Machine-learning accelerated geometry optimization in molecular simulation: Supporting information

Yilin Yang and John R. Kitchin*

Department of Chemical Engineering, Carnegie Mellon University

Omar A. Jiménez-Negró

Department of Chemical Engineering,

University of Puerto Rico-Mayagüez,

Mayagüez, PR 00681, Puerto Rico, USA

(Dated: March 6, 2021)

CONTENTS

INTRODUCTION

There are three parts in this supporting information:

- Example to use NN_ensemble_relaxer for molecule geomoetry optimization
- Figures for configurations used as the example in the manuscript
- Code to access the datasets for reproducing the main figures in the manuscript

EXAMPLE OF NN_ENSEMBLE_RELAXER UTILIZATION

Installation

We first download the NN_ensemble_relaxer repo from the github and compile the required file.

```
1 git clone https://github.com/yilinyang1/NN_ensemble_relaxer.git
2 cd utils && python libsymf_builder
```

Instantiate NN_ensemble_relaxer and run the relaxation

Assume we have a ASE database called `db` that contains the configurations that need to be optimized. We can just feed it into the `Ensemble_relaxer` to conduct the relaxation.

```
from nn_optimize import Ensemble_Relaxer

# NN hyperparameters
nn_params = {'layer_nodes': [40, 40], 'activations': ['tanh', 'tanh'], 'lr': 1}

# confidence coeffients used to control to what extent we trust the NN model
alpha = 2.0

# feed ASE database db, set groud truth calculator,
# specify the folder name to store intermediate models and data
relaxer = Ensemble_Relaxer(db=db, calculator=EMT(), jobname='AuPd-nano-test',
                           ensemble_size=10, alpha=alpha, nn_params=nn_params)

# relaxer.run() returns a ase-db containing relaxed configurations
relaxed_db = relaxer.run(fmax=0.05, steps=50)
```

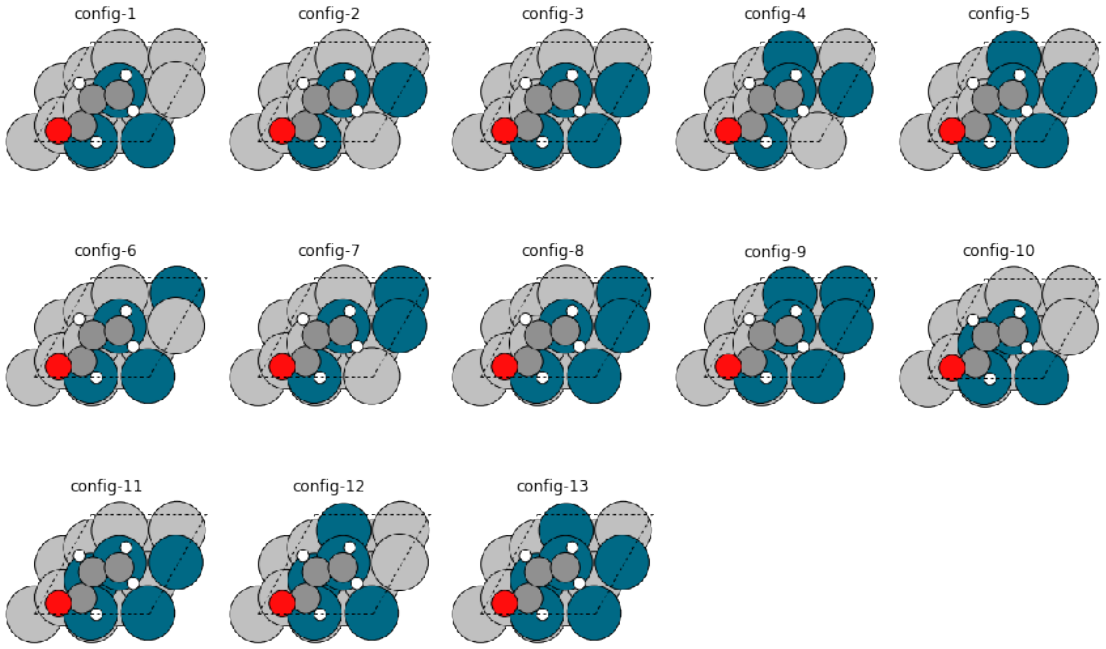
FIGURES FOR CONFIGURATIONS USED AS THE EXAMPLE IN THE MANUSCRIPT

In the second part of the Results section, we used 13 Acrolein/AgPd configurations as the example to show the advantage of multiple configurations during geometry optimization. Here, we provides the figures for these 13 configurations.

```
from ase.db import connect
from ase.io import write
from ase.visualize import view
import matplotlib.image as mpimg
from matplotlib import pyplot as plt
%matplotlib inline

path = 'Acrolein-AgPd-single-multiple-configs'
data_path = f'./{path}/initial-configs.db'
db = connect(data_path)
i = 1
for entry in db.select():
    write(f'./{path}/images/image-{i}.png', entry.toatoms())
    i += 1

```

```
fig, axes = plt.subplots(3, 5, figsize=(12, 8))
for i in range(15):
    row, col = i // 5, i % 5
    if i < 13:
        tmp_image = mpimg.imread(f'./{path}/images/image-{i+1}.png')
        axes[row][col].imshow(tmp_image)
        axes[row][col].set_title(f'config-{i+1}')
    axes[row][col].axis('off')
fig.tight_layout()
```

```
<Figure size 864x576 with 15 Axes>
```



CODE TO ACCESS THE DATASETS FOR REPRODUCING THE MAIN FIGURES IN THE MANUSCRIPT

This section shows how the figures in the manuscript were generated.

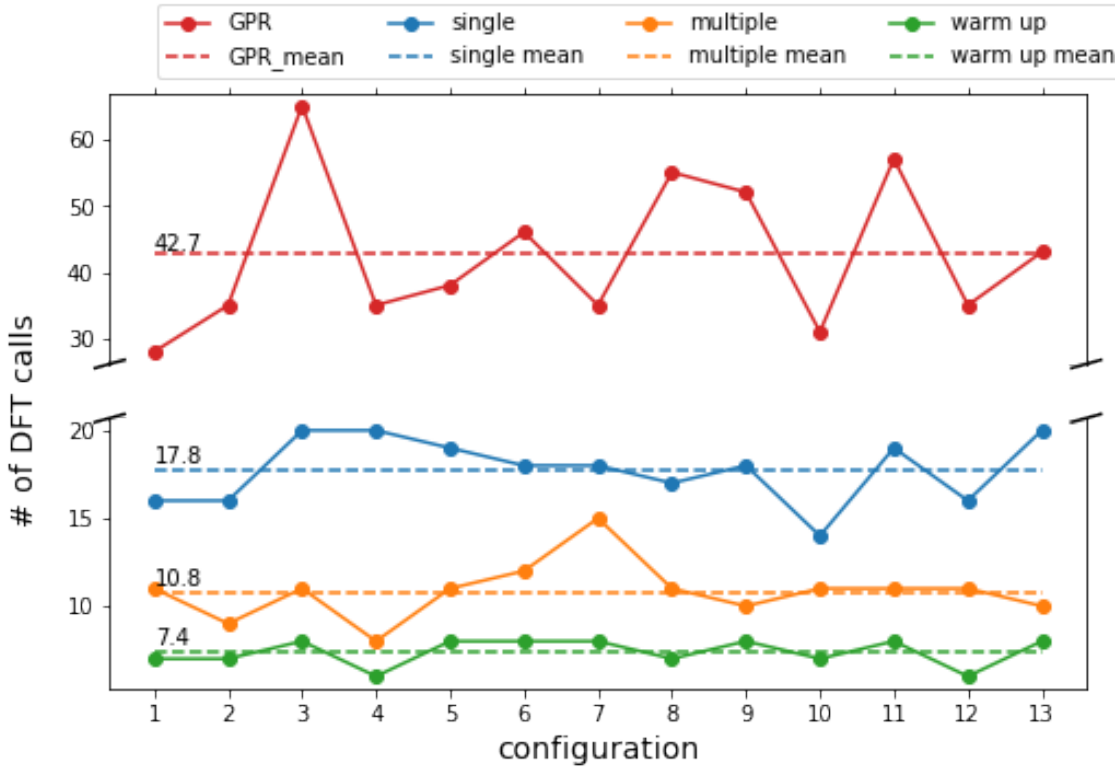
Acceleration of the geometry optimization for Acrolein/AgPd

```
from ase.io.trajectory import Trajectory
import numpy as np
from matplotlib import pyplot as plt
```

```
%matplotlib inline


single_steps = []
multi_steps = []
warm_steps = []
gpr_steps = []

for i in range(13):
    single_traj = Trajectory(f'./Acrolein-AgPd-single-multiple-configs/single-config-scratch-trajs/config-{i+1}.traj')
    single_steps.append(len(single_traj))
    multi_traj = Trajectory(f'./Acrolein-AgPd-single-multiple-configs/multi-config-scratch-trajs/config-{i+1}.traj')
    multi_steps.append(len(multi_traj))
    warm_traj = Trajectory(f'./Acrolein-AgPd-single-multiple-configs/multi-config-warmup-trajs/config-{i+1}.traj')
    warm_steps.append(len(warm_traj))
    gpr_traj = Trajectory(f'./Acrolein-AgPd-single-multiple-configs/gpr-trajs/config-{i+1}.traj')
    gpr_steps.append(len(gpr_traj))

steps = [single_steps, multi_steps, warm_steps, gpr_steps]
labels = ['single', 'multiple', 'warm up', 'GPR']
mean_labels = ['single mean', 'multiple mean', 'warm up mean', 'GPR_mean']
xs = range(1, 14)

f, (ax, ax2) = plt.subplots(2, 1, sharex=True, figsize=(8, 5), )
ax.plot(xs, steps[3], '-o', label = labels[3], color=f'C{3}')
means = [round(np.mean(steps[3]), 1)] * len(xs)
ax.plot(xs, means, '--', color=f'C{3}', label = mean_labels[3])
ax.text(1, means[0] + 0.4, str(means[0]))
ax.set_xticks(range(1, 14))

for i in range(3):
    ax2.plot(xs, steps[i], '-o', label = labels[i])
    means = [round(np.mean(steps[i]), 1)] * len(xs)
    ax2.plot(xs, means, '--', color=f'C{i}', label = mean_labels[i])
    ax2.text(1, means[0] + 0.4, str(means[0]))

ax2.set_xticks(range(1, 14))
ax.spines['bottom'].set_visible(False)
ax2.spines['top'].set_visible(False)
ax.xaxis.tick_top()
ax.tick_params(labeltop=False)
ax2.xaxis.tick_bottom()
d = .015
kwargs = dict(transform=ax.transAxes, color='k', clip_on=False)
ax.plot((-d, +d), (-d, +d), **kwargs)
ax.plot((1 - d, 1 + d), (-d, +d), **kwargs)
kwargs.update(transform=ax2.transAxes)
```

```
ax2.plot((-d, +d), (1 - d, 1 + d), **kwargs)
ax2.plot((1 - d, 1 + d), (1 - d, 1 + d), **kwargs)

ax2.set_xlabel('configuration', fontsize=14)
ax2.set_ylabel('# of DFT calls', fontsize=14)
ax2.yaxis.set_label_coords(-0.07, 1.0)
f.legend(bbox_to_anchor=(0.12, 1.0), ncol=4, loc='upper left')
f.savefig('./single-multi-warmup-gpr.png', dpi=300)
```

<Figure size 576x360 with 2 Axes>



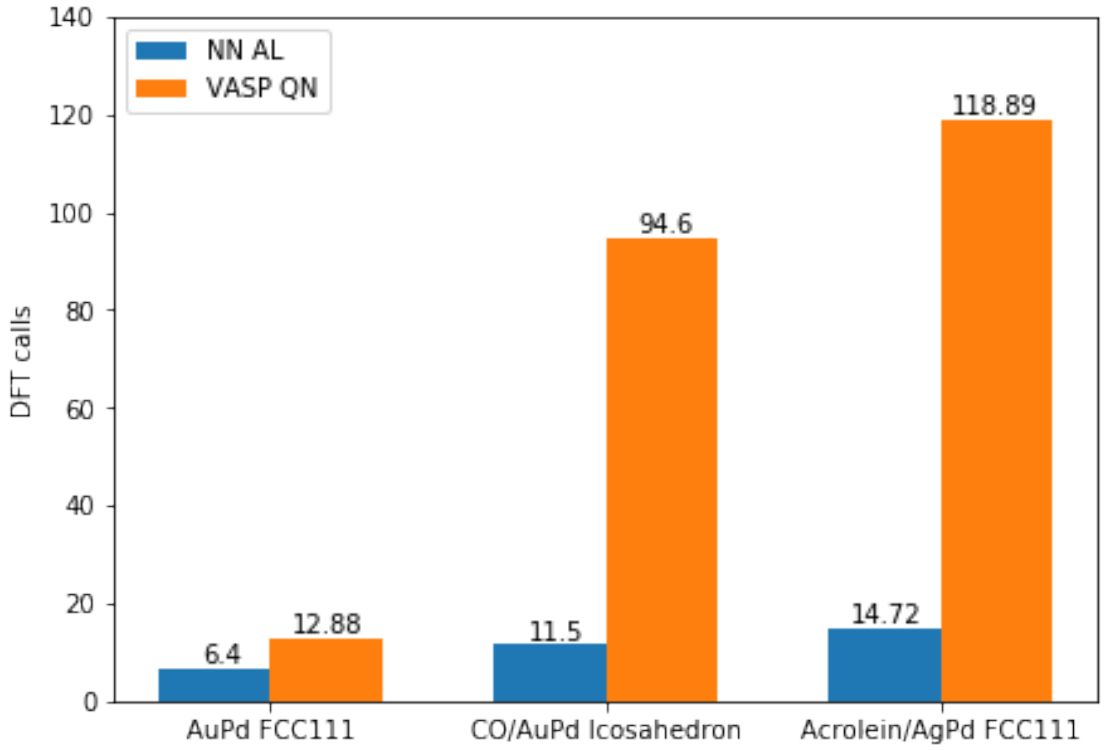
Performance for various systems

```
from matplotlib import pyplot as plt
import pickle
import numpy as np

path = 'more-geometry-optimization-data'

with open(f'./{path}/AuPd-slab-vasp-data-25.pkl', 'rb') as f:
    slab_vasp_data = pickle.load(f)

```

```
with open(f'./{path}/AuPd-slab-nn-data-25.pkl', 'rb') as f:
    slab_nn_data = pickle.load(f)

with open(f'./{path}/Acrolein-AgPd-vasp-data-100.pkl', 'rb') as f:
    ads_vasp_data = pickle.load(f)

with open(f'./{path}/Acrolein-AgPd-nn-data-scratch-100.pkl', 'rb') as f:
    ads_nn_data = pickle.load(f)

with open(f'./{path}/CO-AuPd-Ico-vasp-data-10.pkl', 'rb') as f:
    nano_vasp_data = pickle.load(f)

with open(f'./{path}/CO-AuPd-Ico-nn-data-10.pkl', 'rb') as f:
    nano_nn_data = pickle.load(f)

ads_nn_step_mean = np.mean(ads_nn_data['steps'])
ads_vasp_step_mean = np.mean(ads_vasp_data['steps'])

slab_nn_step_mean = np.mean(slab_nn_data['steps'])
slab_vasp_step_mean = np.mean(slab_vasp_data['steps'])

nano_nn_step_mean = np.mean(nano_nn_data['steps'])
nano_vasp_step_mean = np.mean(nano_vasp_data['steps'])

step_mean = np.array([slab_nn_step_mean, slab_vasp_step_mean, nano_nn_step_mean,
                      nano_vasp_step_mean, ads_nn_step_mean, ads_vasp_step_mean])

nn_steps = step_mean[[0, 2, 4]]
vasp_steps = step_mean[[1, 3, 5]]
nn_xs = [1, 2.5, 4]
vasp_xs = [1.5, 3, 4.5]

fig = plt.figure(figsize=(7, 5))
ax = fig.add_subplot(111)
ax.bar(nn_xs, nn_steps, width=0.5, label='NN AL')
ax.bar(vasp_xs, vasp_steps, width=0.5, label='VASP QN')
ax.set_ylabel('DFT calls')
ax.set_yticks(range(0, 160, 20))
ax.set_xticks([1.25, 2.75, 4.25])
ax.set_xticklabels(['AuPd FCC111', 'CO/AuPd Icosahedron', 'Acrolein/AgPd FCC111'])
ax.legend(loc='upper left')
ax.text(0.95, 7.5, str(nn_steps[0]))
ax.text(1.35, vasp_steps[0] + 1.1, str(vasp_steps[0]))
ax.text(2.4, nn_steps[1] + 1.1, str(nn_steps[1]))
ax.text(2.9, vasp_steps[1] + 1.1, str(vasp_steps[1]))
ax.text(3.85, nn_steps[2] + 1.1, str(nn_steps[2]))
ax.text(4.3, vasp_steps[2] + 1.1, str(vasp_steps[2]))
```

```
Text(4.3, 119.99, '118.89')
```

```
<Figure size 504x360 with 1 Axes>
```

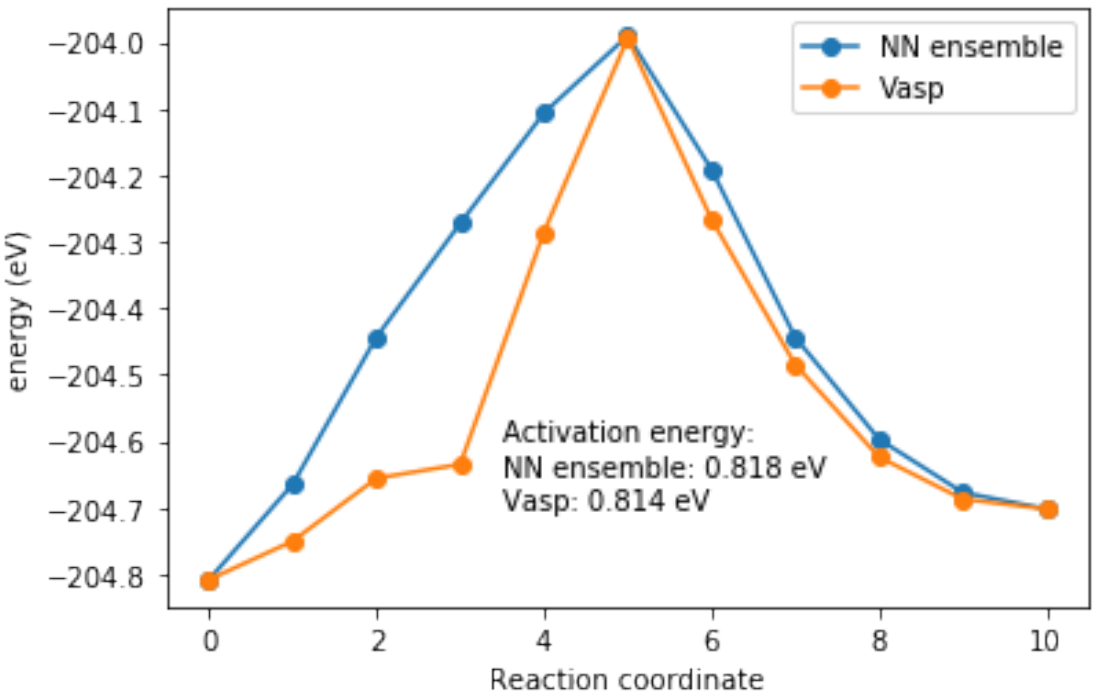



```python
from ase.db import connect
from matplotlib import pyplot as plt

init_db = connect('./Acetylele-hydrogenation-NEB/Acetylene-hydro-initial-configs.db')
vasp_db = connect('./Acetylele-hydrogenation-NEB/Acetylenen-hydro-vasp-cnvg.db')
nn_db = connect('./Acetylele-hydrogenation-NEB/Acetylene-hydro-nn-cnvg.db')

init_nrgs = [entry.energy for entry in init_db.select()]
vasp_nrgs = [entry.energy for entry in vasp_db.select()]
nn_nrgs = [entry.energy for entry in nn_db.select()]
xs = range(len(vasp_nrgs))

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(xs, nn_nrgs, '-o')
ax.plot(xs, vasp_nrgs, '-o')
ax.set_xlabel('Reaction coordinate')
ax.set_ylabel('energy (eV)')
```

```
ax.legend(['NN ensemble', 'Vasp'])
vasp_act = vasp_nrgs[5] - vasp_nrgs[0]
nn_act = nn_nrgs[5] - nn_nrgs[0]
ax.text(3.5, -204.6, 'Activation energy:')
ax.text(3.5, -204.65, f'NN ensemble: {round(nn_act, 3)} eV')
ax.text(3.5, -204.7, f'Vasp: {round(vasp_act, 3)} eV')
```

```
Text(3.5, -204.7, 'Vasp: 0.814 eV')
```

```
<Figure size 432x288 with 1 Axes>
```



GAUSSIAN PROCESS REGRESSION

We adapt the Gaussian Process Regression (GRP) method from previous literatures [1, 2] as one of the comparisons in this work. The GPR model uses the positions of the atoms as the feature $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_N]$, and the model is trained on the corresponding energies ($\mathbf{e}$) and the first order derivative which is the negative forces in this application. Thus, $\mathbf{y} = [\mathbf{e}, -\mathbf{f}_1, ..., -\mathbf{f}_N]$

Therefore, the prediction function could be sampled from the Gaussian Process defined by a prior mean and a kernel function:

$$f(x) \sim GP\left(\boldsymbol{\mu}, k\left(\boldsymbol{x}, \boldsymbol{x'}\right)\right) \tag{1}$$

where μ is the prior for the energies and forces and it is set as zero in our work. Given a training set D, the predicted mean and variance are

$$E\left[f(\boldsymbol{x}|D)\right] = \boldsymbol{k}(\boldsymbol{x})\left[\boldsymbol{K}(\boldsymbol{x} + \sigma_n^2\boldsymbol{I})\right]^{-1}\boldsymbol{y} \tag{2}$$

and

$$V\left[f(\boldsymbol{x}|D)\right] = k(\boldsymbol{x},\boldsymbol{x}) - \boldsymbol{k}(\boldsymbol{x})^T\left[\boldsymbol{K}(\boldsymbol{x} + \sigma_n^2\boldsymbol{I})\right]^{-1}\boldsymbol{k}(\boldsymbol{x}) \tag{3}$$

where σ_n is the noise of the data.

The kernel function could be partitioned into:

$$\boldsymbol{K}(\boldsymbol{x}) = \begin{pmatrix} \boldsymbol{K}_{ee}(\boldsymbol{x},\boldsymbol{x}) & \boldsymbol{K}_{ef}(\boldsymbol{x},\boldsymbol{x}) \\ \boldsymbol{K}_{ef}(\boldsymbol{x},\boldsymbol{x}) & \boldsymbol{K}_{ff}(\boldsymbol{x},\boldsymbol{x}) \end{pmatrix} \tag{4}$$

Squared exponential kernel is used in our implementation. Thus, the formula for these kernel function are:

$$k_{ee}(\boldsymbol{x},\boldsymbol{x'}) = \sigma_f^2 \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{(x_d - x'_d)^2}{l_d^2}\right) \tag{5}$$

$$k_{fe}(\boldsymbol{x},\boldsymbol{x'}) = -\frac{\sigma_f^2(x_d - x'_d)}{l_d^2}\exp\left(-\frac{1}{2}\sum_{j=1}^{D}\frac{(x_j - x'_j)^2}{l_j^2}\right) \tag{6}$$

$$k_{ff}(\boldsymbol{x},\boldsymbol{x'}) = \frac{-\sigma_f^2}{l_{d1}^2}\left(\delta_{d_1 d_2} - \frac{(x_{d1} - x'_{d1})(x_{d2} - x'_{d2})}{l_{d1}^2}\right)\exp\left(-\frac{1}{2}\sum_{j=1}^{D}\frac{(x_j - x'_j)^2}{l_j^2}\right) \tag{7}$$

where σ_f is fixed as 1.0, and the bandwidth l is optimized isotropically.

* jkitchin@andrew.cmu.edu

[1] J. A. G. Torres, P. C. Jennings, M. H. Hansen, J. R. Boes, and T. Bligaard, Physical Review Letters **122**, 156001 (2019).

[2] O.-P. Koistinen, F. B. Dagbjartsdóttir, V. Ásgeirsson, A. Vehtari, and H. Jónsson, The Journal of Chemical Physics **147**, 152720 (2017).