import collections

import os

from pickle import TRUE

import pickle

import numpy as np

from tqdm import trange

import yaml

import json

import re

import copy

import matplotlib.pyplot as plt

plt.rcParams.update({'font.size': 18})

from archai.nlp.nas.nas\_utils.plotter import plot\_2d\_pareto, plot\_3d\_pareto

def meta\_constructor\_mapping(loader, node):

value = loader.construct\_mapping(node)

return value

def meta\_constructor\_sequence(loader, node):

value = loader.construct\_sequence(node)

return value

yaml.add\_constructor(u'tag:yaml.org,2002:python/object/apply:numpy.core.multiarray.scalar', meta\_constructor\_sequence)

yaml.add\_constructor(u'tag:yaml.org,2002:python/object/apply:numpy.dtype', meta\_constructor\_mapping)

def get\_indices\_to\_keep(x\_pareto, y\_pareto, threshold=0.005):

# x\_pareto lower is better

# y\_pareto lower is better

indices\_to\_keep = []

for i, (x1, y1) in enumerate(zip(x\_pareto, y\_pareto)):

is\_pareto = True

for j,(x2, y2) in enumerate(zip(x\_pareto, y\_pareto)):

if i==j:

continue

if y2 < y1 and x2 <= x1:

if (y1-y2) > threshold \* y2:

is\_pareto = False

break

if is\_pareto:

indices\_to\_keep.append(i)

return indices\_to\_keep

def plot\_paper(x\_pareto, y\_pareto, x\_baseline, y\_baseline, x\_label, y\_label, path\_to\_save, indices\_to\_keep, scale\_x=1):

x\_pareto, y\_pareto = x\_pareto[indices\_to\_keep], y\_pareto[indices\_to\_keep]

indices = np.argsort(x\_pareto)

x\_pareto, y\_pareto = x\_pareto[indices], y\_pareto[indices]

indices = np.argsort(x\_baseline)

x\_baseline, y\_baseline = x\_baseline[indices], y\_baseline[indices]

plt.figure(figsize=(5,3))

plt.plot(np.asarray(x\_pareto) \* scale\_x, y\_pareto, markersize=10, label='LTS', color='midnightblue', marker='.')

plt.plot(np.asarray(x\_baseline) \* scale\_x, y\_baseline, markersize=5, label='Scaled Transformer', color='tab:blue', marker='d')

# plt.xlim((min(np.min(x\_pareto), np.min(x\_baseline))\*scale\_x-10, np.max(gt\_latencies)\*1000+10))

plt.xlabel(x\_label)

plt.ylabel(y\_label)

plt.grid(axis='y')

plt.legend(handletextpad=0.1, borderpad=0)

plt.savefig(path\_to\_save, bbox\_inches="tight")

print('baseline points:', [(v, l) for v, l in zip(y\_baseline, x\_baseline)])

print('pareto points:', [(v, l) for v, l in zip(y\_pareto, x\_pareto)])

diff = []

for (x, y) in zip(x\_pareto, y\_pareto):

# find closest baseline point

idx = np.argmin(np.absolute(x\_baseline - x))

if abs(x\_baseline[idx] - x) < 0.05\*x:

diff.append((y\_baseline[idx] - y)\*100./y)

print('on {}/{} baseline models, for the same metric we get {}%% lower ppl on average.'.format(len(diff), len(x\_pareto), np.mean(diff)))

def get\_config\_name(job):

# idx = re.search('(config\_[0-9]+)', job).span()[0]

# job = job[idx:]

# config\_name = job.split('/')[0]

# return config\_name + '\_' + job.split('/')[1]

try:

idx = re.search('(config\_[0-9]+)', job).span()[0]

job = job[idx:]

config\_name = job.split('/')[0]

return config\_name + '\_' + job.split('/')[1]

except:

config\_name = re.search('(M[0-9]+)', job).group(1)

return config\_name

def get\_info\_from\_json(json\_file, metric=['valid\_perplexity', 'valid\_ppl']):

'''

step: step number to extract the ppl log, live empty to get the final ppl

metric: type of metric to read from the json file

'''

out\_dict = {}

with open(json\_file, 'r', encoding='utf-8') as f:

lines = f.readlines()[::-1]

try:

job\_desc = re.search('DLLL \{(.+?)\n', lines[-1])

except:

return None

job\_desc = '{'+job\_desc.group(1)

work\_dir = json.loads(job\_desc)['data']['work\_dir']

try:

idx\_start = re.search('amlt-results', work\_dir).span()[-1] + 1

amlt\_job = work\_dir[idx\_start:].split('/')[0]

except:

amlt\_job = None

for line in lines:

str = re.search('DLLL \{(.+?)\}', line)

str = '{'+str.group(1)+'}}'

final\_train\_log = json.loads(str)

try:

out\_dict['train\_elapsed'] = float(final\_train\_log['data']['train\_elapsed'])\*60

for k in final\_train\_log['data'].keys():

if k in metric:

out\_dict[k] = final\_train\_log['data'][k]

out\_dict['amlt\_job'] = amlt\_job

break

except:

return None

return out\_dict

def recurse\_dir(path\_to\_dir, fname='config.yaml'):

results = {}

for j in os.listdir(path\_to\_dir):

j\_path = os.path.join(path\_to\_dir, j)

if os.path.isdir(j\_path):

results.update(recurse\_dir(j\_path, fname=fname))

else:

config = None

if os.path.basename(j\_path) == fname:

if '.yaml' in fname:

with open(os.path.join(j\_path), 'r') as f:

config = yaml.safe\_load(f)

elif '.json' in fname:

config = get\_info\_from\_json(os.path.join(j\_path))

else:

raise NotImplementedError

if config is None and '.yaml' in fname:

try:

json\_file = os.path.join(path\_to\_dir, 'train\_log.json')

with open(json\_file, 'r', encoding='utf-8') as f:

lines = f.readlines()

try:

job\_desc = re.search('DLLL \{(.+?)\}', lines[0])

except:

return None

job\_desc = '{'+job\_desc.group(1)+'}}'

config = json.loads(job\_desc)['data']

except:

print(f'##### no logfile found for {j\_path}')

if config is not None:

config\_name = get\_config\_name(j\_path)

results[config\_name] = config

return results

def config\_to\_key(config, name=None, keys=['n\_layer', 'd\_model', 'd\_inner','n\_head', 'div\_val']):

short\_config = {}

for k in keys:

if isinstance(config[k], list) and len(config[k])==1:

short\_config[k] = config[k][0]

else:

short\_config[k] = config[k]

if name is not None:

short\_config['name'] = name

return short\_config

def profile\_baseline(evolution\_obj, path\_to\_results, device\_name, dataset):

training\_path = os.path.join(path\_to\_results, dataset)

path\_to\_configs = os.path.join(training\_path, 'model\_configs.yaml')

if os.path.exists(path\_to\_configs):

with open(path\_to\_configs, 'r') as f:

configs = yaml.safe\_load(f)

else:

fname = 'config.yaml' if evolution\_obj.model\_type == 'mem\_transformer' else 'model\_config.yaml'

configs = recurse\_dir(training\_path, fname=fname)

with open(path\_to\_configs, 'w') as f:

yaml.dump(configs, f)

proxies = {}

total\_params = {}

latencies = {}

memories = {}

logs = {'configs': [],

'proxies': [],

'total\_params': [],

'latencies': [],

'memories': []}

for job\_name, model\_config in configs.items():

gene = evolution\_obj.converter.config\_to\_gene(model\_config)

config\_, proxy, total\_param, latency, memory = evolution\_obj.\_calculate\_gene\_constraints(gene)

print(f'''{job\_name} has:

{proxy} {evolution\_obj.constraint\_strategy}

{total\_param} total\_params

{latency:.4f}s latency

{memory:.4f}MB memory''')

proxies[job\_name] = proxy

total\_params[job\_name] = total\_param

latencies[job\_name] = latency

memories[job\_name] = memory

logs['configs'].append(config\_)

logs['proxies'].append(proxy)

logs['total\_params'].append(total\_param)

logs['latencies'].append(latency)

logs['memories'].append(memory)

logs\_path = os.path.join(path\_to\_results, device\_name)

with open(os.path.join(logs\_path, 'logs.pkl'), 'wb') as f:

pickle.dump({'configs': logs['configs'],

'proxies': logs['proxies'],

'total\_params': logs['total\_params'],

'latencies': logs['latencies'],

'memories': logs['memories']}, f)

yaml\_file = os.path.join(logs\_path, 'proxies\_summary.yaml')

with open(yaml\_file, 'w') as f:

yaml.dump(proxies, f)

yaml\_file = os.path.join(logs\_path, 'total\_params\_summary.yaml')

with open(yaml\_file, 'w') as f:

yaml.dump(total\_params, f)

yaml\_file = os.path.join(logs\_path, 'latencies\_summary.yaml')

with open(yaml\_file, 'w') as f:

yaml.dump(latencies, f)

yaml\_file = os.path.join(logs\_path, 'memories\_summary.yaml')

with open(yaml\_file, 'w') as f:

yaml.dump(memories, f)

return logs

def select\_pareto(evolution\_obj, path\_to\_results):

with open(os.path.join(path\_to\_results, 'logs.pkl'), 'rb') as f:

baseline = pickle.load(f)

indices = set()

for p\_b, l\_b, m\_b in zip(baseline['proxies'], baseline['latencies'], baseline['memories']):

if evolution\_obj.constraint\_strategy == 'decoder\_params':

candidate\_proxy = 0

else:

raise NotImplementedError

candidate\_latency = np.Inf

candidate\_memory = np.Inf

index\_proxy = None

index\_latency = None

index\_memory = None

for i, (p, l, m) in enumerate(zip(evolution\_obj.pareto['proxies'], evolution\_obj.pareto['latencies'], evolution\_obj.pareto['memories'])):

if l == np.min(evolution\_obj.pareto['latencies']) or m == np.min(evolution\_obj.pareto['memories']):

indices.add(i)

if abs(p-p\_b) < 0.05\*p\_b:

if l < l\_b and l < candidate\_latency:

index\_latency = i

candidate\_latency = l

if m < m\_b and m < candidate\_memory:

index\_memory = i

candidate\_memory = m

if abs(l-l\_b) < 0.05\*l\_b:

if p > p\_b and p > candidate\_proxy:

index\_proxy = i

candidate\_proxy = p

if m < m\_b and m < candidate\_memory:

index\_memory = i

candidate\_memory = m

if abs(m-m\_b) < 0.05\*m\_b:

if p > p\_b and p > candidate\_proxy:

index\_proxy = i

candidate\_proxy = p

if l < l\_b and l < candidate\_latency:

index\_latency = i

candidate\_latency = l

if index\_proxy is not None:

indices.add(index\_proxy)

if index\_latency is not None:

indices.add(index\_latency)

if index\_memory is not None:

indices.add(index\_memory)

indices = list(indices)

print(f'Selected {len(indices)} of pareto jobs based on their proximity to the baseline architectures.')

evolution\_obj.pareto = {'population': [evolution\_obj.pareto['population'][i] for i in indices],

'proxies': [evolution\_obj.pareto['proxies'][i] for i in indices],

'total\_params': [evolution\_obj.pareto['total\_params'][i] for i in indices],

'latencies': [evolution\_obj.pareto['latencies'][i] for i in indices],

'memories': [evolution\_obj.pareto['memories'][i] for i in indices]}

evolution\_obj.plot\_search\_state(iteration=None, parents=None, baseline=baseline)

model\_configs = []

for gene in evolution\_obj.pareto['population']:

config = evolution\_obj.converter.gene\_to\_config(gene)

curr\_model\_config = copy.deepcopy(evolution\_obj.model\_config)

curr\_model\_config.update(config)

curr\_model\_config = curr\_model\_config.to\_dict()

model\_configs.append(curr\_model\_config)

evolution\_obj.pareto['model\_configs'] = model\_configs

baseline['pareto'] = [evolution\_obj.pareto]

with open(os.path.join(path\_to\_results, 'logs.pkl'), 'wb') as f:

pickle.dump(baseline, f)

return evolution\_obj

def plot\_baseline\_and\_pareto(path\_to\_amlt\_logs, path\_to\_baseline\_logs, dataset, device\_name):

# load all info for baseline models

baseline\_logs = recurse\_dir(os.path.join(path\_to\_baseline\_logs, dataset), fname='train\_log.json') # load baseline val\_ppls

# print(baseline\_logs)

baseline\_configs = recurse\_dir(os.path.join(path\_to\_baseline\_logs, dataset), fname='model\_config.yaml') # load baseline model configs

# print(baseline\_configs)

with open(os.path.join(os.path.join(path\_to\_baseline\_logs, device\_name), 'latencies\_summary.yaml'), 'r') as f: # load baseline latencies

baseline\_latencies = yaml.safe\_load(f)

# print(baseline\_latencies)

with open(os.path.join(os.path.join(path\_to\_baseline\_logs, device\_name), 'memories\_summary.yaml'), 'r') as f: # load baseline memories

baseline\_memories = yaml.safe\_load(f)

# print(baseline\_memories)

with open(os.path.join(os.path.join(path\_to\_baseline\_logs, device\_name), 'proxies\_summary.yaml'), 'r') as f: # load baseline memories

baseline\_params = yaml.safe\_load(f)

# print(baseline\_params)

# load all info for (selected) pareto models

pareto\_train\_logs = recurse\_dir(path\_to\_amlt\_logs, fname='train\_log.json') # load pareto val\_ppls

print(pareto\_train\_logs)

pareto\_configs = recurse\_dir(path\_to\_amlt\_logs, fname='model\_config.yaml') # load pareto model configs

print(pareto\_configs)

fname = 'logs.pkl'

with open(os.path.join(path\_to\_baseline\_logs, device\_name, fname), 'rb') as f: # load pareto memories and latencies

pareto\_logs = pickle.load(f)['pareto'][0]

print("pareto\_logs",pareto\_logs)

# with open('logdir/mem\_transformer\_titanxp\_3d/logs\_iter\_29.pkl', 'rb') as f:

# pareto\_ = pickle.load(f)

# match\_indices = collections.OrderedDict({})

# for config\_idx in range(4):

# for job\_idx in range(10):

# if f'config\_{config\_idx}\_j0' not in pareto\_train\_logs.keys():

# continue

# for idx in range(len(pareto\_logs['model\_configs'])):

# match = True

# this\_config = config\_to\_key(pareto\_configs[f'config\_{config\_idx}\_j{job\_idx}'])

# orig\_config = config\_to\_key(pareto\_logs['model\_configs'][idx])

# for k, v in this\_config.items():

# if v != orig\_config[k]:

# match = False

# break

# if match:

# found = True

# break

# assert found

# print('idx {} matched to {}'.format(idx, f'config\_{config\_idx}\_j{job\_idx}'))

# match\_indices[f'config\_{config\_idx}\_j{job\_idx}'] = idx

# keys = list(pareto\_logs.keys())

# for k in keys:

# print('reordering ', k)

# pareto\_logs[k] = (np.asarray(pareto\_logs[k])[list(match\_indices.values())]).tolist()

# with open(os.path.join(path\_to\_baseline\_logs, device\_name, fname), 'rb') as f: # load pareto memories and latencies

# all\_logs = pickle.load(f)['pareto'][0]

# all\_logs['pareto'] = [pareto\_logs]

# with open(os.path.join(path\_to\_baseline\_logs, device\_name, fname), 'wb') as f: # load pareto memories and latencies

# pickle.dump(all\_logs, f)

all\_val\_ppls = []

all\_configs = []

all\_latencies = []

all\_memories = []

# config\_idx, job\_idx, idx = 0, 0, 0

n\_configs = 4

n\_jobs\_perconfig = np.ceil(len(pareto\_logs['model\_configs']) / n\_configs)

print(n\_jobs\_perconfig,"n\_jobs\_perconfig")

print(len(pareto\_logs['model\_configs']),"len(pareto\_logs['model\_configs'])")

for idx in range(len(pareto\_logs['model\_configs'])):

config\_idx = int(idx // n\_jobs\_perconfig)

job\_idx = int(idx % n\_jobs\_perconfig)

# if f'config\_{config\_idx}\_j0' not in pareto\_train\_logs.keys():

# break

# while True:

# if f'config\_{config\_idx}\_j{job\_idx}' not in pareto\_train\_logs.keys():

# break

print(pareto\_train\_logs.keys())

if f'config\_{config\_idx}\_j{job\_idx}\_train\_log.json' not in pareto\_train\_logs.keys():

continue

print(f'config\_{config\_idx}\_j{job\_idx}\_train\_log.json')

# make sure these are the same models

this\_config = config\_to\_key(pareto\_configs[f'config\_{config\_idx}\_j{job\_idx}\_model\_config.yaml'])

print(this\_config, "this\_config")

orig\_config = config\_to\_key(pareto\_logs['model\_configs'][idx])

print(orig\_config, "orig\_config")

for k, v in this\_config.items():

assert v == orig\_config[k], print(f'config\_{config\_idx}\_j{job\_idx}', k, v, orig\_config[k])

l, m = pareto\_logs['latencies'][idx], pareto\_logs['memories'][idx]

all\_val\_ppls.append(pareto\_train\_logs[f'config\_{config\_idx}\_j{job\_idx}']['valid\_ppl'])

all\_configs.append(config\_to\_key(pareto\_configs[f'config\_{config\_idx}\_j{job\_idx}'], name=f'config\_{config\_idx}\_j{job\_idx}'))

all\_latencies.append(l)

all\_memories.append(m)

all\_val\_ppls = np.asarray(all\_val\_ppls)

all\_configs = np.asarray(all\_configs)

all\_latencies = np.asarray(all\_latencies)

all\_memories = np.asarray(all\_memories)

job\_keys = np.sort(list(baseline\_memories.keys()))

print(job\_keys)

print(baseline\_logs['config\_0\_j0\_train\_log.json']['valid\_ppl'])

job\_keys\_modified = [k + "\_train\_log.json" for k in job\_keys]

try:

baseline\_val\_ppls = np.asarray([baseline\_logs[k]['valid\_perplexity'] for k in job\_keys\_modified])

except:

baseline\_val\_ppls = np.asarray([baseline\_logs[k]['valid\_ppl'] for k in job\_keys\_modified])

job\_keys\_modified2 = [k + "\_model\_config.yaml" for k in job\_keys]

print(baseline\_configs['config\_0\_j0\_model\_config.yaml'])

baseline\_configs = np.asarray([config\_to\_key(baseline\_configs[k], name=job\_keys[job\_keys\_modified2.index(k)]) for k in job\_keys\_modified2])

print(baseline\_configs)

baseline\_latencies = np.asarray([baseline\_latencies[k] for k in job\_keys])

baseline\_memories = np.asarray([baseline\_memories[k] for k in job\_keys])

indices\_latency = get\_indices\_to\_keep(x\_pareto=all\_latencies, y\_pareto=all\_val\_ppls)

print(indices\_latency,"latency")

indices\_memory = get\_indices\_to\_keep(x\_pareto=all\_memories, y\_pareto=all\_val\_ppls)

indices\_to\_keep = np.union1d(indices\_latency, indices\_memory)

print(f'keeping {len(indices\_to\_keep)} pareto models')

with open(os.path.join(path\_to\_baseline\_logs, device\_name, f'indices\_to\_keep\_{dataset}.pkl'), 'wb') as f:

pickle.dump(indices\_to\_keep, f)

lat\_diff = []

mem\_diff = []

for l, m , ppl in zip(baseline\_latencies, baseline\_memories, baseline\_val\_ppls):

idx = np.argmin(np.absolute(all\_val\_ppls - ppl))

if idx in indices\_to\_keep and np.absolute(all\_val\_ppls[idx] - ppl) < 0.1 \* ppl:

lat\_diff.append((l - all\_latencies[idx])\*100./l)

mem\_diff.append((m - all\_memories[idx])\*100./m)

print('on {}/{} baseline models, for the same ppl we get less {}%% latency and {}%% memory on average.'.format(len(baseline\_val\_ppls), len(lat\_diff), np.mean(lat\_diff), np.mean(mem\_diff)))

# 2D plot: val\_ppl x latencies

visited\_dict = {'x': all\_val\_ppls, 'y': all\_latencies, 'config': all\_configs}

pareto\_dict = visited\_dict

baseline\_dict = {'x': baseline\_val\_ppls, 'y': baseline\_latencies, 'config': baseline\_configs}

output\_path = os.path.join(path\_to\_amlt\_logs, f'val\_ppl\_vs\_latency')

plot\_2d\_pareto(visited\_dict,

pareto\_dict,

parents=None,

baseline=baseline\_dict,

hover\_template='Val ppl: %{x:.2f}' + '<br>Latency (s): %{y:.4f}<br>' + '%{text}',

title\_text=f'Val ppl vs. Latency (s)',

xaxis\_title='Val ppl',

yaxis\_title='Latency (s)',

output\_path=output\_path)

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* latency \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

plot\_paper(x\_pareto=all\_latencies, y\_pareto=all\_val\_ppls, x\_baseline=baseline\_latencies, y\_baseline=baseline\_val\_ppls,

x\_label='Latency (ms)', y\_label='Validation PPL', path\_to\_save=output\_path, scale\_x=1000., indices\_to\_keep=indices\_latency)

# 2D plot: val\_ppl x memories

visited\_dict = {'x': all\_val\_ppls, 'y': all\_memories, 'config': all\_configs}

pareto\_dict = visited\_dict

baseline\_dict = {'x': baseline\_val\_ppls, 'y': baseline\_memories, 'config': baseline\_configs}

output\_path = os.path.join(path\_to\_amlt\_logs, f'val\_ppl\_vs\_memory')

plot\_2d\_pareto(visited\_dict,

pareto\_dict,

parents=None,

baseline=baseline\_dict,

hover\_template='Val ppl: %{x:.2f}' + '<br>Memory (MB): %{y:.4f}<br>' + '%{text}',

title\_text=f'Val ppl vs. Memory (MB)',

xaxis\_title='Val ppl',

yaxis\_title='Memory (MB)',

output\_path=output\_path)

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* memory \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

plot\_paper(x\_pareto=all\_memories, y\_pareto=all\_val\_ppls, x\_baseline=baseline\_memories, y\_baseline=baseline\_val\_ppls,

x\_label='Memory (MB)', y\_label='Validation PPL', path\_to\_save=output\_path, scale\_x=1., indices\_to\_keep=indices\_memory)

# 3D plot: val\_ppl x latencies x memories

visited\_dict = {'x': all\_val\_ppls, 'y': all\_memories, 'z': all\_latencies, 'config': all\_configs}

pareto\_dict = visited\_dict

baseline\_dict = {'x': baseline\_val\_ppls, 'y': baseline\_memories, 'z': baseline\_latencies, 'config': baseline\_configs}

output\_path = os.path.join(path\_to\_amlt\_logs, f'val\_ppl\_vs\_memory\_vs\_latency')

plot\_3d\_pareto(visited\_dict,

pareto\_dict,

parents=None,

baseline=baseline\_dict,

hover\_template='Val ppl: %{x:.2f}' + '<br>Memory (MB): %{y:.4f}<br>' + 'Latency (s): %{z:.4f}<br>' + '%{text}',

title\_text=f'Val ppl vs. Memory (MB) vs. Latency (s)',

xaxis\_title='Val ppl',

yaxis\_title='Memory (MB)',

zaxis\_title='Latency (s)',

output\_path=output\_path)

def get\_latex\_tables(path\_to\_baseline\_logs, dataset, device\_name):

fname = 'logs.pkl'

with open(os.path.join(path\_to\_baseline\_logs, device\_name, fname), 'rb') as f: # load pareto memories and latencies

pareto\_logs = pickle.load(f)['pareto'][0]

# pareto\_logs = {}

# model\_configs = recurse\_dir(f'archai/nlp/nas/saved\_logs/pareto\_TransXL\_lm1b\_{device\_name}', fname='model\_config.yaml')

# keys = list(model\_configs.keys())

# pareto\_logs['model\_configs'] = [model\_configs[k] for k in keys]

# with open(os.path.join(f'archai/nlp/nas/saved\_logs/pareto\_TransXL\_lm1b\_{device\_name}', 'params\_summary.yaml'), 'r') as f:

# params\_dict = yaml.safe\_load(f)

# pareto\_logs['proxies'] = [params\_dict[k]['nonembedding'] for k in keys]

filename = 'table\_{}.tex'.format(device\_name)

with open(filename, 'w') as f:

f.write('\\begin{table\*}[h]\n'+

'\\resizebox{\\textwidth}{!}{\n'+

'\\begin{tabular}{lcccccc} \hline\n'+

'\multicolumn{2}{c}{}\n'+

'& n\\textsubscript{layer} \n'+

'& d\\textsubscript{model} \n'+

'& n\\textsubscript{head} \n'+

'& d\\textsubscript{inner}\n'+

'& DecoderParams (M) \\\\ \\hline \n'+

'\\multicolumn{2}{l}{baseline} & $\in$[1,16] & 8 & 2048 & 512 & - \\\\ \\hline\n')

f.write('\multirow{%s}{\*}{\\rotatebox{90}{%s}}' % (len(pareto\_logs['model\_configs']), device\_name))

# with open(os.path.join(path\_to\_baseline\_logs, device\_name, f'indices\_to\_keep\_{dataset}.pkl'), 'rb') as f:

# indices\_to\_keep = pickle.load(f)

indices\_to\_keep = range(len(pareto\_logs['model\_configs']))

table\_configs = []

params = []

for idx in range(len(pareto\_logs['model\_configs'])):

if not idx in indices\_to\_keep:

continue

this\_config = config\_to\_key(pareto\_logs['model\_configs'][idx])

nonemb\_params = pareto\_logs['proxies'][idx]

table\_configs.append('&{} &{} &{} &{} &{:.1f} \\\\ \n'.format(this\_config['n\_layer'],

this\_config['d\_model'], this\_config['n\_head'], this\_config['d\_inner'],

nonemb\_params/1e6))

params.append(nonemb\_params)

indices = np.argsort(params)

for i, idx in enumerate(indices):

with open(filename, 'a+') as f:

f.write('&M{} '.format(i+1) + table\_configs[idx])

with open(filename, 'a+') as f:

f.write('\\hline\n' + '\\end{tabular}}\n' + '\\end{table\*}')

exit()