**数据集**：dataset-30

API\_NUM=900  
SATELLITE\_NUM=30  
SERVICE\_NUM=20000  
PATH="dataset\_30/"

**API属性**：api\_info.csv

data = np.zeros((API\_NUM, 3))  
for i in range(data.shape[0]):  
 data[i][0] = random.uniform(4.5,75)  
 data[i][1] = random.uniform(0.1,60)  
 data[i][2] = random.uniform(5,7)  
api\_info = pd.DataFrame(data)  
#columns=['cpu frequency','memory','delay\_constraint']  
api\_info.to\_csv(PATH+'api\_info.csv', index=False,header=False)

CPU [4.5,70]

Memory [0.1,60]

Delay [5,7]

**卫星限制**：satellite\_info.pickle

for env in range (SERVICE\_NUM):  
 data = np.zeros((SATELLITE\_NUM, 5))  
 for i in range(data.shape[0]):  
 data[i][0] = random.randint(75,200)  
 data[i][1] = random.uniform(40,200)  
 data[i][2] = random.uniform(2,6)  
 data[i][3] = random.uniform(50,100)  
 data[i][4] = random.uniform(20,90)  
 satellite[env]=data

CPU:[75,200]

Memory:[40,200]

Delay:[2,6]

Energy:[50,100]

Temp:[20,90]

**卫星能耗：satellite\_api\_cost**

Satellite energy consumption: satellite\_api\_cost

Restrictions:

1. The service CPU is smaller than the satellite CPU

2. Service Memory is smaller than satellite Memory

3. Service Delay is greater than satellite Delay

4. Service power consumption is less than satellite power\*20%

5. The satellite temperature is less than 80 degrees

optimize the target:

Minimal total energy consumption

import os

import pickle

from functools import partial

from turtle import left

import numpy as np

import pandas as pd

import torch

from torch.utils.data import DataLoader, Dataset

from tqdm import tqdm

import pickle

from torch.nn.utils.rnn import pad\_sequence

class Input(object):

def \_\_init\_\_(self, \*\*kwgs):

for key, value in kwgs.items():

value = torch.stack(value, dim=0)

setattr(self, key, value)

def cuda(self):

for key in self.\_\_dict\_\_:

setattr(self, key, getattr(self, key).cuda())

return self

class TheDataLoader(object):

def \_\_init\_\_(self, dataset='random', root='.', train\_batch\_size=128, eval\_batch\_size=1024,

num\_workers=8, negative\_sample=-1, candidate\_sample=False):

self.dataset = dataset

self.root = root

self.loadpath = os.path.join(self.root, self.dataset)

self.train\_batch\_size = train\_batch\_size

self.eval\_batch\_size = eval\_batch\_size

self.num\_workers = num\_workers

self.\_build\_dataset()

def \_build\_dataset(self):

# with open(os.path.join(self.loadpath, f'service\_api.pkl'), 'rb') as f:

# self.info = pickle.load(f)

# self.n\_items = self.info['item\_num']

self.train\_data = TheDataset(dataset=self.dataset, root=self.root, file = '30\_train\_input.csv',

phase='train')

self.valid\_data = TheDataset(dataset=self.dataset, root=self.root, file = '30\_valid\_input.csv',

phase='valid')

self.test\_data = TheDataset(dataset=self.dataset, root=self.root, file = '30\_test\_input.csv',

phase='test')

def \_collate\_fn(self, data):

combination = [tup[0] for tup in data]

pad = pad\_sequence(combination ,batch\_first=True, padding\_value = 0)

opt\_comb = [tup[4] for tup in data]

comb\_pad = pad\_sequence(opt\_comb, batch\_first=True, padding\_value = 0)

opt\_comb\_loc = [tup[5] for tup in data]

comb\_loc\_pad = pad\_sequence(opt\_comb\_loc, batch\_first=True, padding\_value = 0)

# 'api\_count': torch.tensor(len(pad[0])),

feed\_dict = {

'api\_combination': [torch.as\_tensor(i) for i in pad],

'api\_count': [tup[1] for tup in data],

'sat\_fea\_info':[tup[2] for tup in data],

'label':[tup[3] for tup in data],

'opt\_sat\_comb':[torch.as\_tensor(i) for i in comb\_pad],

'opt\_sat\_comb\_loc':[torch.as\_tensor(i) for i in comb\_loc\_pad]

}

return Input(\*\*feed\_dict)

def train\_dataloader(self):

return DataLoader(dataset=self.train\_data,

batch\_size=self.train\_batch\_size,

num\_workers=self.num\_workers,

shuffle=True,

collate\_fn=partial(self.\_collate\_fn))

def valid\_dataloader(self):

return DataLoader(dataset=self.valid\_data,

batch\_size=self.eval\_batch\_size,

num\_workers=self.num\_workers,

shuffle=False,

collate\_fn=partial(self.\_collate\_fn))

def test\_dataloader(self):

return DataLoader(dataset=self.test\_data,

batch\_size=self.eval\_batch\_size,

num\_workers=self.num\_workers,

shuffle=False,

collate\_fn=partial(self.\_collate\_fn))

class TheDataset(Dataset):

def \_\_init\_\_(self, dataset='random', root='.', file = '30\_valid\_input.csv',phase='train') -> None:

self.loadpath = os.path.join(root, dataset)

self.dataset = dataset

self.phase = phase

self.file = file

self.\_load\_data()

def \_load\_data(self):

# self.user\_ids = []

# self.api\_combination = []

# self.api\_count = []

# self.sat\_fea\_info = []

# self.label = []

# sat\_sta = pd.read\_pickle(os.path.join(self.loadpath, f'satellite\_info.pickle'))

data = pd.read\_csv(os.path.join(self.loadpath, self.file))

self.api\_combination = [torch.tensor(eval(i)) for i in data['api\_combination'].tolist()][:1]

self.api\_count = [torch.tensor(len(eval(i))) for i in data['api\_combination'].tolist()][:1]

self.sat\_fea\_info = [torch.tensor(eval(i)) for i in data['sat\_fea\_info'].tolist()][:1]

self.label = [torch.tensor(i) for i in data['label'].tolist()][:1]

self.opt\_sat\_comb = [torch.tensor(eval(i)) for i in data['opt\_sat\_comb'].tolist()][:1]

self.opt\_sat\_comb\_loc = [torch.tensor(eval(i)) for i in data['opt\_sat\_comb\_loc'].tolist()][:3]

def \_\_len\_\_(self):

return len(self.api\_combination)

def \_\_getitem\_\_(self, idx):

if self.phase in ['valid', 'test']:

return self.api\_combination[idx], self.api\_count[idx], self.sat\_fea\_info[idx], self.label[idx], self.opt\_sat\_comb[idx], self.opt\_sat\_comb\_loc[idx]

else:

return self.api\_combination[idx], self.api\_count[idx], self.sat\_fea\_info[idx], self.label[idx], self.opt\_sat\_comb[idx], self.opt\_sat\_comb\_loc[idx]

# if \_\_name\_\_ == "\_\_main\_\_":

# loader = TheDataLoader('random', '../dataset', negative\_sample=10)

# for seq\_input in tqdm(loader.train\_dataloader()):

# seq\_input = seq\_input.cuda()

# print(seq\_input.\_\_dict\_\_)

# break

# for seq\_input in tqdm(loader.valid\_dataloader()):

# seq\_input = seq\_input.cuda()

# print(seq\_input.\_\_dict\_\_)

# break

# for seq\_input in tqdm(loader.test\_dataloader()):

# seq\_input = seq\_input.cuda()

# print(seq\_input.\_\_dict\_\_)

# break