  (double-click to open). All other results:  (double-click to open). Examples of using these data files are provided in the following sections.

Large portions of the supporting information file function as demonstrations of how the data can be accessed and manipulated from these database files. For ease of search-ability the supporting information file is organized into sections which reflect the organization of the manuscript. That way, references from the manuscript can be directly referenced in the corresponding section of the supporting information file. The code used to generate the figures in each section of the manuscript are also included.

The supporting information document was prepared in org-mode (<http://orgmode.org>) syntax, which was subsequently exported to L^AT_EX and converted to a PDF. Briefly, org-mode is a plain editing format that enables intermingling of text, code and figures, with markup for typical document elements such as headings, links, tables, etc. . . . , and arbitrary inclusion of L^AT_EX for equations. With the Emacs editor, the code in an org-mode document can be executed in place, and the output captured in the document. For example, tables can be generated by code, or the code for generating a figure can be embedded in the document. The

data in tables can be used as input to other code blocks in the document as well. Org-mode enables selective export of the content to various formats including html, L^AT_EX, and PDF. These features, and many others, make org-mode a convenient platform for reproducible research, where all of the steps leading to conclusions drawn in the work can be documented in one place, but where it may be desirable not to show all the details in every view. For example, in an exported manuscript where code should usually not be visible, or supporting information document such as this one where it is desirable to see the code. Nevertheless, it may still be valuable to go back to the source, for example, to figure out how some analysis was done, especially if all the code is not exported.

The org-mode source for this document can be found here: .

2 Methods

2.1 Density Functional Theory

All DFT calculations were performed using VASP (<https://www.vasp.at>). Due to the large amount of data produced from each individual VASP calculation, the results from each calculation are summarized in the databases discussed in the introduction. The following section demonstrates how these database can be searched for easy access to each of the 9972 calculation referred to in the manuscript.

2.1.1 Generating the ASE database from DFT calculations

The VASP calculation directories are too large to be directly incorporated into the supporting information file, but critical information from these files can be organized into an ASE database and embedded into this PDF (See introduction for how to access these files). The ASE database is an extremely valuable tool and others may find it useful to know how we implement it into this work. For that reason the Python code used to generate the database is shown below for demonstrative purposes. Of course, this code is dependent upon files local to the Kitchen group, and can not be directly utilized *unlike* the other samples of code provided in this document.

```

1  import os, sys
2  from ase.io.trajectory import Trajectory
3
4  # Functions produced by Jacob Boes for work in computational catalysis
5  # These are freely available at: https://github.com/jboes/jbtools.git
6  import jbtools.gilgamesh as jb
7
8  # JASP is a wrapper used to streamline VASP calculation in VASP.
9  # More information can be found: https://github.com/jkitchin/jasp.git
10 from jasp import *
11 JASPRC['restart_unconverged'] = False
12
13 # We want to collect enegies and postions for all ionic steps in the
14 # geometry minimization. So first we compile a list of out.traj files.
15 for r, d, f in os.walk('DFT/'): # The location of the DFT calculations
16     if 'OUTCAR' in f and 'XDATCAR' in f:
17         try:
18             with jasp(r) as calc:
19                 # Create a trajectory file

```

```

20         jb.compile_trajectory(calc)
21
22         # Access the created file
23         traj = Trajectory('out.traj')
24         n = len(traj)
25
26         # Add the contents to the database
27         for i, atoms in enumerate(traj):
28
29             jb.makedb(calc,
30                       atoms=atoms,
31                       dbname='~/research/neural/data.db',
32                       keys={'traj': int(n-i-1)})
33     except:
34         print(r)
35         print(sys.exc_info()[0])
36         print('')

```

2.1.2 Using keywords to get information from the database

We organize information from the VASP calculations using descriptive keywords, which define the purpose of each calculation. A full list of the keywords used to categorize each calculation can be found in the results produced from the code block below.

```

1  from ase.db import connect
2
3  # Connect to the ASE database
4  db = connect('data.db')
5
6  # Selects all calculations in the database
7  data = db.select(['traj=0'])
8
9  # Create a dictionary of all the keys
10 keys, cnt = {}, 0
11 for entry in data:
12     cnt += 1
13     for k, v in entry.key_value_pairs.iteritems():
14
15         # Add all possible values to each dictionary
16         if k in keys:
17             keys[k] += [v]
18         else:
19             keys[k] = [v]
20
21 # Iterate through each key and print the values
22 print('{0:15s} {1:15s} {2} calculations total'.format('keyword', 'value', cnt))
23 print('-----')
24 for k, v in keys.iteritems():
25     vals = list(set(v))
26
27     # Only print the first 5 values
28     if len(vals) <= 5:
29         val = ", ".join(str(e) for e in vals)
30         print('{0:15s}: {1}'.format(k, val))
31     else:
32         val = ", ".join(str(e)[:5] for e in vals[:5])
33         print('{0:15s}: {1}, etc...'.format(k, val))

```

keyword	value	896 calculations total
surf	: fcc	

```

xc                : PBE, PW91
image             : 0, 1, 2, 899, 4, etc...
NEB               : False, True, initial, final
site              : tetrahedral, octahedral
encut             : 300.0, 350.0
cluster           : neutra, icosah, plane, amorp, octah, etc...
ibz_kpts          : 1
nbands            : 250
type              : vacancy, interstitial
miller            : 100, 111
ediff             : 1e-07, 1e-05
concentration      : 0.125, 0.015, 0.037, 0.062, 0.009, etc...
reax_energy       : -12.7, -154., -5.31, -8.34, -103., etc...
fit               : db2-to6atom, db2-to13atom, wo-natom6
lattice           : primitive, cubic
ediffg            : -0.05, -0.02
factor            : 0.875, 1.0, 2.0, 0.99, 1.2, etc...
converged         : True
config            : 0, 1, 2, 3, 4, etc...
kpt1              : 1, 4, 5, 6, 7, etc...
kpt3              : 1, 2, 4, 5, 6, etc...
kpt2              : 1, 4, 5, 6, 7, etc...
relaxed           : True
bulk              : sc, hcp, fcc, diam, bcc
ibrion            : -1
gga               : None
neural_energy     : -11.6, -19.2, -201., -147., -6.27, etc...
vacuum            : 10.0
group             : kitchin
post              : minima
structure         : bulk, cluster, surface
diffusion         : slipping, single1, dimer2, step1
volume            : 19.67, 122.5, 130.3, 15.07, 5639., etc...
train_set         : False, True
strain            : xyz
fault             : stacking, twinning
traj              : 0
fermi             : -4.43, 1.016, 7.381, -4.80, -3.37, etc...
path              : /home, /home, /home, /home, /home, etc...
calc_time         : 0.0, 199.6, 26279, 1494., 222.2, etc...

```

NOTE: NEB calculators do not store INCAR, KPOINT, or POTCAR parameters, as these files are kept in the parent directory. Keys generated from these files are place holders, and thus are not correct for NEB calculations. i.e. there are no PW91 calculations, the value was simply not recorded.

The database is broken into three main categories: bulk, surface, and cluster calculations. These categories are organized by the correspondingly named key. For example, to find all bulk calculations, one would search for the 'bulk' key:

```

1  from ase.db import connect
2
3  db = connect('data.db')
4
5  # Now we select only bulk calculations
6  data = db.select(['bulk'])
7  # 'bulk' = all bulk calculations
8  # 'surf' = all surface calculations
9  # 'cluster' = all cluster calculations
10
11 # And again, we print all the possible keys
12 keys, cnt = {}, 0
13 for entry in data:
14     cnt += 1
15     for k, v in entry.key_value_pairs.iteritems():
16
17         if k in keys:
18             keys[k] += [v]
19         else:
20             keys[k] = [v]
21
22 print('{0:15s} {1:15s} {2} calculations total'.format('keyword', 'value', cnt))
23 print('-----')
24 for k, v in keys.iteritems():
25     vals = list(set(v))
26
27     if len(vals) <= 5:
28         val = ", ".join(str(e) for e in vals)
29         print('{0:15s}: {1}'.format(k, val))
30     else:
31         val = ", ".join(str(e)[:5] for e in vals[:5])
32         print('{0:15s}: {1}, etc...'.format(k, val))

```

keyword	value	905 calculations total
xc	: PBE, PW91	
image	: 0, 1, 2, 3, 4	
NEB	: False, True, initial, final	
site	: tetrahedral, octahedral	
encut	: 350.0	
strain	: xyz	
ediff	: 1e-07, 1e-05	
concentration	: 0.125, 0.015, 0.037, 0.062, 0.009, etc...	
reax_energy	: -22.5, -195., -5.31, -2.38, -82.6, etc...	
train_set	: False, True	
lattice	: primitive, cubic	
factor	: 0.875, 1.0, 2.0, 0.99, 1.2, etc...	
type	: vacancy, interstitial	
kpt1	: 1, 4, 5, 6, 7, etc...	
kpt3	: 1, 2, 4, 5, 6, etc...	
kpt2	: 1, 4, 5, 6, 7, etc...	

```

relaxed      : True
volume       : 231.1, 544.7, 19.67, 122.5, 130.3, etc...
gga          : None
neural_energy : -11.6, -19.2, -87.2, -5.40, -87.3, etc...
group        : kitchin
structure     : bulk
bulk         : sc, hcp, fcc, diam, bcc
fault        : stacking, twinning
traj         : 0, 1, 2, 3, 4, etc...
fermi        : 7.381, 13.02, 6.607, -4.45, 11.92, etc...
calc_time    : 0.0, 23.40, 19.04, 15.0, 20.76, etc...

```

This can also be done for surface and cluster calculations as noted in the comments above. Information about specific data sets can be found in the following section.

2.2 Reactive Force Field


The reactive force field produced in this work utilizes 3-body interaction as shown in Table 1. The ReaxFF itself can be opened using the following attachment:  (double-click to open).

Table 1: 3-body parameters used to generate the ReaxFF

Valence Angle	θ_o (degrees)	k_a (kcal/mol)	k_b (radians ²)	$p_{v,1}$	$p_{v,2}$
Au-Au-Au	12.2362	6.788	0.1388	0	0.9168

2.2.1 Comparing Force Fields with and without 3-Body Terms

We parameterized force fields with and without 3-body terms using identical training sets to test the effect of incorporating 3-body terms in Au force fields. Our preliminary results in 1 show that the curvature, optimal volume, and cohesive energy of the Au fcc, sc, and diamond EOS all significantly improve by adding 3-body terms, which appear to shift the onset of the convex regions to larger volumes.

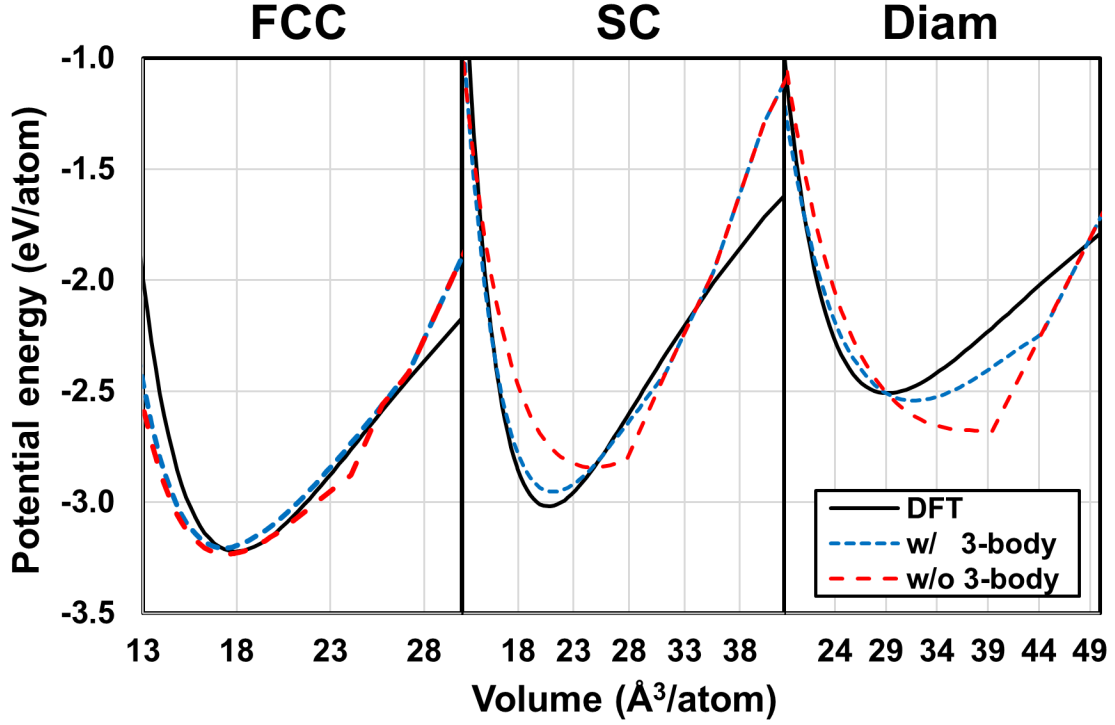


Figure 1: Comparing Au EOS data for our best case force fields with and without 3-body terms

We note that this treatment also brings significantly higher costs to the calculations [2](#). While the increase in computational cost due to 3-body terms is minimal in small systems, large systems prove to be much more expensive.

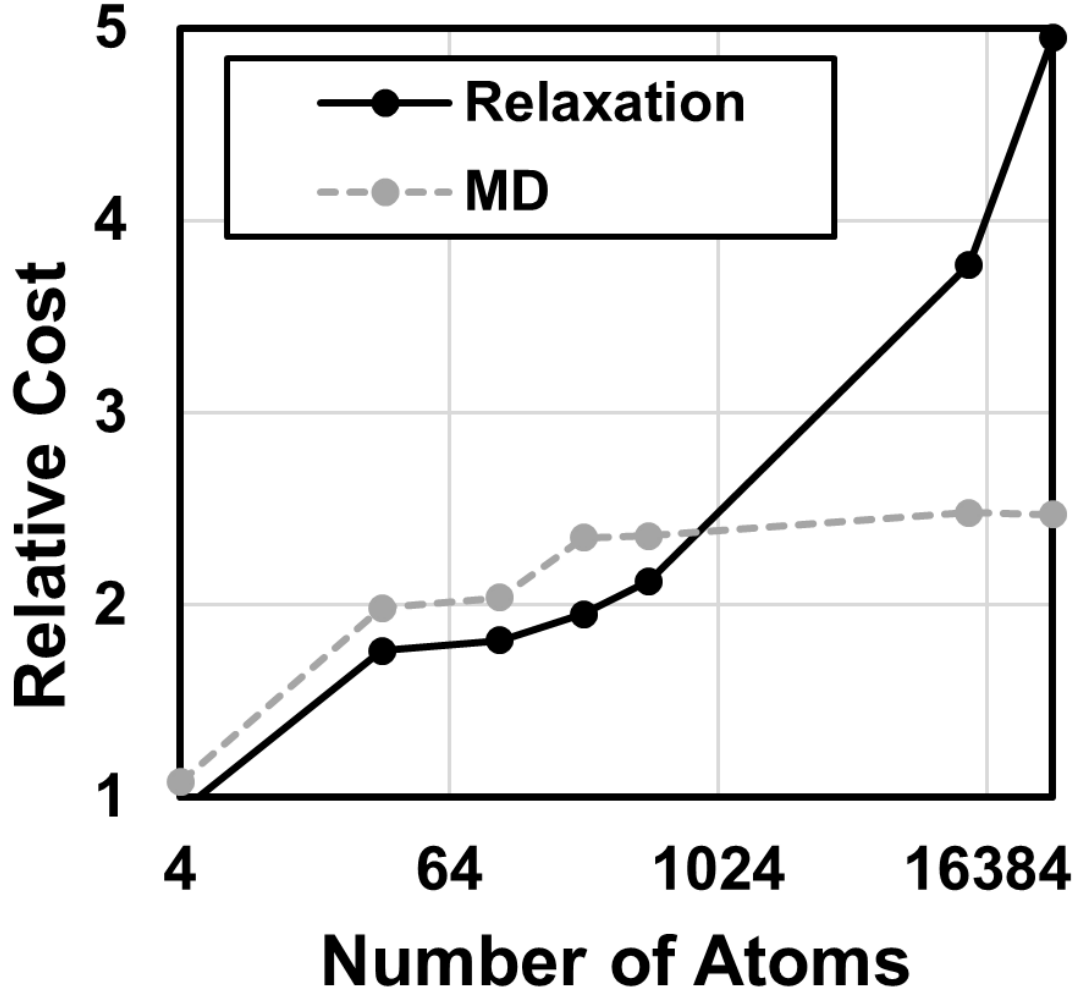


Figure 2: Comparing the relative cost increase from including 3-body terms for Molecular dynamics simulations and geometry relaxations (using a conjugate gradient method) with system size increase.

2.2.2 Monte-Carlo Force-Field Optimization Process (MCFFopt)

MCFFopt is a stochastic force field optimization process that relies on minimizing an objective function (Equation 1).

$$TotalError = \sum_{i=1}^n \left[\frac{E_{FField} - E_{QM}}{Weight} \right] \quad (1)$$

The optimizer program does this by randomly changing force field parameters within a range defined by the user and recalculating the objective function. Any parameter change that decreases the total error is instantly accepted, while moves that increase the total error have a probability of being accepted determined by Equation 2.

$$Probability = \min[1, \exp(-\beta \Delta Error)] \quad (2)$$

Where β is a parameter set by the user that is increased every step of the optimization process. $\beta_{initial}$ is usually small, resulting in an initial “annealing phase” where many parameter changes that increase the total error are allowed. As the β parameter increases, the likelihood of these events occurring decreases, and only parameter changes that decrease the total error occur. The annealing phase allows the process to sample more parameter space and potentially locate multiple distinct, viable parameter sets.

The MCFFopt tool does not have a built in force field parameter convergence criteria. However, over the course of an optimization calculation, the degree that parameters can change decreases. This means that after a certain number of steps parameters will begin to change by insignificant amounts, and the total error will stagnate. Thus, after a single run with the MCFFopt tool parameters will appear converged, but we can achieve still lower total errors, as shown in 2, by simply restarting the MCFFopt process until differences in total errors between runs becomes small.

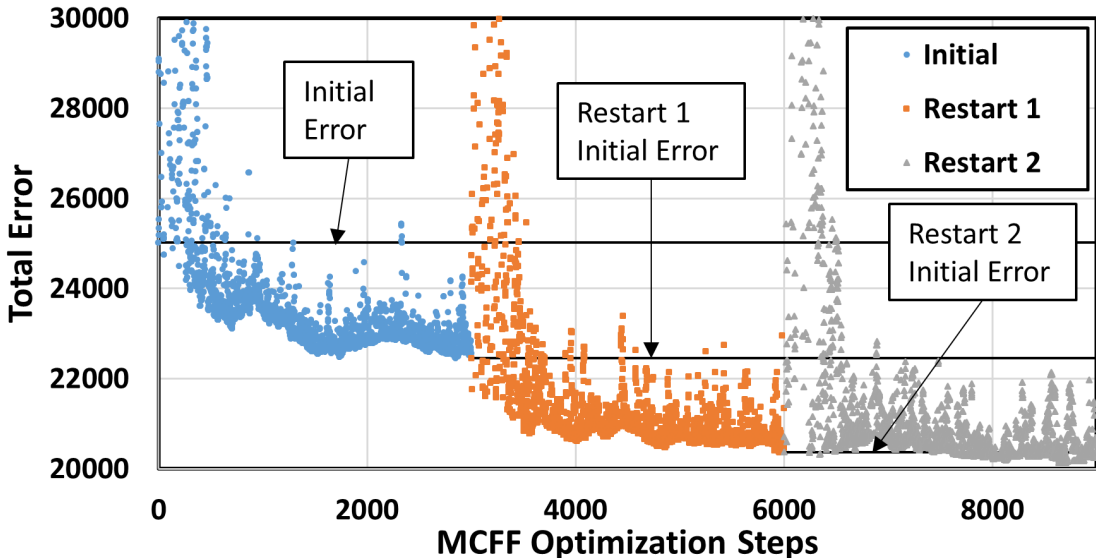


Figure 3: The total error over the course of an MCFF optimization procedure. Three MCFF cycles are shown, each restarted at the last step of the previous optimization.

Restarting the MCFF optimization procedure resets these parameters, and the process will go through another annealing phase, and then the total error will continue to decrease. We stopped optimizations once the final error was less than one percent of the initial error.

The training process requires user-defined parameters to control the MCFF optimization (contained in table 2), and a range of maximum and minimum values for each optimizable ReaxFF parameter. More information on these specific files is available in the ADF documentation.

Table 2: MCFF optimization parameters used in ADF

Parameter	Value
$(\beta_{initial})$ (mcbeta)	0.0100
Increase β by x per MCFFopt step (mcbsca)	1.0001
Fraction of variables active per step (mcacof)	0.2
Target acceptance rate (mctart)	0.30
Max. acceptance rate (mcmart)	0.70
Divide parameter range into y steps (mcrxxd)	100

ReaxFF training requires files that contain the geometries and energies that the force field will be trained to, weights of the relative importance of those geometries and energies, and an initial set of force field parameters taken from Ref. 1. That ReaxFF did not contain 3-body interaction parameters, so we used an average of 3-body interaction terms available for other atom types as our initial guess.

We initially used a weighting scheme nearly equivalent to that used in Ref. 1. We increased and/or decreased the weights of different geometry types (bulk vs. cluster vs. surface) until the force field adequately reproduced all of the data in the training set. Briefly, EOS structures were weighted on scale sliding from 0.2 to 10. Geometries near the minimum energy bulk structures were weighted highly (0.2), while structures further away from the equilibrium geometries were weighted lower (10.0). All surface and cluster calculations had weights of 1.0. In principle, changing these weights can result in force fields that are better suited for different geometry types.

2.2.3 Adding ReaxFF energies to the database

ReaxFF is a module included in the LAMMPS simulation suit (<http://lammmps.sandia.gov>), and must be run inside of this framework. However, a wrapper has been previously developed which allows LAMMPS calculations to be started and managed from a python interface. A python function was developed to rapidly calculate the energy of an ASE atoms object. The code for this function is demonstrated below.

```

1  # Personal function for interaction with LAMMPS in python
2  # Can be found at: https://github.com/jboes/jbtools
3  import jbtools.utils as jb
4  from ase.db import connect
5
6  db = connect('data.db')
7
8  # Select all entries
9  for d in db.select():
10     atoms = db.get_atoms(d.id)
11
12     Rnrg = jb.reax_potential_energy(atoms)
13
14     db.update(d.id, reax_energy=Rnrg)

```

With this function, the energies of all of the structures in the database could then be rapidly calculated and incorporated into the database for ease of access in the analysis portion of this work. This way, ReaxFF predicted energies for all structures included in the database can be accessed through the keyword “reax_energy”.

2.2.4 Manuscript figure fig-reax-train

```
1 import matplotlib.pyplot as plt
2 import matplotlib.mlab as mlab
3 from ase.db import connect
4 import numpy as np
5 from scipy.stats import norm
6 from matplotlib import gridspec
7 import matplotlib.patches as mpatches
8
9 db = connect('data.db')
10
11 S, Re, Qe = [], [], []
12 for d in db.select(['train_set=True']):
13     S += [d.structure]
14     Qe += [d.energy / d.natoms]
15     Re += [d.reax_energy / d.natoms]
16
17 S = np.array(S)
18 Qe = np.array(Qe)
19 Re = np.array(Re)
20
21 cmap, hdl = {}, []
22 for s, c in zip(set(S), ['b', 'r', 'g']):
23     cmap[s] = c
24     hdl += [mpatches.Patch(color=c, label=s)]
25
26 c = []
27 for s in S:
28     c += [cmap[s]]
29
30 RMSE = np.sqrt(sum((Re - Qe) ** 2) / len(Re - Qe))
31
32 (mu, sigma) = norm.fit(Re - Qe)
33
34 fig = plt.figure(figsize=(6, 4))
35 gs = gridspec.GridSpec(1, 2, width_ratios=[2, 1])
36 ax0 = plt.subplot(gs[0])
37 ax0.plot([min(Qe), 0], [0, 0], 'k--', lw=2)
38 ax0.scatter(Qe, Re - Qe, marker='o', color=c)
39 ax0.text(min(Qe) + 0.15, 1.08,
40         'RMSE: {0:1.3f}'.format(RMSE),
41         fontsize=12, va='top', ha='left')
42 ax0.set_xlim(min(Qe), 0)
43 ax0.set_ylim(-1.2, 1.2)
44 ax0.set_xlabel('DFT potential energy (eV/atom)')
45 ax0.set_ylabel('Residual error (eV/atom)')
46 ax0.legend(loc=3, handles=hdl, fontsize=12, frameon=False)
47
48 ax1 = plt.subplot(gs[1])
49
50 n, bins, patches = ax1.hist(Re - Qe, 50,
51                             range=(-1.2, 1.2),
52                             weights=np.ones_like(Re - Qe)/len(Re),
53                             facecolor='k',
54                             alpha=0.5,
55                             orientation='horizontal')
56
57 y = mlab.normpdf(bins, mu, sigma)
58 ax1.text(0.05, 1.12, '$\mu$: {0:1.3f}'.format(mu), fontsize=12,
59         va='top', ha='left')
60 ax1.text(0.05, 0.96, '$\sigma$: {0:1.3f}'.format(sigma), fontsize=12,
61         va='top', ha='left')
62 ax1.plot(y / sum(y), bins, 'k--', lw=2)
63 ax1.plot([0, 10], [0, 0], 'k--', lw=2)
64 ax1.set_xlabel('Probability')
65 ax1.set_ylim(-1.2, 1.2)
```

```

66 ax1.set_xlim(0, 0.6)
67 ax1.set_yticklabels([])
68 ax1.set_xticks(ax1.get_xticks()[1::2])
69 plt.tight_layout(w_pad=-0.5)
70 for ext in ['png', 'eps']:
71     plt.savefig('./images/fig-reax-train.{0}'.format(ext), dpi=300)

```

2.2.5 Manuscript figure fig-reax-vaild

```

1  import matplotlib.pyplot as plt
2  from ase.db import connect
3  import numpy as np
4  from matplotlib import gridspec
5  import matplotlib.patches as mpatches
6
7  db = connect('data.db')
8
9  S, Re, Qe = [], [], []
10 for d in db.select(['train_set=False']):
11     S += [d.structure]
12     Qe += [d.energy / d.natoms]
13     Re += [d.reax_energy / d.natoms]
14
15 S = np.array(S)
16 Qe = np.array(Qe)
17 Re = np.array(Re)
18
19 cmap, hdl = {}, []
20 for s, c in zip(set(S), ['b', 'r', 'g']):
21     cmap[s] = c
22     hdl += [mpatches.Patch(color=c, label=s)]
23
24 c = []
25 for s in S:
26     c += [cmap[s]]
27
28 RMSE = np.sqrt(sum((Re - Qe) ** 2) / len(Re - Qe))
29
30 fig = plt.figure(figsize=(6, 4))
31 gs = gridspec.GridSpec(1, 2, width_ratios=[2, 1])
32 ax0 = plt.subplot(gs[0])
33 ax0.plot([min(Qe), 0], [0, 0], 'k--', lw=2)
34 ax0.scatter(Qe, Re - Qe, marker='o', color=c)
35 ax0.text(min(Qe) + 0.15, 0.7,
36         'RMSE: {0:1.3f}'.format(RMSE),
37         fontsize=12, va='top', ha='left')
38 ax0.set_xlim(min(Qe), 0)
39 ax0.set_ylim(-0.8, 0.8)
40 ax0.set_xlabel('DFT potential energy (eV/atom)')
41 ax0.set_ylabel('Residual error (eV/atom)')
42 ax0.legend(loc=1, handles=hdl, fontsize=12, frameon=False)
43
44 ax1 = plt.subplot(gs[1])
45
46 n, bins, patches = ax1.hist(Re - Qe, 50,
47                             range=(-1.2, 1.2),
48                             weights=np.ones_like(Re - Qe)/len(Re),
49                             facecolor='k',
50                             alpha=0.5,
51                             orientation='horizontal')
52
53 ax1.plot([0, 10], [0, 0], 'k--', lw=2)
54 ax1.set_xlabel('Probability')
55 ax1.set_ylim(-1.2, 1.2)
56 ax1.set_xlim(0, 0.6)

```

```

57 ax1.set_yticklabels([])
58 ax1.set_xticks(ax1.get_xticks()[1::2])
59 plt.tight_layout(w_pad=-0.5)
60 for ext in ['png', 'eps']:
61     plt.savefig('./images/fig-reax-valid.{0}'.format(ext), dpi=300)

```

2.3 Neural Network

BPNNs were produced using the Neural code developed by the Peterson group at Brown University (<https://bitbucket.org/andrewpeterson/neural>). This code is no longer supported since the time the work was completed. However, the Peterson group is currently developing a sister code called AMP which is capable of all the same functionality as Neural (<https://bitbucket.org/andrewpeterson/amp>). The parameter files used by AMP are not backwards compatible with Neural, so the parameter files included here have been manually updated to function with *AMP*.

The parameters file needed to run the BPNN produced in this work is attached here:  (double-click to open).

2.3.1 Adding BPNN energies to the database

The code developed by the Peterson group is already integrated into ASE, making calculation of energy using the BPNN developed in this work trivial. The code used to calculate these energies and add them to the database is included below.

```

1  from ase.db import connect
2  from amp import Amp
3
4  db = connect('data.db')
5
6  # Establish the BPNN as the calculator for our energies
7  calc = Amp(load='neural-parameters.json')
8
9  # Select all entries
10 for d in db.select():
11     atoms = db.get_atoms(d.id)
12
13     atoms.set_calculator(calc)
14     Nnrg = atoms.get_potential_energy()
15
16     db.update(d.id, neural_energy=Nnrg)

```

This allows the BPNN predicted energy of a structure to be accessed easily by first defining the structure of interest and then utilizing the keyword “neural_energy”.

2.3.2 Manuscript figure fig-neural-train

```

1  import matplotlib.pyplot as plt
2  import matplotlib.mlab as mlab
3  from ase.db import connect
4  import numpy as np
5  from scipy.stats import norm
6  from matplotlib import gridspec
7  import matplotlib.patches as mpatches

```

```

8
9 db = connect('data.db')
10
11 S, Qe, Ne = [], [], []
12 for d in db.select(['train_set=True']):
13     S += [d.structure]
14     Qe += [d.energy / d.natoms]
15     Ne += [d.neural_energy / d.natoms]
16
17 S = np.array(S)
18 Qe = np.array(Qe)
19 Ne = np.array(Ne)
20
21 cmap, hdl = {}, []
22 for s, c in zip(set(S), ['b', 'r', 'g']):
23     cmap[s] = c
24     hdl += [mpatches.Patch(color=c, label=s)]
25
26 c = []
27 for s in S:
28     c += [cmap[s]]
29
30 RMSE = np.sqrt(sum((Ne - Qe) ** 2) / len(Ne - Qe))
31
32 (mu, sigma) = norm.fit(Ne - Qe)
33
34 fig = plt.figure(figsize=(6, 4))
35 gs = gridspec.GridSpec(1, 2, width_ratios=[2, 1])
36 ax0 = plt.subplot(gs[0])
37 ax0.plot([min(Qe), 0], [0, 0], 'k--', lw=2)
38 ax0.scatter(Qe, Ne - Qe, marker='o', color=c)
39 ax0.text(min(Qe) + 0.1, 0.14,
40         'RMSE: {0:1.3f}'.format(RMSE),
41         fontsize=12, va='top', ha='left')
42 ax0.set_xlim(min(Qe), 0)
43 ax0.set_ylim(-0.15, 0.15)
44 ax0.set_xlabel('DFT potential energy (eV/atom)')
45 ax0.set_ylabel('Residual error (eV/atom)')
46 ax0.legend(loc='best', handles=hdl, fontsize=12, frameon=False)
47
48 ax1 = plt.subplot(gs[1])
49
50 n, bins, patches = ax1.hist(Ne - Qe, 50,
51                             range=(-0.15, 0.15),
52                             weights=np.ones_like(Ne - Qe)/len(Ne),
53                             facecolor='k',
54                             alpha=0.5,
55                             orientation='horizontal')
56
57 y = mlab.normpdf(bins, mu, sigma)
58 ax1.text(0.05, 0.142, '$\mu$: {0:1.3f}'.format(mu), fontsize=12, va='top', ha='left')
59 ax1.text(0.05, 0.122, '$\sigma$: {0:1.3f}'.format(sigma), fontsize=12, va='top', ha='left')
60 ax1.plot(y / sum(y), bins, 'k--', lw=2)
61 ax1.plot([0, 50], [0, 0], 'k--', lw=2)
62 ax1.set_xlabel('Probability')
63 ax1.set_ylim(-0.15, 0.15)
64 ax1.set_xlim(0, 0.6)
65 ax1.set_yticklabels([])
66 ax1.set_xticks(ax1.get_xticks()[1::2])
67 plt.tight_layout(w_pad=-0.5)
68 for ext in ['png', 'eps']:
69     plt.savefig('./images/fig-neural-train.{0}'.format(ext), dpi=300)

```

2.3.3 Manuscript figure fig-neural-valid

```
1 import matplotlib.pyplot as plt
2 from ase.db import connect
3 import numpy as np
4 from matplotlib import gridspec
5 import matplotlib.patches as mpatches
6
7 db = connect('data.db')
8
9 S, Qe, Ne = [], [], []
10 for d in db.select(['train_set=False']):
11     S += [d.structure]
12     Qe += [d.energy / d.natoms]
13     Ne += [d.neural_energy / d.natoms]
14
15 S = np.array(S)
16 Qe = np.array(Qe)
17 Ne = np.array(Ne)
18
19 cmap, hdl = {}, []
20 for s, c in zip(set(S), ['b', 'r', 'g']):
21     cmap[s] = c
22     hdl += [mpatches.Patch(color=c, label=s)]
23
24 c = []
25 for s in S:
26     c += [cmap[s]]
27
28 RMSE = np.sqrt(sum((Ne - Qe) ** 2) / len(Ne - Qe))
29
30 fig = plt.figure(figsize=(6, 4))
31 gs = gridspec.GridSpec(1, 2, width_ratios=[2, 1])
32 ax0 = plt.subplot(gs[0])
33 ax0.plot([min(Qe), 0], [0, 0], 'k--', lw=2)
34 ax0.scatter(Qe, Ne - Qe, marker='o', color=c)
35 ax0.text(min(Qe) + 0.1, 0.14,
36         'RMSE: {0:1.3f}'.format(RMSE),
37         fontsize=12, va='top', ha='left')
38 ax0.set_xlim(min(Qe), 0)
39 ax0.set_ylim(-0.15, 0.15)
40 ax0.set_xlabel('DFT potential energy (eV/atom)')
41 ax0.set_ylabel('Residual error (eV/atom)')
42 ax0.legend(loc='best', handles=hdl, fontsize=12, frameon=False)
43
44 ax1 = plt.subplot(gs[1])
45
46 n, bins, patches = ax1.hist(Ne - Qe, 50,
47                             range=(-0.15, 0.15),
48                             weights=np.ones_like(Ne - Qe)/len(Ne),
49                             facecolor='k',
50                             alpha=0.5,
51                             orientation='horizontal')
52
53 ax1.plot([0, 50], [0, 0], 'k--', lw=2)
54 ax1.set_xlabel('Probability')
55 ax1.set_ylim(-0.15, 0.15)
56 ax1.set_xlim(0, 0.6)
57 ax1.set_yticklabels([])
58 ax1.set_xticks(ax1.get_xticks()[1::2])
59 plt.tight_layout(w_pad=-0.5)
60 for ext in ['png', 'eps']:
61     plt.savefig('./images/fig-neural-valid.{0}'.format(ext), dpi=300)
```

3 Results

3.1 Bulk properties

3.1.1 Manuscript figure fig-bulk-eos

```
1 import matplotlib.pyplot as plt
2 from ase.utils.eos import EquationOfState
3 from ase.db import connect
4 import numpy as np
5 from ase.units import kJ
6
7 db = connect('data.db')
8
9 print('#+caption: Comparison of EOS metrics for DFT, ReaxFF, and NPNN fits as shown in Figure ref:fig-bulk-eos.')
10 print('#+attr_latex: :placement [H]')
11 print('#+tblname: tbl-eos')
12 print('|Structure|Minimum volume (\AA^3)|Minimum energy (eV)|Bulk Mod. (GPa)|')
13 print('|-|')
14
15 f, ax = plt.subplots(1, 3, figsize=(6, 5))
16
17 tag = ['Face Centered\nCubic', 'Simple Cubic', 'Diamond']
18
19 for i, key in enumerate(['fcc', 'sc', 'diam']):
20
21     V, Qe, Re, Ne = [], [], [], []
22     for d in db.select(['bulk={0}'.format(key), 'factor']):
23         V += [d.volume / d.natoms]
24         Qe += [d.energy / d.natoms]
25         Ne += [d.neural_energy / d.natoms]
26         Re += [d.reax_energy / d.natoms]
27
28     sel = V[Qe.index(min(Qe))]
29     ind = (np.array(V) > sel - 15) & (np.array(V) < sel + 15)
30     x = np.linspace(min(V), max(V), 250)
31     V = np.array(V)[ind]
32
33     for nrg, name, col in zip([Qe, Ne, Re],
34                             ['DFT', 'BPNN', 'ReaxFF'],
35                             ['k-', 'r--', 'b:']):
36
37         nrg = np.array(nrg)[ind]
38         eos = EquationOfState(V, nrg)
39         v0, e0, B = eos.fit()
40         fit = np.poly1d(np.polyfit(V**-(1.0 / 3), nrg, 3))
41
42         ax[i].plot(x, fit(x**-(1.0 / 3)), col, lw=2, label='{0}'.format(name))
43         ax[i].set_xlim(min(V), max(V))
44         ax[i].set_ylim(-3.5, -1.0)
45         ax[i].set_title('{0}'.format(tag[i]))
46         if i > 0:
47             ax[i].set_yticklabels([])
48         print('|{0}-{1}|{2:1.2f}|{3:1.2f}|{4:1.0f}'.format(name, key, v0, e0,
49                                                         B / kJ * 1.0e24))
50     print('|-|')
51
52 ax[0].set_xticks([14, 19, 24, 29])
53 ax[1].set_xticks([17, 22, 27, 32])
54 ax[2].set_xticks([21, 26, 31, 36, 41])
55 ax[0].set_ylabel('Potential energy (eV/atom)')
56 ax[1].set_xlabel('Volume ($\AA^3$/atom)')
57 ax[2].legend(loc='best', fontsize=12)
58 plt.tight_layout(w_pad=-0.3)
59 for ext in ['png', 'eps', 'pdf']:
60     plt.savefig('./images/fig-bulk-eos.{0}'.format(ext), dpi=300)
```

Table 3: Comparison of EOS metrics for DFT, ReaxFF, and NPNN fits as shown in Figure fig-bulk-eos.

Structure	Minimum volume (\AA^3)	Minimum energy (eV)	Bulk Mod. (GPa)
DFT-fcc	17.97	-3.23	147
BPNN-fcc	17.99	-3.23	145
ReaxFF-fcc	17.60	-3.22	122
DFT-sc	20.73	-3.02	110
BPNN-sc	20.66	-3.02	110
ReaxFF-sc	21.29	-2.96	84
DFT-diam	29.04	-2.51	56
BPNN-diam	28.98	-2.51	57
ReaxFF-diam	31.92	-2.54	37

3.1.2 BCC and HCP equations of state

To conserve space in the manuscript, the EOS for hcp and bcc structures are shown in Figure 4. The corresponding data resulting from the fits to the EOS can be found in Table 4

```

1  import matplotlib.pyplot as plt
2  from ase.utils.eos import EquationOfState
3  from ase.db import connect
4  import numpy as np
5  from ase.units import kJ
6
7  db = connect('data.db')
8
9  print('#+caption: Comparison of EOS metrics for DFT, ReaxFF, and NPNN fits as shown in Figure ref:si-bulk-eos.')
10 print('#+attr_latex: :placement [H]')
11 print('#+tblname: tbl-eos2')
12 print('|Structure|Minimum volume (\AA^3)|Minimum energy (eV)|Bulk Mod. (GPa)|')
13 print('|-|')
14
15 f, ax = plt.subplots(1, 2, figsize=(6, 5))
16
17 tag = ['Body Centered\nCubic', 'Hexagonal Close\nPacking']
18
19 for i, key in enumerate(['bcc', 'hcp']):
20
21     V, Qe, Re, Ne = [], [], [], []
22     for d in db.select(['bulk={0}'.format(key), 'factor']):
23         V += [d.volume / d.natoms]
24         Qe += [d.energy / d.natoms]
25         Ne += [d.neural_energy / d.natoms]
26         Re += [d.reax_energy / d.natoms]
27
28     sel = V[Qe.index(min(Qe))]
29     ind = (np.array(V) > sel - 15) & (np.array(V) < sel + 15)
30     x = np.linspace(min(V), max(V), 250)
31     V = np.array(V)[ind]
32
33     for nrg, name, col in zip([Qe, Ne, Re],
34                             ['DFT', 'BPNN', 'ReaxFF'],
35                             ['k-', 'r--', 'b:']):

```

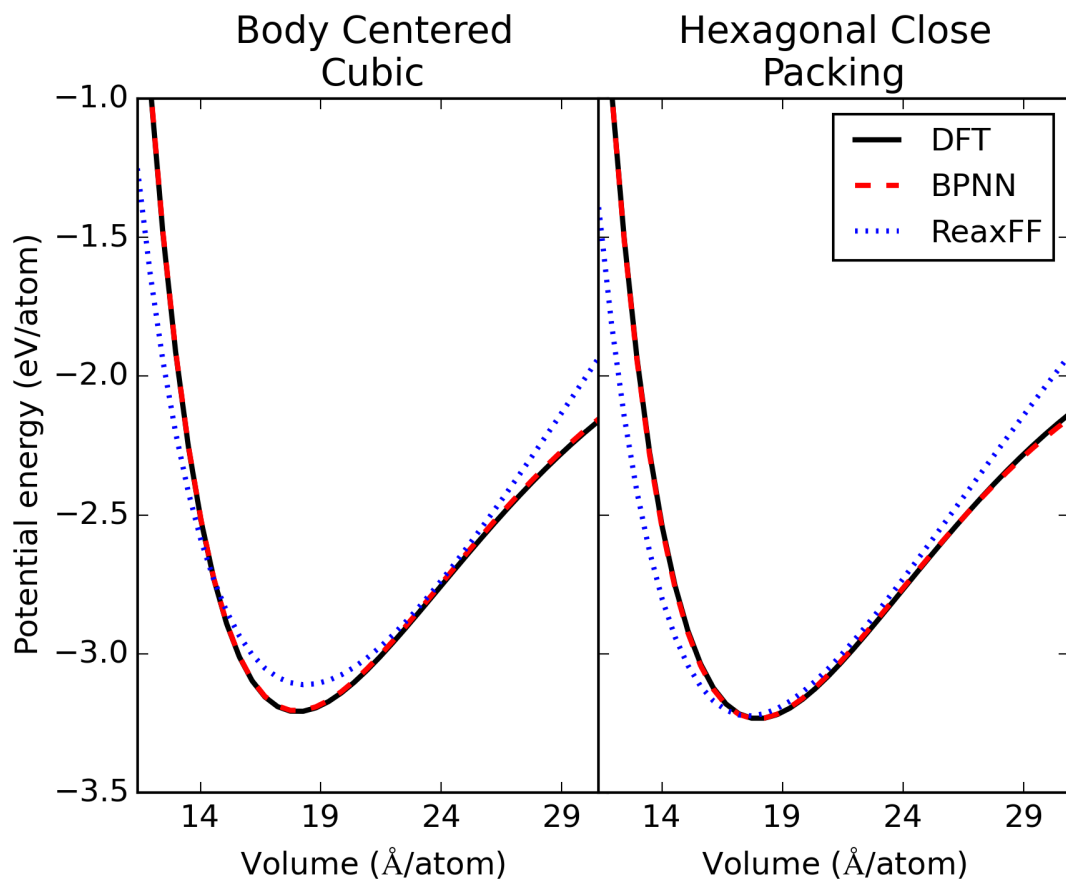


Figure 4: Comparison of EOS fits to DFT, ReaxFF, and NPNN training and validation set data for the bcc and hcp structures.

```

36     nrg = np.array(nrg)[ind]
37     eos = EquationOfState(V, nrg)
38     v0, e0, B = eos.fit()
39     fit = np.poly1d(np.polyfit(V**-(1.0 / 3), nrg, 3))
40
41
42     ax[i].plot(x, fit(x**-(1.0 / 3)), col, lw=2, label='{0}'.format(name))
43     ax[i].set_xlim(min(V), max(V))
44     ax[i].set_ylim(-3.5, -1.0)
45     ax[i].set_title('{0}'.format(tag[i]))
46     if i > 0:
47         ax[i].set_yticklabels([])
48     print('{0}-{1}|{2:1.2f}|{3:1.2f}|{4:1.0f}'.format(name, key, v0, e0,
49                                                     B / kJ * 1.0e24))
50     print('|-')
51
52 ax[0].set_xticks([14, 19, 24, 29])
53 ax[1].set_xticks([14, 19, 24, 29])
54 ax[0].set_ylabel('Potential energy (eV/atom)')
55 ax[0].set_xlabel('Volume ($\AA^3$/atom)')
56 ax[1].set_xlabel('Volume ($\AA^3$/atom)')
57 ax[1].legend(loc='best', fontsize=12)
58 plt.tight_layout(w_pad=-0.3)
59 for ext in ['png', 'eps', 'pdf']:
60     plt.savefig('./images/si-bulk-eos.{0}'.format(ext), dpi=300)

```

Table 4: Comparison of EOS metrics for DFT, ReaxFF, and NPNN fits as shown in Figure 4.

Structure	Minimum volume (\AA^3)	Minimum energy (eV)	Bulk Mod. (GPa)
DFT-bcc	18.02	-3.21	145
BPNN-bcc	18.00	-3.21	146
ReaxFF-bcc	18.36	-3.11	107
DFT-hcp	17.98	-3.23	147
BPNN-hcp	17.93	-3.23	148
ReaxFF-hcp	17.60	-3.22	122

3.1.3 Full equations of state

```

1  import matplotlib.pyplot as plt
2  from ase.utils.eos import EquationOfState
3  from ase.db import connect
4  import numpy as np
5  from ase.units import kJ
6
7  db = connect('data.db')
8
9  f, ax = plt.subplots(1, 5, figsize=(12, 5))
10
11  tag = ['Face Centered\nCubic', 'Body Centered\nCubic',
12        'Hexagonal Close\nPacking', 'Simple Cubic', 'Diamond',]
13
14  for i, key in enumerate(['fcc', 'bcc', 'hcp', 'sc', 'diam',]):
15
16      V, Qe, Re, Ne = [], [], [], []
17      for d in db.select(['bulk={0}'.format(key), 'factor']):
18          V += [d.volume / d.natoms]
19          Qe += [d.energy / d.natoms]
20          Ne += [d.neural_energy / d.natoms]

```

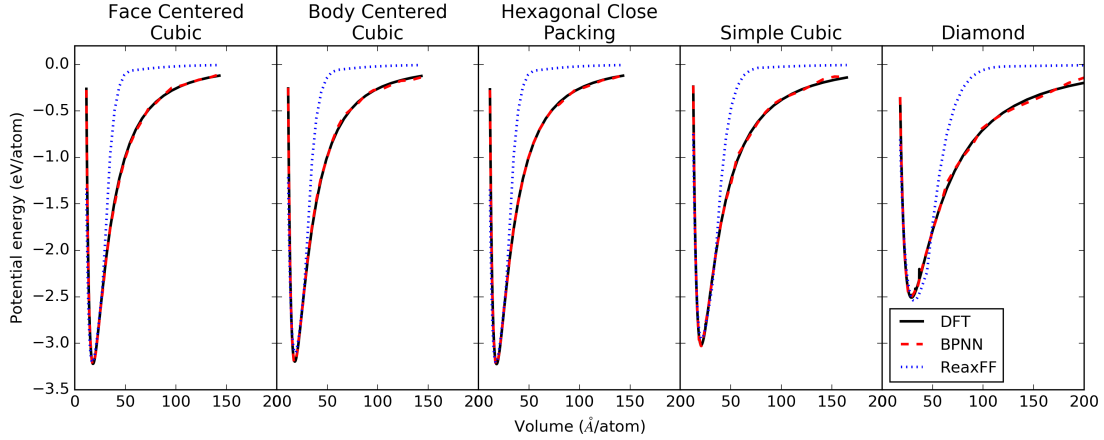



Figure 5: Comparison of full EOS fits to DFT, ReaxFF, and NPNN training and validation set data for all structures.

```

21     Re += [d.reax_energy / d.natoms]
22
23
24     srt = [j[0] for j in sorted(enumerate(V), key=lambda x:x[1])]
25     V = np.array(V)[srt]
26     Qe = np.array(Qe)[srt]
27     Ne = np.array(Ne)[srt]
28     Re = np.array(Re)[srt]
29
30     ax[i].plot(V, Qe, 'k-', lw=2, label='DFT')
31     ax[i].plot(V, Ne, 'r--', lw=2, label='BPNN')
32     ax[i].plot(V, Re, 'b:', lw=2, label='ReaxFF')
33     if i > 0:
34         ax[i].set_yticklabels([])
35     ax[i].set_ylim(-3.5, 0.2)
36     ax[i].set_xlim(0, 200)
37     ax[i].set_title('{0}'.format(tag[i]))
38
39 ax[0].set_ylabel('Potential energy (eV/atom)')
40 ax[2].set_xlabel('Volume ($\text{\AA}^3/\text{atom})$')
41 ax[4].legend(loc='best', fontsize=12)
42 plt.tight_layout(w_pad=-1.3)
43 for ext in ['png', 'eps', 'pdf']:
44     plt.savefig('./images/si-full-eos.{0}'.format(ext), dpi=300)

```

3.1.4 Manuscript figure fig-vacancy-formation

```

1  from ase.db import connect
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  db = connect('data.db')
6
7  # Energy of single atom fcc reference
8  ref = db.get(['bulk=fcc', 'traj=0', 'factor=1'])
9
10 C, Qe, Re, Ne = [], [], [], []
11 for d in db.select(['bulk=fcc', 'type=vacancy', 'traj=0', 'image=0']):
12     C += [d.concentration]
13     Qe += [d.energy - (d.natoms * ref.energy)]

```

```

14     Ne += [d.neural_energy - (d.natoms * ref.neural_energy)]
15     Re += [d.reax_energy - (d.natoms * ref.reax_energy)]
16
17 plt.figure(figsize=(6, 4))
18 plt.plot([0, 0.13], [0.42, 0.42], 'k--')
19 plt.text(0.065, 0.42, 'Literature DFT', va='bottom', fontsize=14)
20 plt.plot([0, 0.13], [0.93, 0.93], 'k--')
21 plt.text(0.065, 0.93, 'Experimental', va='bottom', fontsize=14)
22 plt.text(0.125, Qe[0]+0.03, 'DFT', color='k', fontsize=14, ha='right')
23 plt.scatter(C, Qe, marker='o', color='k')
24 plt.text(0.125, Ne[0]+0.03, 'BPNN', color='r', fontsize=14, ha='right')
25 plt.scatter(C, Ne, marker='s', color='r')
26 plt.text(0.125, Re[0]+0.03, 'ReaxFF', color='b', fontsize=14, ha='right')
27 plt.scatter(C, Re, marker='^', color='b')
28 plt.xlabel('Vacancy concentration (vacancies/atom)')
29 plt.ylabel('Vacancy formation energy (eV/vacancy)')
30 plt.xlim(0, 0.13)
31 plt.ylim(0, 1.0)
32 plt.tight_layout()
33 for ext in ['png', 'eps', 'pdf']:
34     plt.savefig('./images/fig-vacancy-formation.{0}'.format(ext), dpi=300)

```

3.1.5 Structural relaxation of ≈ 0.015 vacancies/atom

As mentioned in the manuscript, the unit cell used to calculate the vacancy concentration at ≈ 0.015 vacancies/atom reconfigured to a different, less favorable, local minima. This is demonstrated in Figure 6 which depicts the energies of the relaxation pathways for the DFT calculation. The reconfiguration is shown as the last structure in the trajectory along side the minimum energy structure.

```

1  from ase.db import connect
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from ase.io import write
5  from matplotlib.offsetbox import OffsetImage, AnnotationBbox
6  import matplotlib.image as mpimg
7  import os
8
9  db = connect('data.db')
10
11  trajectory = db.select(['bulk=fcc', 'type=vacancy', 'image=0',
12                        'concentration=0.015625'])
13
14  E, t = [], []
15  for traj in trajectory:
16      E.append(traj.energy)
17      t.append(traj.traj)
18
19  E = np.array(E) - min(E)
20
21  fig = plt.figure(figsize=(6, 4))
22  ax = fig.add_subplot(111)
23  for i, a in enumerate([3, 0]):
24      atoms = db.get_atoms(['bulk=fcc', 'type=vacancy',
25                          'traj={0}'.format(a), 'image=0',
26                          'concentration=0.015625'])
27
28      write('./images/temp.png', atoms, show_unit_cell=2)
29
30      image = mpimg.imread('./images/temp.png')
31      imagebox = OffsetImage(image)
32
33      ax.add_artist(AnnotationBbox(imagebox,

```

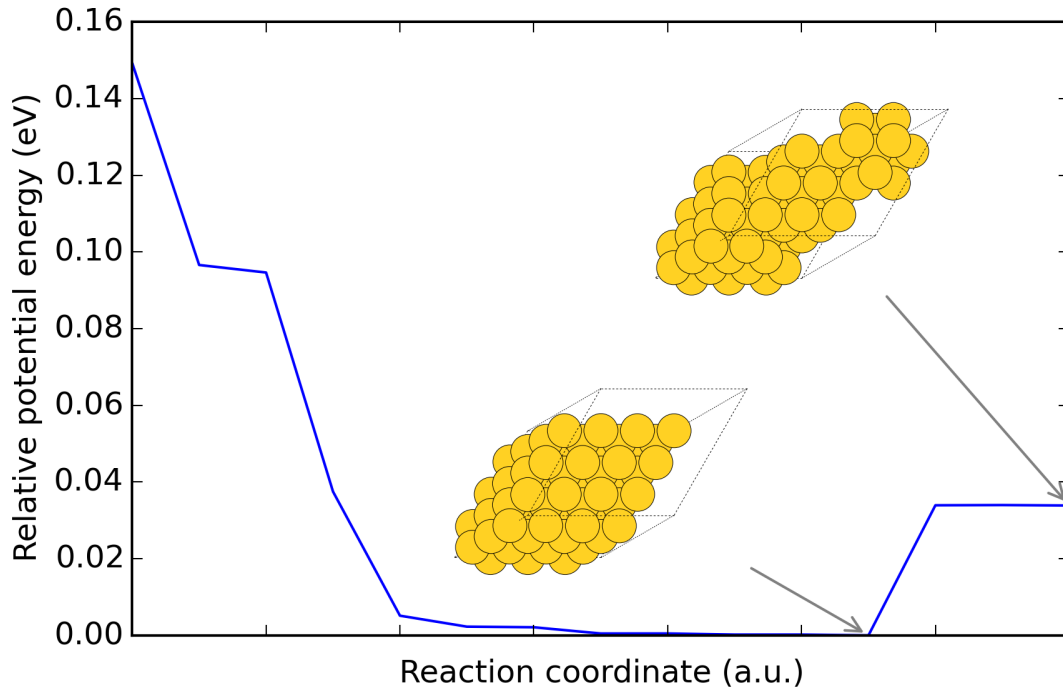


Figure 6: Relaxation pathway of the vacancy structure at the ≈ 0.015 vacancies/atom concentration.

```

34         xy=(a, E[t[a]]),
35         xybox=(a + 4, E[t[a]] + 0.04*(i + 1)),
36         pad=-0.2,
37         frameon=False,
38         arrowprops=dict(arrowstyle='->',
39                           color='0.5',
40                           zorder=5,
41                           connectionstyle='arc,angleA=-90,armA=0')
42     )
43 )
44     os.unlink('./images/temp.png')
45
46     ax.plot(t, E)
47     ax.invert_xaxis()
48     ax.set_xticklabels([])
49     plt.ylabel('Relative potential energy (eV)')
50     plt.xlabel('Reaction coordinate (a.u.)')
51     plt.tight_layout()
52     for ext in ['png', 'eps', 'pdf']:
53         plt.savefig('./images/si-vacancy-reconfig.{0}'.format(ext), dpi=300)

```

3.1.6 Manuscript figure fig-vacancy-diffusion

```

1  from ase.db import connect
2  from ase.visualize import view
3  import matplotlib.pyplot as plt
4  import numpy as np
5  from scipy.interpolate import interp1d
6  from scipy.optimize import fmin

```

```

7
8 db = connect('data.db')
9
10 ref = db.get(['bulk', 'NEB=True', 'image=0', 'type=vacancy', 'traj=0'])
11
12 I, Qe, Re, Ne = [], [], [], []
13 for d in db.select(['bulk', 'NEB=True', 'type=vacancy', 'traj=0']):
14     I += [d.image]
15     Qe += [d.energy - ref.energy]
16     Ne += [d.neural_energy - ref.neural_energy]
17     Re += [d.reax_energy - ref.reax_energy]
18
19 sort = [i[0] for i in sorted(enumerate(I), key=lambda x:x[1])]
20
21 I = np.array([I[i] for i in sort]) + 1
22 x = np.linspace(I.min(), I.max())
23
24 fig = plt.figure(figsize=(6, 4))
25 ax = fig.add_subplot(1,1,1)
26 for nrg, name, s in zip([Qe, Ne, Re],
27                         ['DFT', 'BPNN', 'ReaxFF'],
28                         ['ko-', 'rs--', 'b^:']):
29
30     nrg = np.array([nrg[i] for i in sort])
31
32     f = interp1d(I, -nrg, kind='cubic', bounds_error=False)
33     xmax = fmin(f, 3.0, disp=False)
34
35     ax.plot(I, nrg, s[:2], label=name)
36     ax.plot(x, -f(x), color=s[0], ls=s[2:])
37     ax.annotate('{0} $E^{\ddag}$ = {1:1.2f}'.format(name, float(-f(xmax))),
38               xy=(3.0, -f(xmax)), xytext=(3.7, -f(xmax)),
39               ha='left', va='center', color=s[0],
40               arrowprops=dict(arrowstyle="->",
41                               shrinkB=10,
42                               color=s[0]))
43
44 ax.set_xticklabels([])
45 plt.xlabel('Reaction coordinate (a.u.)')
46 plt.ylabel('Potential energy (eV)')
47 plt.xlim(1, 5)
48 plt.ylim(0, 0.62)
49 plt.tight_layout()
50 for ext in ['png', 'eps', 'pdf']:
51     plt.savefig('./images/fig-vacancy-diffusion.{0}'.format(ext), dpi=300)

```

3.2 Surface calculations

3.2.1 Manuscript figure fig-full-diffusion

```

1 from ase.db import connect
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 db = connect('data.db')
6
7 f, ax = plt.subplots(2, 2, figsize=(6, 5))
8
9 for i, k in enumerate(['single1', 'terrace', 12],
10                      ['dimer1', 'dimer', 6]):
11
12     Qe, Re, Ne = [], [], []
13     for j, d in enumerate(db.select(['config={0}'.format(k[0]),
14                                     'group=timo'])):
15         Qe += [d.energy]

```

```

16         Ne += [d.neural_energy]
17         Re += [d.reax_energy]
18
19     Qe = np.array(Qe[:k[2]])
20     Ne = np.array(Ne[:k[2]])
21     Re = np.array(Re[:k[2]])
22
23     if i == 0:
24         Qe = np.hstack([Qe[:-1], Qe])
25         Ne = np.hstack([Ne[:-1], Ne])
26         Re = np.hstack([Re[:-1], Re])
27
28     m = list(Qe).index(Qe.min())
29     n = range(len(Qe))
30
31     Nerr = (Ne - Ne[m]) - (Qe - Qe[m])
32     Rerr = (Re - Re[m]) - (Qe - Qe[m])
33
34     # Plotting energy points
35     ax[0, i].plot(n, Ne - Ne[m], 'rs', label='BPNN')
36     ax[0, i].plot(n, Re - Re[m], 'b^', label='ReaxFF')
37     ax[0, i].plot(n, Qe - Qe[m], 'ko', fillstyle='none', label='DFT')
38
39     # Plotting the residuals
40     ax[1, i].plot([min(n), max(n)], [0, 0], 'k:')
41     ax[1, i].plot(n, Nerr, 'r.')
42     ax[1, i].plot(n, Rerr, 'b.')
43
44     ax[0, i].set_xlim(min(n), max(n))
45     ax[1, i].set_xlim(min(n), max(n))
46     ax[1, i].set_ylim(-0.3, 0.3)
47     ax[0, i].set_ylim(0, 1.0)
48     ax[0, i].set_title('{0} diffusion'.format(k[1]))
49     ax[1, i].set_xlabel('Reaction pathway (a.u.)')
50
51     ax[0, i].set_xticklabels([])
52     ax[1, i].set_xticklabels([])
53     if i != 0:
54         ax[0, i].set_yticklabels([])
55         ax[1, i].set_yticklabels([])
56
57 ax[1, 0].set_yticks(ax[1, 0].get_yticks()[:-2])
58 ax[1, 0].set_ylabel('Residual error (eV)')
59 ax[0, 0].set_ylabel('Total energy (eV)')
60 ax[0, 0].legend(loc='best', numpoints=1, fontsize=12, frameon=False)
61 plt.tight_layout(w_pad=0.2, h_pad=-0.3)
62 for ext in ['png', 'eps', 'pdf']:
63     plt.savefig('./images/fig-full-diffusion-{0}'.format(ext), dpi=300)

```

3.2.2 Manuscript figure fig-barrier-residuals

```

1  from ase.db import connect
2  import xlrd
3  import matplotlib.pyplot as plt
4  from matplotlib.path import Path
5  import matplotlib.patches as patches
6  import numpy as np
7
8  db = connect('data.db')
9
10 C = {}
11 for d in db.select(['group=timo', 'config!=sl']):
12     if d.config not in C.keys():
13         C[d.config] = {}
14

```

```

15     if d.identity not in C[d.config].keys():
16         C[d.config][d.identity] = []
17
18     C[d.config][d.identity] += [[d.energy, d.neural_energy, d.reax_energy]]
19
20     rQe, rNe, rRe = [], [], []
21     for cfg, data0 in C.iteritems():
22         for lbl, data1 in data0.iteritems():
23
24             Qe = np.array(data1).T[0]
25             m = list(Qe).index(Qe.min())
26
27             Qe = Qe - Qe[m]
28             Ne = np.array(data1).T[1] - np.array(data1).T[1][m]
29             Re = np.array(data1).T[2] - np.array(data1).T[2][m]
30
31             Qe = np.delete(Qe, m)
32             Ne = np.delete(Ne, m)
33             Re = np.delete(Re, m)
34
35             rQe += list(Qe)
36             rNe += list(Ne - Qe)
37             rRe += list(Re - Qe)
38
39     pct = 0.1
40     verts = [(0., pct), (1.2, pct), (1.2, -pct), (0., -pct), (0., 0.)]
41     codes = [Path.MOVETO, Path.LINETO, Path.LINETO, Path.LINETO, Path.CLOSEPOLY]
42     path = Path(verts, codes)
43
44     Nc = path.contains_points(zip(rQe, rNe))
45     Ni = float(sum(Nc)) / len(Nc)
46
47     Rc = path.contains_points(zip(rQe, rRe))
48     Ri = float(sum(Rc)) / len(Rc)
49
50     fig = plt.figure(figsize=(6, 4))
51     ax = fig.add_subplot(111)
52     patch = patches.PathPatch(path, facecolor='y', edgecolor='y', alpha=0.3)
53     ax.add_patch(patch)
54     ax.plot([0, 1.2], [0, 0], 'k--', zorder=1)
55
56     ax.scatter(rQe[:8] + rQe[30:], rNe[:8] + rNe[30:], color='r', marker='s', zorder=10, s=30)
57     ax.scatter(rQe[:8] + rQe[30:], rRe[:8] + rRe[30:], color='b', marker='^', zorder=20, s=30)
58     ax.scatter(rQe[8:30], rNe[8:30], color='r', marker='s', facecolor='none', zorder=30, s=20)
59     ax.scatter(rQe[8:30], rRe[8:30], color='b', marker='^', facecolor='none', zorder=40, s=30)
60     ax.text(0.02, 0.38, 'Within $\pm$ 0.1 eV error:', va='top', ha='left')
61     ax.text(0.02, 0.32,
62            'ReaxFF: {0:1.1f}%'.format(Ri*100, pct*100),
63            va='top', ha='left', color='b', zorder=100)
64     ax.text(0.02, 0.26,
65            'BPNN: {0:1.1f}%'.format(Ni*100, pct*100),
66            va='top', ha='left', color='r', zorder=100)
67
68     plt.xlabel('DFT Potential Energy (eV)')
69     plt.ylabel('Residual Error (eV)')
70     plt.xlim(0, max(rQe))
71     plt.ylim(-0.4, 0.4)
72     plt.tight_layout()
73     for ext in ['png', 'eps']:
74         plt.savefig('./images/fig-barrier-residuals.{0}'.format(ext), dpi=300)

```

3.2.3 Figures of individual fcc(100) diffusion pathways

Not all of the fcc(100) surface diffusion pathways could be directly shown in the manuscript. Instead, we show them here to demonstrate how each of the potentials performs in each

case. The residuals of each method are also shown. These residuals are the same values incorporated in Figure fig-barrier-residuals included in the manuscript.

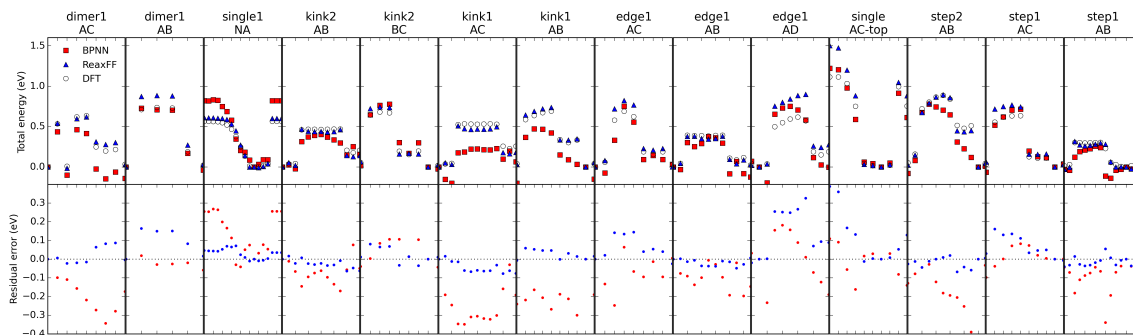


Figure 7: Residuals to all diffusion pathways of the fcc(100) surface. Structures are reproduced from those used in Ref. 7.

```

1  from ase.db import connect
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  db = connect('data.db')
6
7  C = {}
8  for d in db.select(['group=timo', 'config!=sl']):
9      if d.config not in C.keys():
10         C[d.config] = {}
11
12     if d.identity not in C[d.config].keys():
13         C[d.config][d.identity] = []
14
15     C[d.config][d.identity] += [[d.energy, d.neural_energy, d.reax_energy]]
16
17  f, ax = plt.subplots(2, 14, figsize=(20, 6))
18
19  i = 0
20  for cfg, data0 in C.iteritems():
21      for lbl, data1 in data0.iteritems():
22
23          Qe = np.array(data1).T[0]
24          m = list(Qe).index(Qe.min())
25          n = range(len(Qe))
26
27          Qe = Qe - Qe[m]
28          Ne = np.array(data1).T[1] - np.array(data1).T[1][m]
29          Re = np.array(data1).T[2] - np.array(data1).T[2][m]
30
31          Nerr = Ne - Qe
32          Rerr = Re - Qe
33
34          # Plotting energy points
35          ax[0, i].plot(n, Ne - Ne[m], 'rs', label='BPNN')
36          ax[0, i].plot(n, Re - Re[m], 'b^', label='ReaxFF')
37          ax[0, i].plot(n, Qe - Qe[m], 'ko', fillstyle='none', label='DFT')
38
39          # Plotting the residuals
40          ax[1, i].plot([min(n), max(n)], [0, 0], 'k:')
41          ax[1, i].plot(n, Nerr, 'r.')
42          ax[1, i].plot(n, Rerr, 'b.')
43
44          ax[0, i].set_xlim(min(n), max(n))

```

```

45         ax[1, i].set_xlim(min(n), max(n))
46         ax[1, i].set_ylim(-0.4, 0.4)
47         ax[0, i].set_ylim(-0.25, 1.6)
48         ax[0, i].set_title('{0}\n{1}'.format(cfg, lbl))
49
50         ax[0, i].set_xticklabels([])
51         ax[1, i].set_xticklabels([])
52         if i != 0:
53             ax[0, i].set_yticklabels([])
54             ax[1, i].set_yticklabels([])
55         i += 1
56
57 ax[1, 0].set_yticks(ax[1, 0].get_yticks()[:-2])
58 ax[1, 0].set_ylabel('Residual error (eV)')
59 ax[0, 0].set_ylabel('Total energy (eV)')
60 ax[0, 0].legend(loc='best', numpoints=1, fontsize=12, frameon=False)
61 plt.tight_layout(w_pad=0.1, h_pad=-0.3)
62 for ext in ['png', 'eps', 'pdf']:
63     plt.savefig('./images/si-full-diffusion.{0}'.format(ext), dpi=300)

```

3.2.4 Manuscript figure fig-111-slipping

```

1  from ase.db import connect
2  import matplotlib.pyplot as plt
3  import numpy as np
4  from scipy.interpolate import interp1d
5  from scipy.optimize import fmin
6  from ase.io import write
7  import matplotlib.image as mpimg
8  from matplotlib.offsetbox import OffsetImage, AnnotationBbox
9  import os
10
11 db = connect('data.db')
12
13 Qe, Re, Ne = [], [], []
14 for d in db.select(['miller=111', 'diffusion=slipping', 'config!=double', 'traj=0']):
15     Qe += [d.energy]
16     Ne += [d.neural_energy]
17     Re += [d.reax_energy]
18
19 Qe = np.array([Qe[-1]] + Qe) - Qe[-1]
20 Ne = np.array([Ne[-1]] + Ne) - Ne[-1]
21 Re = np.array([Re[-1]] + Re) - Re[-1]
22
23 x = np.linspace(0, len(Qe) - 1)
24
25 fig = plt.figure(figsize=(6, 4))
26 ax = fig.add_subplot(111)
27
28 for nrg, name, c, o in [[Re, 'ReaxFF', 'b^', 0.0],
29                        [Ne, 'BPNN', 'rs', 0.0],
30                        [Qe, 'DFT', 'ko', 0.0]]:
31     f = interp1d(range(len(nrg)), -nrg, 'cubic')
32     xmax = fmin(f, len(nrg) / 2., disp=False)
33
34     ax.plot(x, -f(x), c[0] + '--')
35     ax.plot(nrg, c, label=name)
36
37     ax.annotate('{0} $E^{\ddag}$= {1:1.2f}'.format(name, float(-f(xmax))),
38               xy=(xmax, -f(xmax)), xytext=(4.0, -f(xmax) + o),
39               ha='right', va='center', color=c[0],
40               arrowprops=dict(arrowstyle="->",
41                               shrinkB=10, color=c[0]))
42
43 for i, a in enumerate([[0, 1.9], [2, 5.1], [7, 8.3]]):

```



```

44     atoms = db.get_atoms(['miller=111',
45                           'diffusion=slipping',
46                           'config!=double',
47                           'traj=0', 'image={0}'.format(a[0])])
48     del atoms[0]
49     atoms *= (3, 3, 1)
50
51     write('./images/temp.png',
52           atoms,
53           colors=['#333333', '#999999', '#CCCCCC',] * 27,
54           scale=20,
55           show_unit_cell=2,
56           radii=0.75)
57
58     image = mping.imread('./images/temp.png')
59     imagebox = OffsetImage(image, zoom=1.5)
60
61     ax.add_artist(AnnotationBbox(imagebox,
62                                  xy=(a[0], nrg[a[0]]),
63                                  xybox=(a[1], 0.25),
64                                  pad=-0.2,
65                                  frameon=False,
66                                  arrowprops=dict(arrowstyle='->',
67                                                    color='0.5',
68                                                    zorder=5,
69                                                    connectionstyle='arc,angleA=-90,armA=0')
70                                  )
71     )
72     os.unlink('./images/temp.png')
73
74     ax.set_xticklabels([])
75     plt.xlabel('Reaction coordinate (a.u.)')
76     plt.ylabel('Potential energy (eV)')
77     plt.ylim(0, 0.3)
78     plt.tight_layout()
79     for ext in ['png', 'eps', 'pdf']:
80         plt.savefig('./images/fig-111-slipping.{0}'.format(ext), dpi=300)

```

3.2.5 Additional slipping barriers

To conserve space in the manuscript, only the fcc(111) single-layer surface slipping barrier is shown, while the fcc(100) single- and double-layer barriers are depicted in this section.

1. fcc(100) single-layer slipping barrier

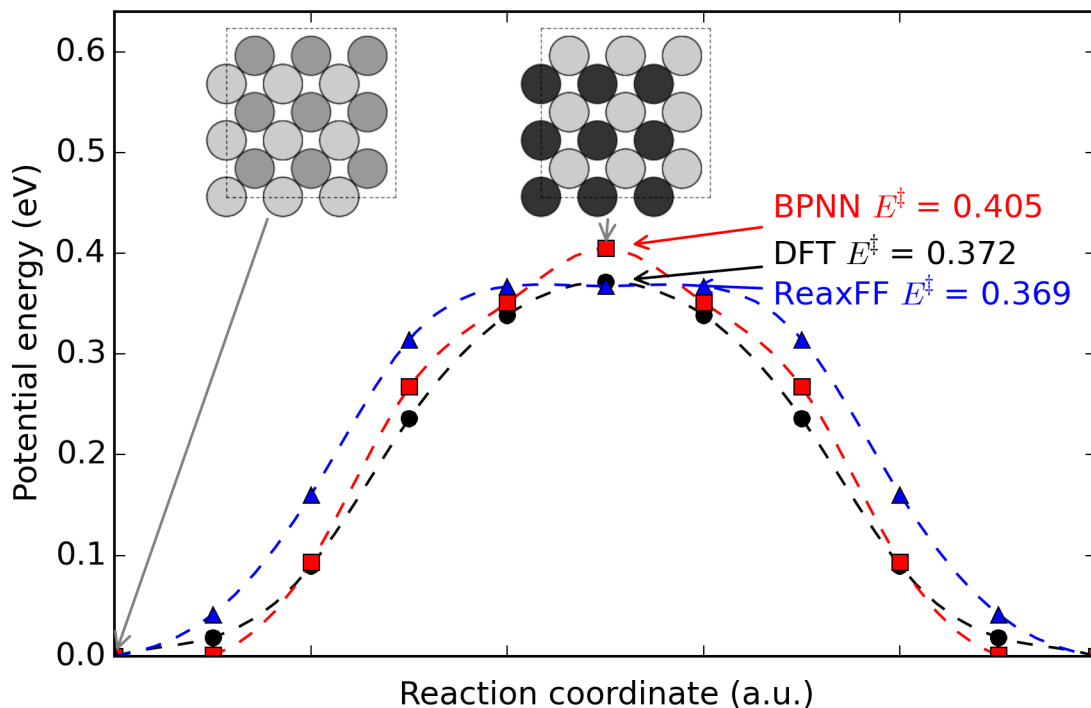


Figure 8: NEB predicted slipping barrier for single layer of Au fcc(100). Initial and top positions are shown for visual reference.

```

1  from ase.db import connect
2  import matplotlib.pyplot as plt
3  import numpy as np
4  from scipy.interpolate import interp1d
5  from scipy.optimize import fmin
6  from ase.io import write
7  import matplotlib.image as mping
8  from matplotlib.offsetbox import OffsetImage, AnnotationBbox
9  import os
10
11  db = connect('data.db')
12
13  Qe, Re, Ne = [], [], []
14  for d in db.select(['miller=100', 'diffusion=slipping', 'config!=double', 'traj=0']):
15      Qe += [d.energy]
16      Ne += [d.neural_energy]
17      Re += [d.reax_energy]
18
19  Qe = np.array([Qe[-1]] + Qe) - Qe[-1]
20  Ne = np.array([Ne[-1]] + Ne) - Ne[-1]
21  Re = np.array([Re[-1]] + Re) - Re[-1]
22
23  x = np.linspace(0, len(Qe) - 1)
24
25  fig = plt.figure(figsize=(6, 4))
26  ax = fig.add_subplot(111)
27
28  for nrg, name, c, o in [[Qe, 'DFT', 'ko', 0.0275],
29                          [Ne, 'BPNN', 'rs', 0.04],
30                          [Re, 'ReaxFF', 'b~', -0.010]]:
31      f = interp1d(range(len(nrg)), -nrg, 'cubic')
32      xmax = fmin(f, len(nrg) / 2., disp=False)

```

```

33
34     ax.plot(x, -f(x), c[0] + '--')
35     ax.plot(nrg, c, label=name)
36
37     ax.annotate('{0}  $E^{\ddagger}$  = {1:1.3f}'.format(name, float(-f(xmax))),
38                 xy=(xmax, -f(xmax)), xytext=(6.7, -f(xmax) + o),
39                 ha='left', va='center', color=c[0],
40                 arrowprops=dict(arrowstyle="->",
41                                 shrinkB=10, color=c[0]))
42
43 for i, a in enumerate([0, 1.9], [5, 5.1]):
44     atoms = db.get_atoms(['miller=100',
45                           'diffusion=slipping',
46                           'config!=double',
47                           'traj=0', 'image={0}'.format(a[0])])
48
49     del atoms[0]
50     atoms *= (3, 3, 1)
51
52     write('./images/temp.png',
53           atoms,
54           colors=['#333333', '#999999', '#CCCCCC',] * 27,
55           scale=20,
56           show_unit_cell=2,
57           radii=0.75)
58
59     image = mping.imread('./images/temp.png')
60     imagebox = OffsetImage(image, zoom=1.5)
61
62     ax.add_artist(AnnotationBbox(imagebox,
63                                  xy=(a[0], Ne[a[0]]),
64                                  xybox=(a[1], 0.53),
65                                  pad=-0.2,
66                                  frameon=False,
67                                  arrowprops=dict(arrowstyle='->',
68                                                  color='0.5',
69                                                  zorder=5,
70                                                  connectionstyle='arc,angleA=-90,armA=0')
71                                  )
72     os.unlink('./images/temp.png')
73
74 ax.set_xticklabels([])
75 plt.xlabel('Reaction coordinate (a.u.)')
76 plt.ylabel('Potential energy (eV)')
77 plt.ylim(0, 0.64)
78 plt.tight_layout()
79 for ext in ['png', 'eps', 'pdf']:
80     plt.savefig('./images/si-100-slipping.{0}'.format(ext), dpi=300)

```

2. fcc(100) double-layer slipping barrier

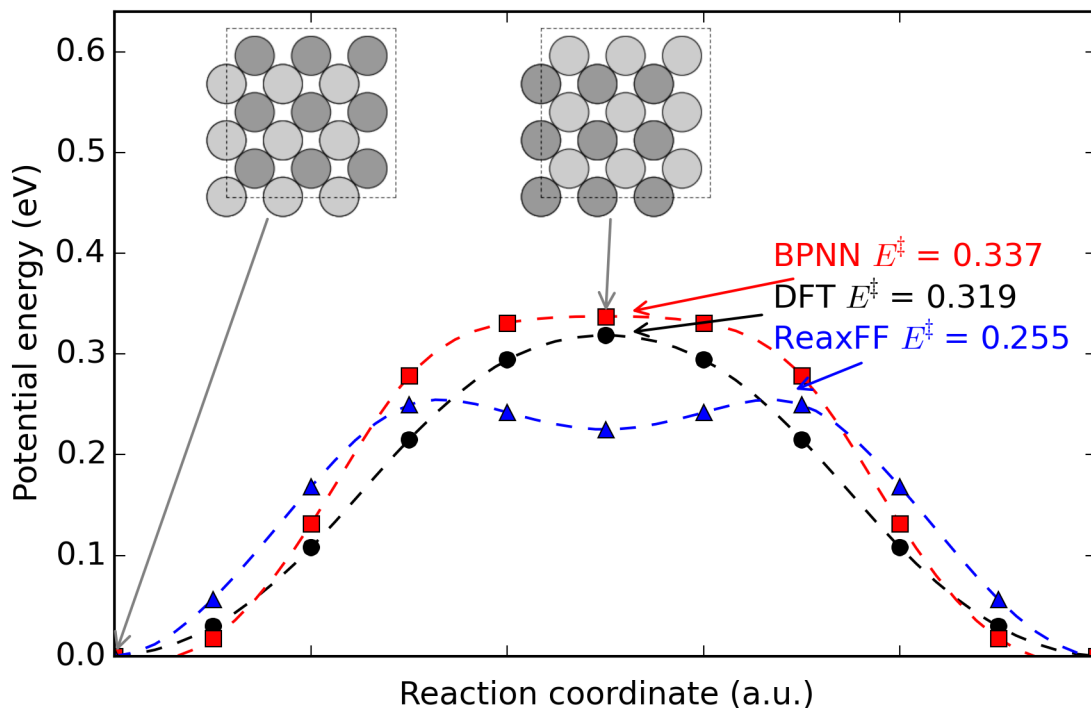


Figure 9: NEB predicted slipping barrier for a double layer of Au fcc(100). Initial and top positions are shown for visual reference.

```

1  from ase.db import connect
2  import matplotlib.pyplot as plt
3  import numpy as np
4  from scipy.interpolate import interp1d
5  from scipy.optimize import fmin
6  from ase.io import write
7  import matplotlib.image as mpimg
8  from matplotlib.offsetbox import OffsetImage, AnnotationBbox
9  import os
10
11  db = connect('data.db')
12
13  Qe, Re, Ne = [], [], []
14  for d in db.select(['miller=100', 'diffusion=slipping', 'config!=single', 'traj=0']):
15      Qe += [d.energy]
16      Ne += [d.neural_energy]
17      Re += [d.reax_energy]
18
19  Qe = np.array([Qe[-1]] + Qe) - Qe[-1]
20  Ne = np.array([Ne[-1]] + Ne) - Ne[-1]
21  Re = np.array([Re[-1]] + Re) - Re[-1]
22
23  x = np.linspace(0, len(Qe) - 1)
24
25  fig = plt.figure(figsize=(6, 4))
26  ax = fig.add_subplot(111)
27
28  for nrg, name, c, o in [[Qe, 'DFT', 'ko', 0.0375],
29                          [Ne, 'BPNN', 'rs', 0.06],
30                          [Re, 'ReaxFF', 'b~', 0.06]]:
31      f = interp1d(range(len(nrg)), -nrg, 'cubic')
32      xmax = fmin(f, len(nrg) / 2., disp=False)

```

```

33
34     ax.plot(x, -f(x), c[0] + '--')
35     ax.plot(nrg, c, label=name)
36
37     ax.annotate('{0} $E^{\ddag}$= {1:1.3f}'.format(name, float(-f(xmax))),
38                 xy=(xmax, -f(xmax)), xytext=(6.7, -f(xmax) + o),
39                 ha='left', va='center', color=c[0],
40                 arrowprops=dict(arrowstyle="->",
41                                 shrinkB=10, color=c[0]))
42
43 for i, a in enumerate([0, 1.9], [5, 5.1]):
44     atoms = db.get_atoms(['miller=100',
45                           'diffusion=slipping',
46                           'config!=single',
47                           'traj=0', 'image={0}'.format(a[0])])
48
49     del atoms[0]
50     atoms *= (3, 3, 1)
51
52     write('./images/temp.png',
53           atoms,
54           colors=['#333333', '#999999', '#CCCCC',] * 27,
55           scale=20,
56           show_unit_cell=2,
57           radii=0.75)
58
59     image = mpimg.imread('./images/temp.png')
60     imagebox = OffsetImage(image, zoom=1.5)
61
62     ax.add_artist(AnnotationBbox(imagebox,
63                                  xy=(a[0], Ne[a[0]]),
64                                  xybox=(a[1], 0.53),
65                                  pad=-0.2,
66                                  frameon=False,
67                                  arrowprops=dict(arrowstyle='->',
68                                                  color='0.5',
69                                                  zorder=5,
70                                                  connectionstyle='arc,angleA=-90,armA=0')
71                                  )
72
73     os.unlink('./images/temp.png')
74
75     ax.set_xticklabels([])
76     plt.xlabel('Reaction coordinate (a.u.)')
77     plt.ylabel('Potential energy (eV)')
78     plt.ylim(0, 0.64)
79     plt.tight_layout()
80     for ext in ['png', 'eps', 'pdf']:
81         plt.savefig('./images/si-100-slipping-2.{0}'.format(ext), dpi=300)

```

3.3 Cluster predictions

3.3.1 Manuscript figure fig-6atom-md

```

1  from matplotlib.offsetbox import OffsetImage, AnnotationBbox
2  import matplotlib.image as mpimg
3  from ase.io import write
4  import matplotlib.pyplot as plt
5  import numpy as np
6  from ase.db import connect
7  import os
8
9  db = connect('md-6atom.db')
10 Qe = dict.fromkeys(range(99, 2000, 100), 0)
11
12 I, Re, Ne = [], [], []

```

```

13 for d in db.select():
14     I += [d.image]
15     Ne += [d.neural_energy / d.natoms]
16     Re += [d.reax_energy / d.natoms]
17
18     # Only every 100th structure is DFT validated
19     if d.image in Qe.keys():
20         Qe[d.image] = d.energy / d.natoms
21
22 sort = [i[0] for i in sorted(enumerate(I), key=lambda x:x[1])]
23
24 I = np.array([I[i] for i in sort])
25 Ne = np.array([Ne[i] for i in sort])
26 Re = np.array([Re[i] for i in sort])
27
28 m = list(Ne).index(min(Ne))
29
30 # Create an image of the DFT predicted minimum
31 db0 = connect('data.db')
32 gm_atoms = db0.get_atoms(['cluster=plane-fcc', 'traj=0', 'config=2', 'natoms=6'])
33 gm = db0.get(['cluster=plane-fcc', 'traj=0', 'config=2', 'natoms=6'])
34 write('./images/tmp-gm.png', gm_atoms)
35
36 # Create an image of the BPNN predicted minimum
37 write('./images/tmp-pm.png', db.get_atoms(m), rotation='45y')
38
39 # Establish baseline for global minimum
40 Q_min = gm.energy / gm.natoms
41 N_min = gm.neural_energy / gm.natoms
42
43 fig = plt.figure(figsize=(6, 4))
44 ax = fig.add_subplot(111)
45
46 fig0 = OffsetImage(mping.imread('./images/tmp-pm.png'))
47 ax.add_artist(AnnotationBbox(fig0,
48                             xy=(m, Ne[m]),
49                             xybox=(1600, -1.8),
50                             pad=0,
51                             arrowprops=dict(arrowstyle='->',
52                                             color='0.5',
53                                             zorder=5,
54                                             connectionstyle='arc,angleA=0')
55                             ))
56
57 fig = OffsetImage(mping.imread('./images/tmp-gm.png'))
58 ax.add_artist(AnnotationBbox(fig,
59                             xy=(m, Q_min),
60                             xybox=(1400, -2.22),
61                             pad=0,
62                             arrowprops=dict(arrowstyle='->',
63                                             color='0.5',
64                                             zorder=5,
65                                             connectionstyle='arc,angleA=0')
66                             ))
67
68 # Remove the temporary images
69 os.unlink('./images/tmp-gm.png')
70 os.unlink('./images/tmp-pm.png')
71
72 ax.text(1700, -2.16, 'DFT', color='k')
73 ax.text(200, -2.27, 'ReaxFF', color='b')
74 ax.text(200, -1.9, 'BPNN', color='r')
75 ax.plot(range(len(Ne)), Re, 'b-')
76 ax.plot([0, len(Ne)], [N_min, N_min], 'r--')
77 ax.plot([0, len(Ne)], [Q_min, Q_min], 'k--')
78
79 for k, v in Qe.iteritems():
80     plt.plot([k, k], [v, Ne[k]], 'k:')


```

```

81
82 ax.plot(range(len(Ne)), Ne, 'r-')
83 ax.scatter(Qe.keys(), Qe.values(), color='k', marker='o', zorder=10)
84
85 ax.set_xlim(0, 2000)
86 ax.set_ylim(-2.3, -1.7)
87 plt.xlabel('Time step')
88 plt.ylabel('Potential energy (eV/atom)')
89 plt.tight_layout()
90 for ext in ['png', 'eps', 'pdf']:
91     plt.savefig('./images/fig-6atom-md.{0}'.format(ext), dpi=300)

```

3.3.2 6-atom MD simulations with 2000 data points

To demonstrate how the BPNN “learns” the potential energy surface as the number of training points increases, we have included a 6 atom MD simulation from one of the earlier BPNNs created. This BPNN was trained to all of the cluster calculations included in the full database up to 13 atoms large. However, no 6 atom clusters were used in the training set in order to observe how well these structures could be extrapolated. This is a training set of approximately 2000 calculations. The MD trajectory can be found here:  (double-click to open).

The resulting 6 atom MD simulation was performed identically to the one described in the paper. The results of the 4000 step MD are shown in Figure 10. For easier comparison, the energies are plotted on the same scale as the 6 atom MD simulation reported in the paper which used the full database as a training set. The result is a substantial improvement in the performance of the full BPNN. This demonstrates how a BPNN can be made to obtain arbitrary levels of accuracy even with a large variety of different structure types being used for training.

Although the errors of the early 6 atom MD simulation shown here are significantly larger, the BPNN still accurately predicts the planer structure to be the lowest energy configuration. Perhaps even more impressive is that it manages to do so without any information about 6 atom structures.

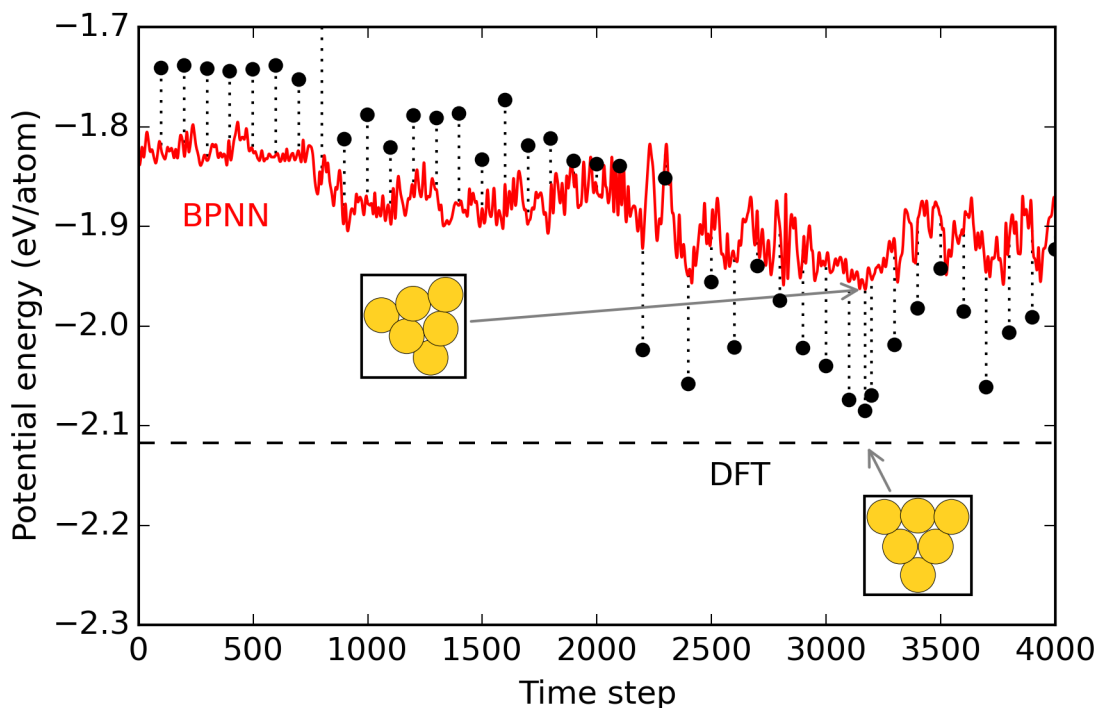


Figure 10: NVT MD simulation for the relaxation of the 6 atom unit cell from a local energy minima to the global minima. Temperature is scaled from 800 K to 300 K over the course of the simulation. Solid lines show MD trajectories while dashed lines show energy predictions for the global minima from DFT (black).

```

1  from matplotlib.offsetbox import OffsetImage, AnnotationBbox
2  import matplotlib.image as mpimg
3  from ase.io import write
4  import matplotlib.pyplot as plt
5  import numpy as np
6  from ase.db import connect
7  import os
8
9  db = connect('md-6atom-old.db')
10
11  Qe = {}
12  I, Ne, = [], []
13  for d in db.select():
14      I += [d.image]
15      Ne += [d.neural_energy / d.natoms]
16
17      if d.DFT:
18          Qe[d.image] = d.energy / d.natoms
19
20  sort = [i[0] for i in sorted(enumerate(I), key=lambda x:x[1])]
21
22  I = np.array([I[i] for i in sort])
23  Ne = np.array([Ne[i] for i in sort])
24
25  m = list(Ne).index(min(Ne))
26
27  # Create an image of the DFT predicted minimum
28  db0 = connect('data.db')
29  gm_atoms = db0.get_atoms(['cluster=plane-fcc', 'traj=0', 'config=2', 'natoms=6'])

```



```

30 gm = db0.get(['cluster=plane-fcc', 'traj=0', 'config=2', 'natoms=6'])
31 write('./images/tmp-gm.png', gm_atoms)
32
33 # Create an image of the BPNN predicted minimum
34 write('./images/tmp-pm.png', db.get_atoms(m), rotation='45y')
35
36 # Establish baseline for global minimum
37 Q_min = gm.energy / gm.natoms
38
39 fig = plt.figure(figsize=(6, 4))
40 ax = fig.add_subplot(111)
41
42 fig0 = OffsetImage(mping.imread('./images/tmp-pm.png'))
43 ax.add_artist(AnnotationBbox(fig0,
44                             xy=(m, Ne[m]),
45                             xybox=(1200, -2.0),
46                             pad=0,
47                             arrowprops=dict(arrowstyle='->',
48                                             color='0.5',
49                                             zorder=5,
50                                             connectionstyle='arc,angleA=0')
51                             ))
52
53 fig = OffsetImage(mping.imread('./images/tmp-gm.png'))
54 ax.add_artist(AnnotationBbox(fig,
55                             xy=(m, Q_min),
56                             xybox=(3400, -2.22),
57                             pad=0,
58                             arrowprops=dict(arrowstyle='->',
59                                             color='0.5',
60                                             zorder=5,
61                                             connectionstyle='arc,angleA=0')
62                             ))
63
64 # Remove the temporary images
65 os.unlink('./images/tmp-gm.png')
66 os.unlink('./images/tmp-pm.png')
67
68 ax.text(2500, -2.16, 'DFT', color='k')
69 ax.text(200, -1.9, 'BPNN', color='r')
70 ax.plot([0, len(Ne)], [Q_min, Q_min], 'k--')
71
72 for k, v in Qe.iteritems():
73     plt.plot([k, k], [v, Ne[k]], 'k:')
74
75 ax.plot(range(len(Ne)), Ne, 'r-')
76 ax.scatter(Qe.keys(), Qe.values(), color='k', marker='o', zorder=10)
77
78 ax.set_xlim(0, 4000)
79 ax.set_ylim(-2.3, -1.7)
80 plt.xlabel('Time step')
81 plt.ylabel('Potential energy (eV/atom)')
82 plt.tight_layout()
83 for ext in ['png', 'eps', 'pdf']:
84     plt.savefig('./images/si-6atom-md.{0}'.format(ext), dpi=300)

```

3.3.3 Manuscript figure fig-38atom-minima

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 from ase.db import connect
4
5 db = connect('data.db')
6
7 Qe, Re, Ne = [], [], []

```

```

8  for d in db.select(['post=minima']):
9      Qe += [d.energy / d.natoms]
10     Ne += [d.neural_energy / d.natoms]
11     Re += [d.reax_energy / d.natoms]
12
13 Re = np.array(Re)
14 Nre = np.array(Ne) - np.array(Qe)
15 Rre = Re - np.array(Qe)
16 C = np.array(range(len(Qe))) + 1
17
18 f, ax = plt.subplots(2, sharex=True, figsize=(6, 4))
19
20 ax[0].plot(C, Ne, mec='none', mfc='r', marker='s', lw=0, label='BPNN')
21 ax[0].plot(C, Re + 0.11, mec='none', mfc='b', marker='^', lw=0, label='ReaxFF')
22 ax[0].plot(C, Qe, mec='k', mfc='none', marker='o', lw=0, label='DFT')
23 ax[0].text(12, -2.6, 'ReaxFF offset by +0.11 eV/atom',
24           va='bottom', color='b', fontsize=12)
25 ax[1].plot([C[0], C[-1]], [0, 0], 'k--')
26 ax[1].scatter(C, Nre, c='r', marker='s', edgecolor='none')
27 ax[1].scatter(C, Rre, c='b', marker='^', edgecolor='none')
28
29 ax[0].set_yticks([-2.65, -2.63, -2.61, -2.59])
30 ax[1].set_xlabel('MD minima')
31 ax[0].set_ylabel('Potential energy\n(eV/atom)')
32 ax[1].set_ylabel('Residual error\n(eV/atom)')
33 ax[0].set_xlim(C[0], C[-1])
34 ax[1].set_xlim(C[0], C[-1])
35 ax[0].legend(loc='best', numpoints=1, fontsize=12, frameon=False)
36 plt.tight_layout(h_pad=-0.0)
37 for ext in ['png', 'eps', 'pdf']:
38     plt.savefig('./images/fig-38atom-minima.{0}'.format(ext), dpi=300)

```

4 TOC

This is the Table of Contents graphic.

```

1  from ase.db import connect
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from ase.io import write
5  from matplotlib.offsetbox import OffsetImage, AnnotationBbox
6  import matplotlib.image as mpimg
7  import os
8
9
10 db = connect('data.db')
11
12 Qe, Re, Ne = [], [], []
13 for d in db.select(['post=minima']):
14     Qe += [d.energy / d.natoms]
15     Ne += [d.neural_energy / d.natoms]
16     Re += [d.reax_energy / d.natoms]
17
18 DFT_time = db.get(['post=minima', 'config=121']).calc_time
19
20 pos = [(0.5, 0.8), (0.2, 0.2), (0.8, 0.2)]
21 pos2 = [(0.5, 0.6), (0.35, 0.4), (0.65, 0.4)]
22 rot = ["-15x", "-130z", "", ""]
23 col = ['k', 'r', 'b']
24 pad = [-0.1, -0.7, -0.7]
25
26 fig = plt.figure(figsize=(5.5, 5))
27 ax = fig.add_subplot(111)

```

```

28 for i, nrgs in enumerate([Qe, Re, Ne]):
29     ind = nrgs.index(min(nrgs))
30     matoms = db.get_atoms(['post=minima',
31                             'config={0}'.format(ind)])
32
33     write('./images/temp.png'.format(ind), matoms, rotation=rot[i])
34
35     image = mping.imread('./images/temp.png')
36     imagebox = OffsetImage(image, zoom=1.75)
37
38     ax.add_artist(AnnotationBbox(imagebox,
39                                   xy=pos2[i],
40                                   xybox=pos[i],
41                                   pad=pad[i],
42                                   frameon=False,
43                                   arrowprops=dict(arrowstyle='<-',
44                                                       color=col[i],
45                                                       zorder=5)
46                                   )
47     )
48
49     os.unlink('./images/temp.png')
50
51     plt.text(pos2[0][0], pos2[0][1], 'DFT',
52             ha='center', size=20, va='top')
53
54     ax.annotate('Neural\nNetwork', xy=(0.45, 0.58), xytext=pos2[1],
55               va='bottom', ha='center', color='r', size=20,
56               arrowprops=dict(arrowstyle="<|-", fc='r',
57                               connectionstyle="arc3,rad=-0.5",))
58
59     ax.annotate('ReaxFF', xy=(0.55, 0.58), xytext=pos2[2],
60               va='bottom', ha='center', color='b', size=20,
61               arrowprops=dict(arrowstyle="<|-", fc='b',
62                               connectionstyle="arc3,rad=0.6",))
63
64     plt.text(0.5, 0.98, '{0:1.1f} hrs'.format(DFT_time/3600.),
65             va='center', ha='center', size=17)
66     plt.text(0.2, 0.01, '0.14 s', color='r',
67             va='center', ha='center', size=17)
68     plt.text(0.8, 0.01, '0.01 s', color='b',
69             va='center', ha='center', size=17)
70
71     fig.patch.set_visible(False)
72     ax.axis('off')
73     plt.tight_layout(rect=[-0.15, -0.03, 1.1, 1.02])
74     for ext in ['png', 'eps', 'pdf']:
75         plt.savefig('./images/toc.{0}'.format(ext), dpi=300)

```

References

- [1] John A. Keith, Donato Fantauzzi, Timo Jacob, and Adri C. T. van Duin. Reactive force-field for simulating gold surfaces and nanoparticles. *Physical Review B*, 81(23):235404, 2010.