30 trials

30 iterations per trial

10 random initialised points

Benchmarking\_alt\_trials.py

setup\_files\_alt import \*

from initialising\_points import \*

from baybe\_models import \*

from bofire\_setup import run\_mobo\_optimization

def run\_benchmark\_baybe(num\_trials):

"""

Main function to run the benchmark for SOBO and MOBO models.

"""

import copy

# Define models and their campaigns

models\_to\_run = {

"SOBO": {

"model\_fn": Models.run\_sobo\_loop,

"campaign": campaign\_sobo,

"use\_campaign": True,

},

"MOBO": {

"model\_fn": Models.run\_mobo\_loop,

"campaign": campaign\_mobo,

"use\_campaign": True,

},

"Bofire": {

"model\_fn": run\_mobo\_optimization,

"mobo\_strategy": mobo\_strategy\_bofire,

"use\_campaign": False,

},

}

emulator = get\_pretrained\_reizman\_suzuki\_emulator(case=1)

#no. iterations for each optimisation run

iterations = 30

# to hold full data from several runs

all\_results = {

"Trials": [],

"Models": [],

"Results": [],

}

for trial in range(1, num\_trials+1):

print(f"Starting trial {trial}/{num\_trials}")

bofire\_initial\_conditions = initialise\_random\_point(domain=domain\_bofire)

df\_random\_initialised\_point = pd.DataFrame(bofire\_initial\_conditions)

initial\_conditions = df\_random\_initialised\_point.rename(columns=name\_map)

for model\_name, model\_config in models\_to\_run.items():

print(f"Running {model\_name} for Trial {trial}")

if model\_name=="Bofire":

results = run\_mobo\_optimization(emulator = emulator,mobo\_strategy = mobo\_strategy\_bofire, bofire\_initial\_conditions = bofire\_initial\_conditions, experimental\_budget=iterations-1)

else:

campaign = copy.deepcopy(model\_config["campaign"])

results = model\_config["model\_fn"](emulator, campaign, iterations, initial\_conditions)

all\_results["Trials"].append(trial)

all\_results["Models"].append(model\_name)

all\_results["Results"].append(results)

results\_df = pd.DataFrame(all\_results)

with open('benchmark\_results\_multi\_trial.pkl', 'wb') as f:

pickle.dump(results\_df, f)

return results\_df

results\_runs\_baybe = run\_benchmark\_baybe(num\_trials=30)

Setup\_files\_alt.py

import summit

from summit.benchmarks import get\_pretrained\_reizman\_suzuki\_emulator

from summit.benchmarks.experimental\_emulator import ReizmanSuzukiEmulator

from summit.utils.dataset import DataSet

import pandas as pd

import numpy as np

import os

import pathlib

import matplotlib.pyplot as plt

from baybe.parameters import CategoricalParameter, NumericalContinuousParameter, SubstanceParameter

from baybe.targets import NumericalTarget

from baybe.recommenders import RandomRecommender, SequentialGreedyRecommender, TwoPhaseMetaRecommender, BotorchRecommender

from baybe.surrogates import GaussianProcessSurrogate

from baybe import Campaign

from baybe.objective import Objective

#from baybe.acquisition import ExpectedImprovement # see others at https://emdgroup.github.io/baybe/\_autosummary/baybe.acquisition.acqfs.html#module-baybe.acquisition.acqfs

#from baybe.acquisition import debotorchize

from baybe.searchspace import SearchSpace, SearchSpaceType, SubspaceDiscrete

#from baybe.acquisition.acqfs import ExpectedImprovement

from botorch import acquisition

import bofire

import botorch

import torch

import summit

import numpy as np

import pandas as pd

import time

import os

import multiprocessing

import importlib

import matplotlib.pyplot as plt

import numpy as np

import numpy.exceptions

import bofire.strategies.api as strategies

from bofire.data\_models.domain.api import Domain

from bofire.data\_models.domain.api import Inputs, Outputs

from bofire.data\_models.features.api import (

ContinuousInput,

ContinuousOutput,

CategoricalInput,

CategoricalDescriptorInput,)

from pprint import pprint as pp

from bofire.data\_models.objectives.api import MinimizeObjective, MaximizeObjective

from bofire.data\_models.strategies.api import MoboStrategy

from bofire.data\_models.acquisition\_functions.api import qEHVI, qLogEHVI

from bofire.data\_models.strategies.api import SoboStrategy

from bofire.data\_models.acquisition\_functions.api import qEI

from bofire.data\_models.strategies.api import (

RandomStrategy as RandomStrategyModel,

)

#Baybe set-up

catalyst\_smiles = pd.read\_csv(pathlib.Path.cwd() / pathlib.Path("suzuki\_miyaura\_catalysts.csv"))

ligand\_smiles = pd.read\_csv(pathlib.Path.cwd() / pathlib.Path("suzuki\_miyaura\_ligands.csv"))

#Transforming categorical variable (catalyst) into chemical variable = need smiles representation for each catalyst

emulator = get\_pretrained\_reizman\_suzuki\_emulator(case=1)

available\_catalysts = {

"P1-L1": f"{catalyst\_smiles[catalyst\_smiles['name'] == 'P1']['smiles'].values[0]}.{ligand\_smiles[ligand\_smiles['name'] == 'L1']['smiles'].values[0]}",

"P1-L2": f"{catalyst\_smiles[catalyst\_smiles['name'] == 'P1']['smiles'].values[0]}.{ligand\_smiles[ligand\_smiles['name'] == 'L2']['smiles'].values[0]}",

"P1-L3": f"{catalyst\_smiles[catalyst\_smiles['name'] == 'P1']['smiles'].values[0]}.{ligand\_smiles[ligand\_smiles['name'] == 'L3']['smiles'].values[0]}",

"P1-L4": f"{catalyst\_smiles[catalyst\_smiles['name'] == 'P1']['smiles'].values[0]}.{ligand\_smiles[ligand\_smiles['name'] == 'L4']['smiles'].values[0]}",

"P1-L5": f"{catalyst\_smiles[catalyst\_smiles['name'] == 'P1']['smiles'].values[0]}.{ligand\_smiles[ligand\_smiles['name'] == 'L5']['smiles'].values[0]}",

"P1-L6": f"{catalyst\_smiles[catalyst\_smiles['name'] == 'P1']['smiles'].values[0]}.{ligand\_smiles[ligand\_smiles['name'] == 'L6']['smiles'].values[0]}",

"P1-L7": f"{catalyst\_smiles[catalyst\_smiles['name'] == 'P1']['smiles'].values[0]}.{ligand\_smiles[ligand\_smiles['name'] == 'L7']['smiles'].values[0]}",

"P2-L1": f"{catalyst\_smiles[catalyst\_smiles['name'] == 'P2']['smiles'].values[0]}.{ligand\_smiles[ligand\_smiles['name'] == 'L1']['smiles'].values[0]}",

}

#Defining parameter space

parameters = [

SubstanceParameter(

name="catalyst",

data=available\_catalysts,

encoding="MORDRED"

),

NumericalContinuousParameter(

name="catalyst\_loading",

bounds=(0.5,2.0),

),

NumericalContinuousParameter(

name="temperature",

bounds=(30,110),

),

NumericalContinuousParameter(

name="t\_res",

bounds=(1,10),

)

]

#print('Parameters defined')

#Defining search space

searchspace = SearchSpace.from\_product(parameters)

#print('Search space defined')

target\_1 = NumericalTarget(name="yld", mode=f"MAX", bounds=(0,100), transformation="LINEAR")

target\_2 = NumericalTarget(name="ton", mode=f"MAX", bounds=(0,100), transformation="LINEAR")

targets\_sobo = [target\_1]

targets\_mobo = [target\_1, target\_2]

objective\_sobo = Objective(mode="SINGLE", targets = targets\_sobo)

#print('Objective defined (sobo)')

objective\_mobo = Objective(mode="DESIRABILITY", targets = targets\_mobo, weights=[50,50], combine\_func="GEOM\_MEAN")

#print('Objective defined (mobo)')

recommender = TwoPhaseMetaRecommender(

initial\_recommender=RandomRecommender(),

recommender=BotorchRecommender()

)

#print('Recommender defined')

campaign\_sobo = Campaign(

searchspace=searchspace,

objective=objective\_sobo,

recommender=recommender

)

#print('Campaign (sobo) defined')

campaign\_mobo = Campaign(

searchspace=searchspace,

objective=objective\_mobo,

recommender=recommender

)

#print('Campaign (mobo) defined')

#BoFire set-up

# We wish the temperature of the reaction to be between 30 and 110 °C

temperature\_feature = ContinuousInput(

key="Temperature", bounds=[30.0, 110.0], unit="°C"

)

# Catalyst Loading

catalyst\_loading\_feature = ContinuousInput(

key="Catalyst Loading", bounds=[0.5, 2], unit="%"

)

# Residence Time

residence\_time\_feature = ContinuousInput(

key="Residence Time", bounds=[1 \* 60, 10 \* 60], unit="minutes"

)

# Catalyst choice

catalyst\_feature = CategoricalInput(

key="Catalyst",

categories=[

"P1-L1",

"P2-L1",

"P1-L2",

"P1-L3",

"P1-L4",

"P1-L5",

"P1-L6",

"P1-L7",

],

)

# gather all individual features

input\_features = Inputs(

features=[

temperature\_feature,

catalyst\_loading\_feature,

residence\_time\_feature,

catalyst\_feature,

]

)

name\_map = {

"Catalyst Loading": "catalyst\_loading",

"Residence Time": "t\_res",

"Temperature": "temperature",

"Catalyst": "catalyst",

"Yield": "yld",

}

candidates = pd.DataFrame(

{

"Catalyst Loading": [0.498],

"Residence Time": [600],

"Temperature": [30],

"Catalyst": ["P1-L3"],

}

).rename(columns=name\_map)

emulator = summit.get\_pretrained\_reizman\_suzuki\_emulator(case=1)

conditions = summit.DataSet.from\_df(candidates)

results = emulator.run\_experiments(conditions, rtn\_std=True).rename(

columns=dict(zip(name\_map.values(), name\_map.keys())),

)

experiments = pd.DataFrame(

{

"Catalyst Loading": results["Catalyst Loading"],

"Residence Time": results["Residence Time"],

"Temperature": results["Temperature"],

"Catalyst": results["Catalyst"],

"Yield": results["Yield"],

"valid\_Yield": 1,

}

)

max\_objective = MaximizeObjective(w=1.0)

min\_objective = MinimizeObjective(w=1.0, bounds=[0, 200])

yield\_feature = ContinuousOutput(key="Yield", objective=max\_objective)

ton\_feature = ContinuousOutput(key="TON", objective=min\_objective)

# create an output feature

output\_features = Outputs(features=[yield\_feature, ton\_feature])

domain\_bofire = Domain(

inputs=input\_features,

outputs=output\_features,

)

# a multi objective BO strategy

#qExpectedImprovement = qEHVI()

qLogExpectedImprovement = qLogEHVI()

mobo\_strategy\_data\_model\_bofire = MoboStrategy(

domain=domain\_bofire,

acquisition\_function=qLogExpectedImprovement,

)

# map the strategy data model to the actual strategy that has functionality

mobo\_strategy\_bofire = strategies.map(mobo\_strategy\_data\_model\_bofire)

#this function should be general (i.e. mobo & sobo) due to the dynamic target value extraction

def perform\_df\_experiment(data\_df: pd.DataFrame, emulator: ReizmanSuzukiEmulator, objective) -> dict:

conditions = DataSet.from\_df(data\_df)

#print(conditions)

emulator\_output = emulator.run\_experiments(conditions, return\_std=True)

result\_df = data\_df.copy()

for target in objective.targets:

target\_name = target.name # Get the name of the target

# Find the column corresponding to the target\_name in the emulator\_output

if target\_name in emulator\_output.columns:

target\_value = emulator\_output[target\_name].values[0]

result\_df[target\_name] = target\_value # Add the target to the result DataFrame

else:

raise ValueError(f"Target column '{target\_name}' not found in emulator output.")

#print(result\_df)

return result\_df

#this function should be general (i.e. mobo & sobo) due to the dynamic target value extraction

def perform\_df\_experiment\_multi(data\_df: pd.DataFrame, emulator: ReizmanSuzukiEmulator, objective) -> dict:

conditions = DataSet.from\_df(data\_df)

#print(conditions)

#emulator = get\_pretrained\_reizman\_suzuki\_emulator(case=1)

emulator\_output = emulator.run\_experiments(conditions, return\_std=True)

result\_df = data\_df.copy()

for target in objective.targets:

target\_name = target.name # Get the name of the target

# Find the column corresponding to the target\_name in the emulator\_output

if target\_name in emulator\_output.columns:

target\_values = emulator\_output[target\_name].values

#result\_df[target\_name] = emulator\_output[target\_name].values # Add the target to the result DataFrame

result\_df[target\_name] = pd.to\_numeric(target\_values, errors='coerce')

else:

raise ValueError(f"Target column '{target\_name}' not found in emulator output.")

#print(result\_df)

return result\_df

def evaluate\_candidates(candidates: pd.DataFrame) -> pd.DataFrame:

"""Evaluate the candidates using the Reizman-Suzuki emulator.

Parameters:

candidates: A DataFrame with the experiments.

Returns:

A DataFrame with the experiments and the predicted yield.

"""

name\_map = {

"Catalyst Loading": "catalyst\_loading",

"Residence Time": "t\_res",

"Temperature": "temperature",

"Catalyst": "catalyst",

"Yield": "yld",

"TON": "ton",

}

candidates = candidates.rename(columns=name\_map)

#emulator = summit.get\_pretrained\_reizman\_suzuki\_emulator(case=1)

conditions = summit.DataSet.from\_df(candidates)

emulator\_output = emulator.run\_experiments(

conditions, rtn\_std=True

).rename(columns=dict(zip(name\_map.values(), name\_map.keys())))

# Check if 'TON' exists in the output

if 'TON' not in emulator\_output.columns:

print("Warning: 'TON' column not found in emulator output.")

# Optionally, add a default value for TON or raise an error

emulator\_output['TON'] = np.nan # Or handle as appropriate

return pd.DataFrame(

{

"Catalyst Loading": emulator\_output["Catalyst Loading"],

"Residence Time": emulator\_output["Residence Time"],

"Temperature": emulator\_output["Temperature"],

"Catalyst": emulator\_output["Catalyst"],

"Yield": emulator\_output["Yield"],

"valid\_Yield": np.ones(len(emulator\_output.index)),

"TON": emulator\_output["TON"],

"valid\_TON": np.ones(len(emulator\_output.index)),

}

)

Initialising\_points.py

from setup\_files\_alt import \*

from sklearn.preprocessing import MinMaxScaler

import pickle

'''Using the random strategy model from bofire to initialise points for all 3 models!'''

def initialise\_random\_point(domain):

random\_strategy\_model = RandomStrategyModel(domain=domain)

random\_strategy = strategies.map(random\_strategy\_model)

candidates = random\_strategy.ask(10)

return candidates

random\_initialised\_points = initialise\_random\_point(domain=domain\_bofire)

df\_random\_initialised\_point = pd.DataFrame(random\_initialised\_points)

df\_random\_initialised\_point\_renamed = df\_random\_initialised\_point.rename(columns=name\_map)

initial\_conditions = df\_random\_initialised\_point\_renamed

bofire\_initial\_conditions = random\_initialised\_points

Baybe\_models.py

from setup\_files\_alt import \*

'''

Making a new version to attempt to incorporate the util function into the baybe loops!

'''

class Models:

"""Class containing 3 bayesian objective models:

Baybe SOBO, Baybe MOBO & BoTorch MOBO"""

@staticmethod

def run\_sobo\_loop(

emulator: summit.benchmarks.experimental\_emulator.ReizmanSuzukiEmulator,

campaign,

iterations: int,

initial\_conditions\_df,

):

"""

Single-objective bayesian optimisation using the BayBe back end

emulator: Summit experimental emulator

campaign: the campaign defined for the optimisation

iterations: the number of cycles/iterations to be completed

"""

#clear the stored measurements between each trial

campaign.\_measurements\_exp = pd.DataFrame()

results\_baybe\_sobo = []

cumulative\_max\_df = pd.DataFrame(columns=["Iteration", "Cumulative Max YLD"])

times\_df\_sobo = pd.DataFrame(columns=["Iteration", "Time\_taken"])

print("Starting the SOBO loop...")

parameter\_columns = [param.name for param in searchspace.parameters]

data\_df = pd.DataFrame(columns=parameter\_columns)

#print(f"Initial conditions - randomly generated: {initial\_conditions\_df}")

target\_measurement = perform\_df\_experiment\_multi(initial\_conditions\_df, emulator, objective=objective\_sobo)

campaign.add\_measurements(target\_measurement)

# Record the first step

results\_baybe\_sobo.append({

"iteration": 0,

"measurements": target\_measurement

})

#print(results\_baybe\_sobo)

#initialising a max.

cumulative\_max\_yld = float('-inf')

for i in range(1, iterations+1):

#print(campaign)

print(f"Running experiment {i }/{iterations}")

t1 = time.time()

recommended\_conditions = campaign.recommend(batch\_size=1)

#print(f"Recommended conditions: {recommended\_conditions}")

data\_df = pd.concat([data\_df, recommended\_conditions], ignore\_index=True)

target\_measurement = perform\_df\_experiment(recommended\_conditions, emulator, objective=objective\_sobo)

campaign.add\_measurements(target\_measurement)

print('measurements in campaign!',campaign.measurements)

time\_taken = time.time() - t1

#print(f"Iteration {i} took {(time.time() - t1):.2f} seconds")

#eval\_df\_sobo = evaluate\_candidates(target\_measurement)

new\_yld = target\_measurement['yld'].values[0]

print(new\_yld)

if new\_yld > cumulative\_max\_yld:

cumulative\_max\_yld = new\_yld

print(cumulative\_max\_yld)

cumulative\_max\_df = pd.concat([cumulative\_max\_df, pd.DataFrame([{

"Iteration": i,

"Cumulative Max YLD": cumulative\_max\_yld

}])], ignore\_index=True)

results\_baybe\_sobo.append({

"iteration": i ,

"measurements": target\_measurement

})

print(f"Iteration {i} took {time\_taken:.2f} seconds")

times\_df\_sobo = pd.concat(

[times\_df\_sobo, pd.DataFrame({"Iteration": [i], "Time\_taken": [time\_taken]})],

ignore\_index=True

)

return campaign.measurements, cumulative\_max\_df, times\_df\_sobo # cumulative\_max\_yld, #results\_baybe\_sobo,

@staticmethod

def run\_mobo\_loop(

emulator: summit.benchmarks.experimental\_emulator.ReizmanSuzukiEmulator,

campaign,

iterations: int,

initial\_conditions\_df,

):

"""

Multi-objective bayesian optimisation using the BayBe back end

emulator: Summit experimental emulator

campaign: the campaign defined for the optimisation

iterations: the number of cycles/iterations to be completed

"""

results\_baybe\_mobo = []

cumulative\_max\_df = pd.DataFrame(columns=["Iteration", "Cumulative Max YLD", "Cumulative Max TON"])

times\_df\_mobo = pd.DataFrame(columns=["Iteration", "Time\_taken"])

print("Starting the BayBE MOBO loop...")

parameter\_columns = [param.name for param in searchspace.parameters]

data\_df = pd.DataFrame(columns=parameter\_columns)

#print(f"Initial conditions - randomly generated: {initial\_conditions\_df}")

target\_measurement = perform\_df\_experiment\_multi(initial\_conditions\_df, emulator, objective=objective\_mobo)

campaign.add\_measurements(target\_measurement)

# Record the first step

results\_baybe\_mobo.append({

"iteration": 0,

"measurements": target\_measurement

})

#print(results\_baybe\_mobo)

cumulative\_max\_yld = float('-inf')

cumulative\_max\_ton = float('-inf')

for i in range(1, iterations+1):

#print(campaign)

print(f"Running experiment {i }/{iterations}")

t1 = time.time()

recommended\_conditions = campaign.recommend(batch\_size=1)

#print(f"Recommended conditions: {recommended\_conditions}")

data\_df = pd.concat([data\_df, recommended\_conditions], ignore\_index=True)

#target\_measurement = perform\_df\_experiment\_multi(recommended\_conditions, emulator, objective=objective\_mobo)

target\_measurement = perform\_df\_experiment(recommended\_conditions, emulator, objective=objective\_mobo)

campaign.add\_measurements(target\_measurement)

print('measurements in campaign!',campaign.measurements)

time\_taken = time.time() - t1

new\_yld = target\_measurement['yld'].values[0]

print('new yld',new\_yld)

new\_ton = target\_measurement['ton'].values[0]

print('new ton',new\_ton)

if new\_yld > cumulative\_max\_yld:

cumulative\_max\_yld = new\_yld

print('cumulative yld',cumulative\_max\_yld)

if new\_ton > cumulative\_max\_ton:

cumulative\_max\_ton = new\_ton

print('cumulative ton',cumulative\_max\_ton)

cumulative\_max\_df = pd.concat([cumulative\_max\_df, pd.DataFrame([{

"Iteration": i,

"Cumulative Max YLD": cumulative\_max\_yld,

"Cumulative Max TON": cumulative\_max\_ton

}])], ignore\_index=True)

results\_baybe\_mobo.append({

"iteration": i ,

"measurements": target\_measurement

})

print(f"Iteration {i} took {time\_taken:.2f} seconds")

times\_df\_mobo = pd.concat(

[times\_df\_mobo, pd.DataFrame({"Iteration": [i], "Time\_taken": [time\_taken]})],

ignore\_index=True

)

return campaign.measurements, cumulative\_max\_df, times\_df\_mobo #cumulative\_max\_yld, cumulative\_max\_ton #results\_baybe\_mobo,

Bofire\_setup

import bofire

import botorch

import torch

import summit

import numpy as np

import pandas as pd

import time

import os

import pickle

from setup\_files\_alt import evaluate\_candidates

import multiprocessing

import importlib

import matplotlib.pyplot as plt

import numpy as np

import numpy.exceptions

import bofire.strategies.api as strategies

from bofire.data\_models.domain.api import Domain

from bofire.data\_models.domain.api import Inputs, Outputs

from bofire.data\_models.features.api import (

ContinuousInput,

ContinuousOutput,

CategoricalInput,

CategoricalDescriptorInput,)

from pprint import pprint as pp

from bofire.data\_models.objectives.api import MinimizeObjective, MaximizeObjective

from bofire.data\_models.strategies.api import MoboStrategy

from bofire.data\_models.acquisition\_functions.api import qEHVI, qLogEHVI

from bofire.data\_models.strategies.api import SoboStrategy

from bofire.data\_models.acquisition\_functions.api import qEI

from bofire.data\_models.strategies.api import (

RandomStrategy as RandomStrategyModel,

)

from initialising\_points import bofire\_initial\_conditions

#from benchmarking\_alt import bofire\_initial\_conditions

# We wish the temperature of the reaction to be between 30 and 110 °C

temperature\_feature = ContinuousInput(

key="Temperature", bounds=[30.0, 110.0], unit="°C"

)

# Catalyst Loading

catalyst\_loading\_feature = ContinuousInput(

key="Catalyst Loading", bounds=[0.5, 2], unit="%"

)

# Residence Time

residence\_time\_feature = ContinuousInput(

key="Residence Time", bounds=[1 \* 60, 10 \* 60], unit="minutes"

)

# Catalyst choice

catalyst\_feature = CategoricalInput(

key="Catalyst",

categories=[

"P1-L1",

"P2-L1",

"P1-L2",

"P1-L3",

"P1-L4",

"P1-L5",

"P1-L6",

"P1-L7",

],

)

# gather all individual features

input\_features = Inputs(

features=[

temperature\_feature,

catalyst\_loading\_feature,

residence\_time\_feature,

catalyst\_feature,

]

)

name\_map = {

"Catalyst Loading": "catalyst\_loading",

"Residence Time": "t\_res",

"Temperature": "temperature",

"Catalyst": "catalyst",

"Yield": "yld",

}

candidates = pd.DataFrame(

{

"Catalyst Loading": [0.498],

"Residence Time": [600],

"Temperature": [30],

"Catalyst": ["P1-L3"],

}

).rename(columns=name\_map)

#print(candidates)

emulator = summit.get\_pretrained\_reizman\_suzuki\_emulator(case=1)

conditions = summit.DataSet.from\_df(candidates)

results = emulator.run\_experiments(conditions, rtn\_std=True).rename(

columns=dict(zip(name\_map.values(), name\_map.keys())),

)

experiments = pd.DataFrame(

{

"Catalyst Loading": results["Catalyst Loading"],

"Residence Time": results["Residence Time"],

"Temperature": results["Temperature"],

"Catalyst": results["Catalyst"],

"Yield": results["Yield"],

"valid\_Yield": 1,

}

)

max\_objective = MaximizeObjective(w=1.0)

#min\_objective = MinimizeObjective(w=1.0, bounds=[0, 200])

max\_objective\_2 = MaximizeObjective(w=1.0, bounds=[0, 200])

yield\_feature = ContinuousOutput(key="Yield", objective=max\_objective)

ton\_feature = ContinuousOutput(key="TON", objective=max\_objective\_2)

# create an output feature

output\_features = Outputs(features=[yield\_feature, ton\_feature])

domain = Domain(

inputs=input\_features,

outputs=output\_features,

)

# a multi objective BO strategy

#qExpectedImprovement = qEHVI()

qLogExpectedImprovement = qLogEHVI()

mobo\_strategy\_data\_model = MoboStrategy(

domain=domain,

acquisition\_function=qLogExpectedImprovement,

)

# map the strategy data model to the actual strategy that has functionality

mobo\_strategy = strategies.map(mobo\_strategy\_data\_model)

# a random strategy

#random\_strategy\_model = RandomStrategyModel(domain=domain)

# we have to provide the strategy with our optimization problem so it knows where to sample from = initial 5 random samples?

#random\_strategy = strategies.map(random\_strategy\_model)

#candidates = random\_strategy.ask(5)

#experiments = evaluate\_candidates(candidates)

'''

candidates = bofire\_initial\_conditions

experiments = evaluate\_candidates(candidates)

mobo\_strategy.tell(experiments, replace=True, retrain=True)

experimental\_budget = 5 # edit as needed

i = 0

done = False

results\_df = pd.DataFrame(columns=["Catalyst Loading", "Residence Time", "Temperature",

"Catalyst", "Yield", "valid\_Yield", "TON", "valid\_TON"])

while not done:

i += 1

t1 = time.time()

# ask for a new experiment

new\_candidate = mobo\_strategy.ask(1)

new\_experiment = evaluate\_candidates(new\_candidate)

mobo\_strategy.tell(new\_experiment)

print(f"Iteration took {(time.time()-t1):.2f} seconds")

# inform the strategy about the new experiment

# experiments = pd.concat([experiments,new\_experiment],ignore\_index=True)

# Add new results to DataFrame

results\_df = pd.concat([results\_df, new\_experiment], ignore\_index=True)

# Calculate cumulative max for Yield and TON

results\_df["Cumulative\_Max\_Yield"] = results\_df["Yield"].cummax()

results\_df["Cumulative\_Max\_TON"] = results\_df["TON"].cummax()

print(f"Iteration {i} took {(time.time() - t1):.2f} seconds")

print("Current Results:")

print(results\_df[["Yield", "Cumulative\_Max\_Yield", "TON", "Cumulative\_Max\_TON"]])

if i > experimental\_budget:

done = True

results= mobo\_strategy.experiments

print(mobo\_strategy.experiments)

print("results incl. cumulative max:", results\_df[["Yield", "Cumulative\_Max\_Yield", "TON", "Cumulative\_Max\_TON"]])

'''

#with open('benchmark\_results\_bofire.pkl', 'wb') as f:

#pickle.dump(results, f)

emulator = summit.get\_pretrained\_reizman\_suzuki\_emulator(case=1)

def run\_mobo\_optimization(emulator: summit.benchmarks.experimental\_emulator.ReizmanSuzukiEmulator, mobo\_strategy, bofire\_initial\_conditions, experimental\_budget=5):

"""

Runs a multi-objective Bayesian optimization loop.

Parameters:

mobo\_strategy: The optimization strategy object with ask and tell methods.

evaluate\_candidates: Function to evaluate candidate experiments.

bofire\_initial\_conditions: Initial conditions for optimization.

experimental\_budget: Number of iterations for the optimization loop.

Returns:

results\_df: DataFrame containing experiment results and cumulative metrics.

results: Final experiments stored in the strategy object.

"""

# We wish the temperature of the reaction to be between 30 and 110 °C

temperature\_feature = ContinuousInput(

key="Temperature", bounds=[30.0, 110.0], unit="°C"

)

# Catalyst Loading

catalyst\_loading\_feature = ContinuousInput(

key="Catalyst Loading", bounds=[0.5, 2], unit="%"

)

# Residence Time

residence\_time\_feature = ContinuousInput(

key="Residence Time", bounds=[1 \* 60, 10 \* 60], unit="minutes"

)

# Catalyst choice

catalyst\_feature = CategoricalInput(

key="Catalyst",

categories=[

"P1-L1",

"P2-L1",

"P1-L2",

"P1-L3",

"P1-L4",

"P1-L5",

"P1-L6",

"P1-L7",

],

)

# gather all individual features

input\_features = Inputs(

features=[

temperature\_feature,

catalyst\_loading\_feature,

residence\_time\_feature,

catalyst\_feature,

]

)

name\_map = {

"Catalyst Loading": "catalyst\_loading",

"Residence Time": "t\_res",

"Temperature": "temperature",

"Catalyst": "catalyst",

"Yield": "yld",

}

candidates = pd.DataFrame(

{

"Catalyst Loading": [0.498],

"Residence Time": [600],

"Temperature": [30],

"Catalyst": ["P1-L3"],

}

).rename(columns=name\_map)

#print(candidates)

#emulator = summit.get\_pretrained\_reizman\_suzuki\_emulator(case=1)

conditions = summit.DataSet.from\_df(candidates)

results = emulator.run\_experiments(conditions, rtn\_std=True).rename(

columns=dict(zip(name\_map.values(), name\_map.keys())),

)

experiments = pd.DataFrame(

{

"Catalyst Loading": results["Catalyst Loading"],

"Residence Time": results["Residence Time"],

"Temperature": results["Temperature"],

"Catalyst": results["Catalyst"],

"Yield": results["Yield"],

"valid\_Yield": 1,

}

)

max\_objective = MaximizeObjective(w=1.0)

#min\_objective = MinimizeObjective(w=1.0, bounds=[0, 200])

max\_objective\_2 = MaximizeObjective(w=1.0, bounds=[0, 200])

yield\_feature = ContinuousOutput(key="Yield", objective=max\_objective)

ton\_feature = ContinuousOutput(key="TON", objective=max\_objective\_2)

# create an output feature

output\_features = Outputs(features=[yield\_feature, ton\_feature])

domain = Domain(

inputs=input\_features,

outputs=output\_features,

)

# a multi objective BO strategy

#qExpectedImprovement = qEHVI()

qLogExpectedImprovement = qLogEHVI()

mobo\_strategy\_data\_model = MoboStrategy(

domain=domain,

acquisition\_function=qLogExpectedImprovement,

)

# map the strategy data model to the actual strategy that has functionality

mobo\_strategy = strategies.map(mobo\_strategy\_data\_model)

# Initialize variables

candidates = bofire\_initial\_conditions

print('initial conditions', candidates)

experiments = evaluate\_candidates(candidates)

mobo\_strategy.tell(experiments, replace=True, retrain=True)

results\_df = pd.DataFrame(columns=["Iteration","Catalyst Loading", "Residence Time", "Temperature",

"Catalyst", "Yield", "valid\_Yield", "TON", "valid\_TON"])

times\_df\_bofire = pd.DataFrame(columns=["Iteration", "Time\_taken"])

i = 0

done = False

while not done:

print('Starting Bofire loop')

i += 1

t1 = time.time()

# Ask for a new experiment

new\_candidate = mobo\_strategy.ask(1)

new\_experiment = evaluate\_candidates(new\_candidate)

# Inform the strategy about the new experiment

mobo\_strategy.tell(new\_experiment)

# Add new results to DataFrame

results\_df = pd.concat([results\_df, new\_experiment], ignore\_index=True)

# Calculate cumulative max for Yield and TON

results\_df["Cumulative\_Max\_Yield"] = results\_df["Yield"].cummax()

results\_df["Cumulative\_Max\_TON"] = results\_df["TON"].cummax()

results\_df.loc[results\_df["Iteration"].isna(), "Iteration"] = i

time\_taken = time.time() - t1

print(f"Iteration {i} took {time\_taken:.2f} seconds")

times\_df\_bofire = pd.concat(

[times\_df\_bofire, pd.DataFrame({"Iteration": [i], "Time\_taken": [time\_taken]})],

ignore\_index=True

)

print("Current Results:")

print(results\_df[["Yield", "Cumulative\_Max\_Yield", "TON", "Cumulative\_Max\_TON"]])

if i > experimental\_budget:

done = True

results = mobo\_strategy.experiments

print("Final experiments:")

print(results)

print("Results including cumulative max metrics:")

print(results\_df[["Yield", "Cumulative\_Max\_Yield", "TON", "Cumulative\_Max\_TON"]])

return results\_df, results, times\_df\_bofire