

# Approximating DFT Relaxation Energies with the Open Catalyst Project

1. Reads in file containing encoded bulks (catalyst compositions)
2. Converts bulks into ASE Atoms using Materials Project
3. Reads in files containing adsorbate 3D structures (from PubChem)
4. Converts adsorbates into ASE Atoms
5. Relaxes adsorbates on bulks using selected OCP Model, saves energies to disk

## Imports

```
In [1]: import sys
print(sys.version)
```

3.11.11 | packaged by Anaconda, Inc. | (main, Dec 11 2024, 16:34:19) [MSC v.1929 64 bit (AMD64)]

```
In [2]: import os
cwd = os.getcwd()
cwd
```

```
Out[2]: 'C:\\Users\\mecks\\Documents\\C02RR OCP'
```

```
In [3]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
In [4]: import json
import glob
import time
from tqdm import tqdm
from tqdm.notebook import tqdm
```

```
In [5]: from ase import Atoms
from ase.build import surface
from ase.optimize import BFGS
import ase.io
from ase.data import chemical_symbols
```

```
In [6]: if not hasattr(Atoms, "has_surface_tagged"):
    def _has_surface_tagged(self):
        return np.any(self.get_tags())
    Atoms.has_surface_tagged = _has_surface_tagged

if not hasattr(Atoms, "atoms"):
    Atoms.atoms = property(lambda self: self)
```

```
In [7]: from mp_api.client.core.client import MPRestError
from requests.exceptions import RequestException
```

```
In [8]: from fairchem.core.common.relaxation.ase_utils import OCPCalculator
from fairchem.core.models.model_registry import model_name_to_local_file
from fairchem.data.oc.core.adsorbate_slab_config import AdsorbateSlabConfig
```

```
In [9]: import torch, torchvision, torchaudio, platform
print("torch      :", torch.__version__)
print("torchvision:", torchvision.__version__)
```

```
print("torchaudio :", torchaudio.__version__)
print("CUDA avail :", torch.cuda.is_available())
print("Device      :", torch.cuda.get_device_name(0))
```

```
torch      : 2.4.0
torchvision: 0.19.0
torchaudio : 2.4.0
CUDA avail : True
Device     : NVIDIA GeForce RTX 4060
```

## User Controls

```
In [10]: from mp_api.client import MPRester
MP_KEY = "YOUR_API_KEY_HERE"
```

```
In [11]: use_cpu = False
individual_defined_bulks_mode = False
output_folder = "OCP_Relaxations"

if individual_defined_bulks_mode == True:

    individual_bulks_list = [['Cu'], # if you set individual_defined_bulks_mode = True, the
                             ['Zn'],
                             ['Ni'],
                             ['Fe'],
                             ['Mn'],
                             ['V'],
                             ['Co'],
                             ['Ti'],
                             ['Cr'],
                             ['Sc']
                             ]

    # Choose an OCP Model (below is not an exhaustive list):

    # OCP_model = 'GemNet-OC-S2EFS-OC20+OC22'
    # OCP_model = 'GemNet-OC-Large-S2EF-OC20-All+MD'
    # OCP_model = 'EquiformerV2-S2EF-ODAC'
    # OCP_model = 'DimeNet++-IS2RE-OC20-100k' #returned nothing
    # OCP_model = 'SchNet-S2EF-OC20-All'
    # OCP_model = 'PaiNN-IS2RE-OC20-All' #returned nothing
    # OCP_model = 'eSCN-IS2RE-ODAC' #returns nothing
    # OCP_model = 'SpinConv-S2EF-OC20-All' #fails
    # OCP_model = 'GemNet-OC-S2EFS-nsn-OC20+OC22'
    OCP_model = 'EquiformerV2-31M-S2EF-OC20-All+MD' #OCP Demo 'EquiformerV2'

    max_sites = 16 # max total heuristic adsorbate placements per slab
    bulk_timeout = 450 # seconds; maximum time the solver will spend on a single bulk before ea
```

## Read bulk CSV and extract element columns

```
In [12]: bulk_df = pd.read_csv("GPT_CO2_Modified.csv")
```

```
In [13]: valid_elements = set(chemical_symbols[1:])
element_columns = [col for col in bulk_df.columns if col in valid_elements]
print("Element columns found:", element_columns)
df_elements = bulk_df[element_columns]
```

```
Element columns found: ['Ag', 'Al', 'Au', 'Bi', 'Ca', 'Cd', 'Co', 'Cr', 'Cu', 'Fe', 'Ga', 'H', 'f', 'In', 'Ir', 'K', 'La', 'Mg', 'Mn', 'Mo', 'Nb', 'Ni', 'Pb', 'Pd', 'Pr', 'Pt', 'Rh', 'Ru', 'Sb', 'Sc', 'Se', 'Sn', 'Sr', 'Te', 'Ti', 'U', 'V', 'W', 'Zn', 'Zr', 'O']
```

Individual defined bulks mode replaces df\_elements (coming from Excel data) with only user-defined bulks located in individual\_bulks\_list

```
In [15]: if individual_defined_bulks_mode == True:
        rows = [{el: int(el in bulk) for el in element_columns} for bulk in individual_bulks_list]
        df_elements = pd.DataFrame(rows, columns=element_columns)
```

```
In [16]: # Remove duplicates so that each composition appears only once
df_elements_unique = df_elements.drop_duplicates().reset_index(drop=True)
print("Number of unique bulk compositions:", len(df_elements_unique))
```

Number of unique bulk compositions: 199

```
In [17]: df_elements_unique
```

```
Out[17]:
```

	Ag	Al	Au	Bi	Ca	Cd	Co	Cr	Cu	Fe	...	Sn	Sr	Te	Ti	U	V	W	Zn	Zr	O
0	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
194	0	0	0	0	0	0	0	0	0	0	...	0	1	0	1	0	0	0	0	0	1
195	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
196	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	0
197	0	0	0	0	0	0	0	0	0	0	...	0	1	0	1	0	0	0	0	0	1
198	0	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	0	0	0	0	1

199 rows x 40 columns

## Build a Bulk Slab from a Row in Data File using Materials Project

```
In [19]: def build_bulk_slab(
        row,
        miller=(1, 1, 1),
        layers=5,
        vacuum=10.0,
        ehull_tol=0.05,
        preferred_formulas=None,
        prefer_smallest=True,
    ):
    elems = [el for el in row.index if row[el] > 0]
    if not elems:
        return None, None

    chemsys = "-".join(sorted(elems))
    attempts = 3
    base_wait = 30
    docs = None
    for n in range(attempts):
        try:
            with MPRester(MP_KEY) as mpr:
                docs = mpr.materials.summary.search(
                    chemsys=chemsys,
```

```

        energy_above_hull=(0, ehull_tol),
        fields=["material_id", "formula_pretty", "energy_above_hull", "structure"],
        all_fields=False,
    )
    if not docs:
        docs = mpr.materials.summary.search(
            elements=elems,
            num_elements=[len(elems), None],
            energy_above_hull=(0, ehull_tol),
            fields=["material_id", "formula_pretty", "energy_above_hull", "structure"],
            all_fields=False,
        )
    break
except (MPRestError, ConnectionError, RequestException):
    if n == attempts - 1:
        return None, None
    time.sleep(base_wait * (2 ** n))

if not docs:
    return None, None

docs = sorted(docs, key=lambda d: d.energy_above_hull or 1e9)
chosen = docs[0]
if preferred_formulas:
    pref = {f: i for i, f in enumerate(preferred_formulas)}
    docs_pref = [d for d in docs if d.formula_pretty in pref]
    if docs_pref:
        chosen = min(docs_pref, key=lambda d: pref[d.formula_pretty])
elif prefer_smallest:
    chosen = min(docs, key=lambda d: d.structure.num_sites)

bulk_str = chosen.formula_pretty
slab = surface(chosen.structure.to_ase_atoms(), miller, layers=layers)
slab.center(vacuum=vacuum, axis=2)
return slab, bulk_str

```

## Parse adsorbate JSON files

In [20]: *# Parses a PubChem JSON file to extract adsorbate geometry*  
*# Returns an ASE Atoms object constructed from the first conformer*

```

def parse_adsorbate_json(json_file):
    with open(json_file, 'r') as f:
        data = json.load(f)
        compound = data["PC_Compounds"][0]
        elem_numbers = compound["atoms"]["element"]
        symbols = [chemical_symbols[num] for num in elem_numbers]
        conformer = compound["coords"][0]["conformers"][0]
        x_coords = conformer.get("x", [])
        y_coords = conformer.get("y", [])
        if "z" in conformer and conformer["z"]:
            z_coords = conformer["z"]
        else:
            z_coords = [0] * len(x_coords)
        positions = list(zip(x_coords, y_coords, z_coords))
        adsorbate = Atoms(symbols=symbols, positions=positions)

        adsorbate.binding_indices = [0]
    return adsorbate

```

In [21]: adsorbate\_files = glob.glob(os.path.join("pubchem\_ads", "\*.json"))  
adsorbate\_names = {os.path.basename(f).split('.')[0]: f for f in adsorbate\_files}  
print("Found adsorbate files:", adsorbate\_names)

Found adsorbate files: {'pubchem\_C00H': 'pubchem\_ads\\pubchem\_C00H.json'}

## Loading in Chosen OCP Model

```
In [22]: os.makedirs(output_folder, exist_ok=True)
```

Valid Models:

('CGCNN-S2EF-OC20-200k', 'CGCNN-S2EF-OC20-2M', 'CGCNN-S2EF-OC20-20M', 'CGCNN-S2EF-OC20-All', 'DimeNet-S2EF-OC20-200k', 'DimeNet-S2EF-OC20-2M', 'SchNet-S2EF-OC20-200k', 'SchNet-S2EF-OC20-2M', 'SchNet-S2EF-OC20-20M', 'SchNet-S2EF-OC20-All', 'DimeNet++-S2EF-OC20-200k', 'DimeNet++-S2EF-OC20-2M', 'DimeNet++-S2EF-OC20-20M', 'DimeNet++-S2EF-OC20-All', 'SpinConv-S2EF-OC20-2M', 'SpinConv-S2EF-OC20-All', 'GemNet-dT-S2EF-OC20-2M', 'GemNet-dT-S2EF-OC20-All', 'PaiNN-S2EF-OC20-All', 'GemNet-OC-S2EF-OC20-2M', 'GemNet-OC-S2EF-OC20-All', 'GemNet-OC-S2EF-OC20-All+MD', 'GemNet-OC-Large-S2EF-OC20-All+MD', 'SCN-S2EF-OC20-2M', 'SCN-t4-b2-S2EF-OC20-2M', 'SCN-S2EF-OC20-All+MD', 'eSCN-L4-M2-Lay12-S2EF-OC20-2M', 'eSCN-L6-M2-Lay12-S2EF-OC20-2M', 'eSCN-L6-M2-Lay12-S2EF-OC20-All+MD', 'eSCN-L6-M3-Lay20-S2EF-OC20-All+MD', 'EquiformerV2-83M-S2EF-OC20-2M', 'EquiformerV2-31M-S2EF-OC20-All+MD', 'EquiformerV2-153M-S2EF-OC20-All+MD', 'SchNet-S2EF-force-only-OC20-All', 'DimeNet++-force-only-OC20-All', 'DimeNet++-Large-S2EF-force-only-OC20-All', 'DimeNet++-S2EF-force-only-OC20-20M+Rattled', 'DimeNet++-S2EF-force-only-OC20-20M+MD', 'CGCNN-IS2RE-OC20-10k', 'CGCNN-IS2RE-OC20-100k', 'CGCNN-IS2RE-OC20-All', 'DimeNet-IS2RE-OC20-10k', 'DimeNet-IS2RE-OC20-100k', 'DimeNet-IS2RE-OC20-all', 'SchNet-IS2RE-OC20-10k', 'SchNet-IS2RE-OC20-100k', 'SchNet-IS2RE-OC20-All', 'DimeNet++-IS2RE-OC20-10k', 'DimeNet++-IS2RE-OC20-100k', 'DimeNet++-IS2RE-OC20-All', 'PaiNN-IS2RE-OC20-All', 'GemNet-dT-S2EFS-OC22', 'GemNet-OC-S2EFS-OC22', 'GemNet-OC-S2EFS-OC20+OC22', 'GemNet-OC-S2EFS-nsn-OC20+OC22', 'GemNet-OC-S2EFS-OC20->OC22', 'EquiformerV2-IE4-IF100-S2EFS-OC22', 'SchNet-S2EF-ODAC', 'DimeNet++-S2EF-ODAC', 'PaiNN-S2EF-ODAC', 'GemNet-OC-S2EF-ODAC', 'eSCN-S2EF-ODAC', 'EquiformerV2-S2EF-ODAC', 'EquiformerV2-Large-S2EF-ODAC', 'Gemnet-OC-IS2RE-ODAC', 'eSCN-IS2RE-ODAC', 'EquiformerV2-IS2RE-ODAC')

```
In [ ]: checkpoint_path = model_name_to_local_file(OCP_model, local_cache='/tmp/fairchem_checkpoints')
print("Using checkpoint:", checkpoint_path)
calc = OCPCalculator(checkpoint_path=checkpoint_path, cpu=False) #set cpu=False if using GP
```

## Run Relaxations and Save to Disk

```
In [24]: results_by_ads = {ads_name: [] for ads_name in adsorbate_names.keys()}
```

```
In [ ]: os.environ["MPI_NO_PROGRESS"] = "1"

global_progress = tqdm(total=len(adsorbate_names) * len(df_elements_unique), desc="Overall p
results_by_ads = {ads_name: [] for ads_name in adsorbate_names.keys()}

for ads_name, ads_file in adsorbate_names.items():
    try:
        adsorbate = parse_adsorbate_json(ads_file)
    except Exception:
        global_progress.update(len(df_elements_unique))
        continue
    pb_ads = tqdm(total=len(df_elements_unique), desc=f"Processing {ads_name}", leave=False,

    for _, row in df_elements_unique.iterrows():
        bulk_start = time.time()
        slab, bulk_str = build_bulk_slab(row)
        if slab is None:
            pb_ads.update(1)
```

```

        global_progress.update(1)
        continue

    bulk_info = {col: row[col] for col in element_columns}
    bulk_info["bulk_string"] = bulk_str

    heur_cfg = AdsorbateSlabConfig(slab, adsorbate, mode="heuristic")
    rand_cfg = AdsorbateSlabConfig(slab, adsorbate, mode="random_site_heuristic_placement")

    heur_sites = heur_cfg.atoms_list[: max_sites // 2]
    rand_sites = rand_cfg.atoms_list[: max_sites - len(heur_sites)]
    adslabs = heur_sites + rand_sites

    best_e = None
    start_time = time.time()
    for adslab in adslabs:
        if time.time() - bulk_start > bulk_timeout:
            tqdm.write(f"timeout {bulk_str}")
            break
        adslab.calc = calc
        BFGS(adslab, logfile=None).run(fmax=0.05, steps=300)
        e = adslab.get_potential_energy()
        best_e = e if best_e is None or e < best_e else best_e

    if best_e is None:
        pb_ads.update(1)
        global_progress.update(1)
        continue

    elapsed = int(time.time() - start_time)
    entry = bulk_info | {"relaxation_energy": best_e, "relaxation_time": f"{elapsed//360}"}
    results_by_ads[ads_name].append(entry)

    pb_ads.set_postfix({"Bulk": bulk_str, "Energy (eV)": f"{best_e:.3f}"})
    pb_ads.update(1)
    global_progress.update(1)

pb_ads.close()
pd.DataFrame(results_by_ads[ads_name]).to_csv(
    os.path.join(output_folder, f"'individual_' if individual_defined_bulks_mode else '
    index=False, sep="\t"
)
tqdm.write(f"Updated {ads_name}")

global_progress.close()
print("Final results saved in", output_folder)

```

In [ ]: