Two Quick Filtering Methods

1 Quick MACE Filtering

Step 1: Get data ready

Put all your .xyz files in one .xyz file, called combined_all_AuCeO.xyz for example.

Step 2: Train a little MACE

Option 1: From scratch

You could train a little MACE from the beginning.

```
mace_run_train \
    --name="lil_mace" \
    --model="MACE" \
    --num_channels=64 \
    --max_L=2 \
    --r_max=6.0 \
    --stress_weight=0.0 \
    --forces_weight=0.0 \
    --energy_weight=1.0 \
    --model_dir="MACE_models" \
    --log_dir="MACE_models" \
    --checkpoints_dir="MACE_models" \
    --results_dir="MACE_models" \
    --train_file="/work/e05/e05/uccayz3/Tom/lil_mace/combined_all_AuCe
O.xyz" \
    --valid_fraction=0.1 \
    --energy_key="energy_vasp" \
    --E0s="average" \
    --device=cpu \
    --batch_size=4 \
    --max_num_epochs=400 \
    --eval_interval=1 \
    --patience=99999 \
    --seed=345 \
    --ema \
    --ema_decay=0.995 \
    --default_dtype="float64"
```

Option 2: Finetune a pretrained MACE

Or you could finetune a pretrained MACE, for example MACE-MPA-0.

```
mace_run_train \
    --name="lil_mace" \
    --foundation_model="mace-mpa-0-medium.model" \
    --model="MACE" \
    --stress_weight=0.0 \
    --forces_weight=0.0 \
    --energy_weight=1.0 \
    --multiheads_finetuning=False \
    --model_dir="MACE_models" \
    --log_dir="MACE_models" \
    --checkpoints_dir="MACE_models" \
    --results_dir="MACE_models" \
    --train_file="/work/e05/e05/uccayz3/Tom/lil_mace/combined_all_AuCe
O.xyz" \
    --valid_fraction=0.1 \
    --energy_key="energy_vasp" \
    --E0s="average" \
    --device=cpu \
    --batch_size=4 \
    --max_num_epochs=400 \
    --eval_interval=1 \
    --patience=99999 \
    --seed=345 \
    --ema \
    --ema_decay=0.995 \
    --default_dtype="float64"
```

Step 3: Predict energy with lil_mace

```
import os
from ase.io import read
from mace.calculators import MACECalculator
import torch

torch.set_num_threads(1)
torch.set_num_interop_threads(1)

model_path = "/work/e05/e05/uccayz3/Tom/lil_mace/try/MACE_models/lil_
mace.model"
xyz_dir = "/work/e05/e05/uccayz3/Tom/2layer/xyz"
output_file = "/work/e05/e05/uccayz3/Tom/try_mace_predict/regular/mace
_predicted.txt"

start_idx = 0
end_idx = 100

calc = MACECalculator(model_paths=[model_path], device="cpu")

with open(output_file, "a") as f_out:
    total = end_idx - start_idx + 1
    print(f"A{start_idx}.xyz – A{end_idx}.xyz, {total} structures in total\n")

    for num in range(start_idx, end_idx + 1):
        file_name = f"A{num}.xyz"
        xyz_path = os.path.join(xyz_dir, file_name)

        if not os.path.exists(xyz_path):
            print(f"{file_name} skipped")
            continue

        try:
            atoms = read(xyz_path)
            atoms.calc = calc
            energy = atoms.get_potential_energy()

            f_out.write(f"{num}  {energy:.6f}\n")
            f_out.flush()
        except Exception as e:
            print(f"failed {file_name}: {e}")
            continue

print(f"\n A{start_idx}.xyz – A{end_idx}.xyz done, {output_file} written")
```

Now you could filter them based on their predicted energies by yourself.

2 Quick REMatch Filtering

Quick REMatch Filtering is designed to eliminate excessive identical or similar structures.

Step 1: Convert all .gin files to SOAP

You need to write a loop by yourself to call the following functions to convert all the structures (.gin) into SOAP (.csv).

```
import numpy as np
from ase import Atoms
from dscribe.descriptors import SOAP

def parse_gin_to_atoms(gin_path):
    with open(gin_path, 'r') as gin_file:
        lines = gin_file.readlines()

    symbols = []
    positions = []
    for line in lines:
        parts = line.strip().split()
        if len(parts) == 11:
            element = parts[0]
            x, y, z = float(parts[2]), float(parts[3]), float(parts[4])
            symbols.append(element)
            positions.append((x, y, z))

    atoms = Atoms(symbols=symbols, positions=positions)
    return atoms
```

```
def generate_soap_descriptor(atoms, csv_file):
    target_elements = ['Cu'] #change this
    target_indices = [atom.index for atom in atoms if atom.symbol in target_e
lements]

        # set these based on your structures
    species = ["Cu", "Zn", "O"]
    r_cut = 10.0
    n_max = 2
    l_max = 2

    soap = SOAP(
        species=species,
        periodic=False,
        r_cut=r_cut,
        n_max=n_max,
        l_max=l_max,
        compression={'mode':'mu1nu1','species_weighting':None}
    )

    soap_descriptors = soap.create(atoms, centers=target_indices)
    np.savetxt(csv_file, soap_descriptors, delimiter=',')

def process_gin(gin_path, output_path):
    atoms = parse_gin_to_atoms(gin_path)
    generate_soap_descriptor(atoms, output_path)
```

Step 2: Find your THRESHOLD

Find two structures that you consider similar by yourself, and then use the following code to obtain the similarity value between them. And that will be the THRESHOLD.

```
import numpy as np
from dscribe.kernels import REMatchKernel
from sklearn.preprocessing import normalize
```

```
def load_soap_from_csv(csv_path):
    """
    read SOAP (.csv), return np
    """
    return np.loadtxt(csv_path, delimiter=',')

def soap_rematch_similarity(soap1, soap2, gamma=1.0, alpha=1.0, threshol
d=1e-6):
    """
    input two SOAP, return REMatch
    """
    soap1_norm = normalize(soap1)
    soap2_norm = normalize(soap2)

    re = REMatchKernel(metric="rbf",gamma=gamma,alpha=alpha,threshold
=threshold)

    kernel_matrix = re.create([soap1_norm, soap2_norm])

    return kernel_matrix[0, 1]

def soap_rematch_similarity_from_files(csv_path1, csv_path2, gamma=1.0, a
lpha=1.0, threshold=1e-6):
    soap1 = load_soap_from_csv(csv_path1)
    soap2 = load_soap_from_csv(csv_path2)
    return soap_rematch_similarity(soap1, soap2, gamma=gamma, alpha=alp
ha, threshold=threshold)



if __name__ == "__main__":
    file1 = "/work/e05/e05/uccayz3/Tom/soap/X0.csv"
    file2 = "/work/e05/e05/uccayz3/Tom/soap/X1.csv"

    sim = soap_rematch_similarity_from_files(file1, file2, gamma=1.0, alpha=1.
0)
    print("SOAP REMatch similarity:", sim)
```

Step 3: Filter by REMatch

After you run this, you will fine we you need in SELECTED_LIST_PATH.

```
import os
from get_rematch import soap_rematch_similarity_from_files

SOAP_DIR = "/work/e05/e05/uccayz3/Tom/quick_soap"
LOG_PATH = "/work/e05/e05/uccayz3/Tom/python/rematch_log.txt"
SELECTED_LIST_PATH = "/work/e05/e05/uccayz3/Tom/python/selected_st
ructures.txt"

THRESHOLD = 0.9 # Set this THRESHOLD

GAMMA = 1.0
ALPHA = 1.0
THRESHOLD_RE = 1e-6

def main():
    all_files = sorted(
        f for f in os.listdir(SOAP_DIR)
        if f.endswith(".csv")
    )

    n_files = len(all_files)
    if n_files == 0:
        print(f"No .csv files found in {SOAP_DIR}")
        return

    print(f"Found {n_files} SOAP files in {SOAP_DIR}")

    active = [True] * n_files

    with open(LOG_PATH, "w") as log_f:
        log_f.write("fileA\tfileB\tREMatch\n")

        for i in range(n_files):
            if not active[i]:
                continue
            fileA = all_files[i]
            pathA = os.path.join(SOAP_DIR, fileA)

            print(f"Reference: {fileA} (index {i})")

            for j in range(i + 1, n_files):
                if not active[j]:
                    continue

                fileB = all_files[j]
                pathB = os.path.join(SOAP_DIR, fileB)

                sim = soap_rematch_similarity_from_files(
                    pathA,
                    pathB,
                    gamma=GAMMA,
                    alpha=ALPHA,
                    threshold=THRESHOLD_RE,
                )

                log_f.write(f"{fileA}\t{fileB}\t{sim:.8f}\n")
                log_f.flush()

                if sim > THRESHOLD:
                    active[j] = False
                    print(f"  Remove {fileB} (index {j}) due to high similarity: {sim:.4
f}")

    selected = [fname for flag, fname in zip(active, all_files) if flag]

    with open(SELECTED_LIST_PATH, "w") as f_sel:
        for fname in selected:
            f_sel.write(fname + "\n")

    print("Filtering done.")
    print(f"Original number of structures: {n_files}")
    print(f"Number of selected structures: {len(selected)}")
    print(f"Pairwise similarity log saved to: {LOG_PATH}")
    print(f"Selected structures list saved to: {SELECTED_LIST_PATH}")


if __name__ == "__main__":
    main()
```