# Import Packages

import numpy as np

import matplotlib.pyplot as plt

import matplotlib

# ASE imports

from ase import Atoms, units

from ase.build import supercells

from ase.visualize import view

from ase.build import fcc111, add\_adsorbate

from ase.io import read, write

from ase.calculators.vasp import Vasp

from ase.md.velocitydistribution import (MaxwellBoltzmannDistribution,

 Stationary, ZeroRotation)

# FLARE imports

from flare.bffs.sgp.\_C\_flare import SparseGP, NormalizedDotProduct, B2, \

 Structure

from flare.bffs.sgp import sparse\_gp

from flare.bffs.sgp import calculator

from flare.learners.otf import OTF

# Retreive Structure

atoms= read('OptBottomFix/surf.traj', index=':')

atoms= atoms[-1]

n\_atoms= len(atoms)

# MD Parameters

temperature= 1000

MaxwellBoltzmannDistribution(atoms, temperature\_K=temperature, force\_temp= True)

Stationary(atoms)

ZeroRotation(atoms)

md\_engine= 'VelocityVerlet'

md\_dict= {}

# Create sparse GP model.

cutoff = 5.00

sigma = 2.0

power = 2

kernel = NormalizedDotProduct(sigma, power)

cutoff\_function = "quadratic"

many\_body\_cutoffs = [cutoff]

radial\_basis = "chebyshev"

radial\_hyps = [0., cutoff]

cutoff\_hyps = []

descriptor\_settings = [2, 8, 3]

descriptor\_calculator = \

 B2(radial\_basis, cutoff\_function, radial\_hyps, cutoff\_hyps,

 descriptor\_settings)

sigma\_e = 0.01\*n\_atoms # 1 meV/atom

sigma\_f = 0.05

sigma\_s = 0.0006 # 0.1 GPa

# Wrap C++ model in Python.

rydberg\_to\_ev = 13.6056980659

species\_map = {8: 0, 22: 1}

single\_atom\_energies = {0: 0.0, 1: 0.0}

variance\_type = "SOR"

max\_iterations = 20

opt\_method = "L-BFGS-B"

bounds=[(None, None), (sigma\_e, None), (None, None), (sigma\_s, None)]

gp\_model = sparse\_gp.SGP\_Wrapper([kernel], [descriptor\_calculator], cutoff,

 sigma\_e, sigma\_f, sigma\_s,

 species\_map, single\_atom\_energies=single\_atom\_energies,

 variance\_type=variance\_type,

 opt\_method=opt\_method, bounds=bounds,

 max\_iterations=max\_iterations)

flare\_calculator = calculator.SGP\_Calculator(gp\_model)

cutoff= 350

k=1

dft\_calc = Vasp(xc= 'PBE',

 encut=cutoff,

 gga='PE',

 metagga='R2SCAN', #metagga R2SCAN

 lasph=True, #includes non-spherical contributions

 lmaxtau=6, #6 characterizes d orbitals, default value

 lmixtau=True, #Sends Kinetic energy through density mixer

 kpts=(k,k,1), #changing k-point values

 gamma=True, #using gamma centered points

 lwave=False,

 lcharg=False,

 ispin=2,

 ncore=10, #approximately sqrt(cores)

 lreal='Auto',

 ismear=0,

 sigma=0.1,

 #nupdown=2,

 ialgo=58,

 nelm=300,

 prec='accurate',

 ediff=1E-5,

 isym=0

 )

otf\_params = {'init\_atoms': list(range(len(atoms))),

 'output\_name': 'otf',

 'store\_dft\_output': None,

 'std\_tolerance\_factor': -0.05,

 'max\_atoms\_added' : n\_atoms,

 'update\_style': 'threshold',

 'update\_threshold' : 0.025,

 'train\_hyps': (20, 200),

 'min\_steps\_with\_model':5,

 'write\_model': 3} # If you will probably resume the training, please set to 3

test\_otf= OTF(atoms, dt= 0.001,

 number\_of\_steps=1000,

 dft\_calc=dft\_calc, md\_engine=md\_engine, md\_kwargs=md\_dict,

 flare\_calc=flare\_calculator,

 force\_only= True,

 \*\*otf\_params)

test\_otf.run()

#test\_otf= OTF.from\_checkpoint('otf\_checkpt.json')

#test\_otf.run()