

# import argparse import itertools import json import os import random import re import time from functools import partial import torch from internvl.model import load\_model\_and\_tokenizer from internvl.train.dataset import build\_transform, dynamic\_preprocess from PIL import Image from torchvision.ops.boxes import box\_area from tqdm import tqdm ds\_collections = { 'refcoco\_val': 'data/refcoco/refcoco\_val.jsonl', 'refcoco\_testA': 'data/refcoco/refcoco\_testA.jsonl', 'refcoco\_testB': 'data/refcoco/refcoco\_testB.jsonl', 'refcoco+\_val': 'data/refcoco/refcoco+\_val.jsonl', 'refcoco+\_testA': 'data/refcoco/refcoco+\_testA.jsonl', 'refcoco+\_testB': 'data/refcoco/refcoco+\_testB.jsonl', 'refcocog\_val': 'data/refcoco/refcocog\_val.jsonl', 'refcocog\_test': 'data/refcoco/refcocog\_test.jsonl', } def box\_iou(boxes1, boxes2): area1 = box\_area(boxes1) area2 = box\_area(boxes2) lt = torch.max(boxes1[:, None, :2], boxes2[:, :2]) # [N,M,2] rb = torch.min(boxes1[:, None, 2:], boxes2[:, 2:]) # [N,M,2] wh = (rb - lt).clamp(min=0) # [N,M,2] inter = wh[:, :, 0] \* wh[:, :, 1] # [N,M] union = area1[:, None] + area2 - inter iou = inter / union return iou, union def collate\_fn(batches, tokenizer): pixel\_values = torch.cat([\_['pixel\_values'] for \_ in batches], dim=0) texts = [\_['text'] for \_ in batches] bboxes = [\_['bbox'] for \_ in batches] hws = [\_['hw'] for \_ in batches] return pixel\_values, texts, bboxes, hws class RefCOCODataset(torch.utils.data.Dataset): def \_\_init\_\_(self, test, prompt, input\_size=224, dynamic\_image\_size=False, use\_thumbnail=False, max\_num=6): self.datas = open(test).readlines() self.prompt = prompt self.input\_size = input\_size self.dynamic\_image\_size = dynamic\_image\_size self.use\_thumbnail = use\_thumbnail self.max\_num = max\_num self.transform = build\_transform(is\_train=False, input\_size=input\_size) def \_\_len\_\_(self): return len(self.datas) def \_\_getitem\_\_(self, idx): data = json.loads(self.datas[idx].strip()) image = data['image'] text = data['sent'] bbox = data['bbox'] w, h = data['width'], data['height'] image = Image.open(image).convert('RGB') if self.dynamic\_image\_size: images = dynamic\_preprocess(image, image\_size=self.input\_size, use\_thumbnail=self.use\_thumbnail, max\_num=self.max\_num) else: images = [image] pixel\_values = [self.transform(image) for image in images] pixel\_values = torch.stack(pixel\_values) return { 'text': self.prompt.format(text), 'pixel\_values': pixel\_values, 'bbox': bbox, 'hw': (h, w), } class InferenceSampler(torch.utils.data.sampler.Sampler): def \_\_init\_\_(self, size): self.\_size = int(size) assert size > 0 self.\_rank = torch.distributed.get\_rank() self.\_world\_size = torch.distributed.get\_world\_size() self.\_local\_indices = self.\_get\_local\_indices(size, self.\_world\_size, self.\_rank) @staticmethod def \_get\_local\_indices(total\_size, world\_size, rank): shard\_size = total\_size // world\_size left = total\_size % world\_size shard\_sizes = [shard\_size + int(r < left) for r in range(world\_size)] begin = sum(shard\_sizes[:rank]) end = min(sum(shard\_sizes[:rank + 1]), total\_size) return range(begin, end) def \_\_iter\_\_(self): yield from self.\_local\_indices def \_\_len\_\_(self): return len(self.\_local\_indices) def evaluate\_chat\_model(): print('prompt:', prompt) random.seed(args.seed) summaries = [] for ds\_name in args.datasets: dataset = RefCOCODataset( test=ds\_collections[ds\_name], prompt=prompt, input\_size=image\_size, dynamic\_image\_size=args.dynamic, use\_thumbnail=use\_thumbnail, max\_num=args.max\_num ) dataloader = torch.utils.data.DataLoader( dataset=dataset, sampler=InferenceSampler(len(dataset)), batch\_size=args.batch\_size, num\_workers=args.num\_workers, pin\_memory=True, drop\_last=False, collate\_fn=partial(collate\_fn, tokenizer=tokenizer), ) outputs = [] for \_, (pixel\_values, questions, bboxes, hws) in enumerate(tqdm(dataloader)): pixel\_values = pixel\_values.to(torch.bfloat16).cuda() generation\_config = dict( num\_beams=args.num\_beams, max\_new\_tokens=100, min\_new\_tokens=1, do\_sample=True if args.temperature > 0 else False, temperature=args.temperature, ) pred = model.chat( tokenizer=tokenizer, pixel\_values=pixel\_values, question=questions[0], generation\_config=generation\_config, verbose=True ) answers = [pred] for bbox, hw, answer in zip(bboxes, hws, answers): outputs.append({ 'answer': answer, 'gt\_bbox': bbox, 'hw': hw, }) torch.distributed.barrier() world\_size = torch.distributed.get\_world\_size() merged\_outputs = [None for \_ in range(world\_size)] torch.distributed.all\_gather\_object(merged\_outputs, outputs) merged\_outputs = [\_ for \_ in itertools.chain.from\_iterable(merged\_outputs)] if torch.distributed.get\_rank() == 0: print(f'Evaluating {ds\_name} ...') time\_prefix = time.strftime('%y%m%d%H%M%S', time.localtime()) results\_file = f'{ds\_name}\_{time\_prefix}.json' results\_file = os.path.join(args.out\_dir, results\_file) json.dump(merged\_outputs, open(results\_file, 'w')) correct = total\_cnt = 0 for i, output in enumerate(merged\_outputs): predict\_bbox = re.findall(PATTERN, output['answer']) try: predict\_bbox = (float(predict\_bbox[0][0]), float(predict\_bbox[0][1]), float(predict\_bbox[0][2]), float(predict\_bbox[0][3])) except: predict\_bbox = (0., 0., 0., 0.) target\_bbox = torch.tensor(output['gt\_bbox'], dtype=torch.float32).view(-1, 4) predict\_bbox = torch.tensor(predict\_bbox, dtype=torch.float32).view(-1, 4) if predict\_bbox.sum() >= 4: predict\_bbox = predict\_bbox / 1000 predict\_bbox[:, 0::2] \*= output['hw'][1] predict\_bbox[:, 1::2] \*= output['hw'][0] iou, \_ = box\_iou(predict\_bbox, target\_bbox) iou = iou.item() total\_cnt += 1 if iou >= 0.5: correct += 1 print(f'Evaluating {ds\_name} ...') print(f'Precision @ 1: {correct / total\_cnt} \n') summaries.append([args.checkpoint, ds\_name, f'Precision @ 1: {correct / total\_cnt} \n']) torch.distributed.barrier() out\_path = '\_'.join(args.checkpoint.split('/')[-2:]) writer = open(os.path.join(args.out\_dir, f'{out\_path}.txt'), 'a') print(f"write results to file {os.path.join(args.out\_dir, f'{out\_path}.txt')}") for summary in summaries: print(summary) writer.write(f'{summary}\n') writer.close() if \_\_name\_\_ == '\_\_main\_\_': parser = argparse.ArgumentParser() parser.add\_argument('--checkpoint', type=str, default='') parser.add\_argument('--datasets', type=str, default='refcoco\_val,refcoco\_testA,refcoco\_testB,' 'refcoco+\_val,refcoco+\_testA,refcoco+\_testB,' 'refcocog\_val,refcocog\_test') parser.add\_argument('--batch-size', type=int, default=1) parser.add\_argument('--num-workers', type=int, default=1) parser.add\_argument('--num-beams', type=int, default=1) parser.add\_argument('--out-dir', type=str, default='results') parser.add\_argument('--sample', type=bool, default=False) parser.add\_argument('--temperature', type=float, default=0.0) parser.add\_argument('--seed', type=int, default=0) parser.add\_argument('--dynamic', action='store\_true') parser.add\_argument('--max-num', type=int, default=6) parser.add\_argument('--load-in-8bit', action='store\_true') parser.add\_argument('--load-in-4bit', action='store\_true') parser.add\_argument('--auto', action='store\_true') args = parser.parse\_args() if not os.path.exists(args.out\_dir): os.makedirs(args.out\_dir, exist\_ok=True) args.datasets = args.datasets.split(',') print('datasets:', args.datasets) assert args.batch\_size == 1, 'Only batch size 1 is supported' torch.distributed.init\_process\_group( backend='nccl', world\_size=int(os.getenv('WORLD\_SIZE', '1')), rank=int(os.getenv('RANK', '0')), ) torch.cuda.set\_device(int(os.getenv('LOCAL\_RANK', 0))) PATTERN = re.compile(r'\[\*\[(.\*?),(.\*?),(.\*?),(.\*?)\]\]\*') model, tokenizer = load\_model\_and\_tokenizer(args) image\_size = model.config.force\_image\_size or model.config.vision\_config.image\_size use\_thumbnail = model.config.use\_thumbnail prompt = 'Please provide the bounding box coordinate of the region this sentence describes: <ref>{}</ref>' total\_params = sum(p.numel() for p in model.parameters()) / 1e9 if total\_params > 20 or args.dynamic: args.num\_beams = 1 print(f'[test] total\_params: {total\_params}B, use num\_beams: {args.num\_beams}') else: print(f'[test] total\_params: {total\_params}B') print(f'[test] image\_size: {image\_size}') print(f'[test] template: {model.config.template}') print(f'[test] dynamic\_image\_size: {args.dynamic}') print(f'[test] use\_thumbnail: {use\_thumbnail}') print(f'[test] max\_num: {args.max\_num}') evaluate\_chat\_model()这是evaluate\_grounding.py,将其迁移到vlmevalkit

下面是将 evaluate\_grounding.py 脚本迁移到 VLMEvalKit 框架的完整实现。迁移后的代码遵循 VLMEvalKit 的结构和规范：

Pythonimport os  
import re  
import json  
import time  
import torch  
import argparse  
import itertools  
from functools import partial  
from tqdm import tqdm  
from PIL import Image  
from torchvision.ops.boxes import box\_area  
  
from vlmeval.smp import \*  
from vlmeval.evaluate import BaseEvaluator  
from vlmeval.utils import track\_progress\_rich  
from vlmeval.config import supported\_VLM  
  
os.environ['TF\_ENABLE\_ONEDNN\_OPTS'] = '0'  
  
# 正则表达式模式用于提取边界框  
PATTERN = re.compile(r'$$\*\[(.\*?),(.\*?),(.\*?),(.\*?)$$\]\*')  
  
class RefCOCODataset(Dataset):  
 """RefCOCO 数据集类"""  
   
 def \_\_init\_\_(self, test\_file, prompt, input\_size=224,   
 dynamic\_image\_size=False, use\_thumbnail=False, max\_num=6):  
 self.datas = open(test\_file).readlines()  
 self.prompt = prompt  
 self.input\_size = input\_size  
 self.dynamic\_image\_size = dynamic\_image\_size  
 self.use\_thumbnail = use\_thumbnail  
 self.max\_num = max\_num  
 self.transform = build\_transform(is\_train=False, input\_size=input\_size)  
  
 def \_\_len\_\_(self):  
 return len(self.datas)  
  
 def \_\_getitem\_\_(self, idx):  
 data = json.loads(self.datas[idx].strip())  
 image = data['image']  
 text = data['sent']  
 bbox = data['bbox']  
 w, h = data['width'], data['height']  
  
 image = Image.open(image).convert('RGB')  
   
 if self.dynamic\_image\_size:  
 images = dynamic\_preprocess(  
 image,   
 image\_size=self.input\_size,  
 use\_thumbnail=self.use\_thumbnail,  
 max\_num=self.max\_num  
 )  
 else:  
 images = [image]  
   
 pixel\_values = [self.transform(image) for image in images]  
 pixel\_values = torch.stack(pixel\_values)  
  
 return {  
 'text': self.prompt.format(text),  
 'pixel\_values': pixel\_values,  
 'bbox': bbox,  
 'hw': (h, w),  
 'image\_path': image,  
 'question\_id': idx  
 }  
  
def collate\_fn(batches, tokenizer):  
 """数据批处理函数"""  
 pixel\_values = torch.cat([\_['pixel\_values'] for \_ in batches], dim=0)  
 texts = [\_['text'] for \_ in batches]  
 bboxes = [\_['bbox'] for \_ in batches]  
 hws = [\_['hw'] for \_ in batches]  
 image\_paths = [\_['image\_path'] for \_ in batches]  
 qids = [\_['question\_id'] for \_ in batches]  
 return pixel\_values, texts, bboxes, hws, image\_paths, qids  
  
def box\_iou(boxes1, boxes2):  
 """计算边界框的交并比(IoU)"""  
 area1 = box\_area(boxes1)  
 area2 = box\_area(boxes2)  
  
 lt = torch.max(boxes1[:, None, :2], boxes2[:, :2])  
 rb = torch.min(boxes1[:, None, 2:], boxes2[:, 2:])  
  
 wh = (rb - lt).clamp(min=0)  
 inter = wh[:, :, 0] \* wh[:, :, 1]  
  
 union = area1[:, None] + area2 - inter  
 iou = inter / union  
 return iou, union  
  
class RefCOCOEvaluator(BaseEvaluator):  
 """RefCOCO 评估器类"""  
   
 def \_\_init\_\_(self, model='internvl', \*\*kwargs):  
 # 初始化配置  
 self.model = model  
 self.kwargs = kwargs  
 self.batch\_size = kwargs.get('batch\_size', 1)  
 self.num\_beams = kwargs.get('num\_beams', 1)  
 self.temperature = kwargs.get('temperature', 0.0)  
 self.dynamic = kwargs.get('dynamic', False)  
 self.max\_num = kwargs.get('max\_num', 6)  
 self.num\_workers = kwargs.get('num\_workers', 1)  
 self.out\_dir = kwargs.get('out\_dir', 'results/grounding')  
 self.device = torch.device('cuda') if torch.cuda.is\_available() else torch.device('cpu')  
   
 # 数据集映射  
 self.ds\_collections = {  
 'refcoco\_val': 'data/refcoco/refcoco\_val.jsonl',  
 'refcoco\_testA': 'data/refcoco/refcoco\_testA.jsonl',  
 'refcoco\_testB': 'data/refcoco/refcoco\_testB.jsonl',  
 'refcoco+\_val': 'data/refcoco/refcoco+\_val.jsonl',  
 'refcoco+\_testA': 'data/refcoco/refcoco+\_testA.jsonl',  
 'refcoco+\_testB': 'data/refcoco/refcoco+\_testB.jsonl',  
 'refcocog\_val': 'data/refcoco/refcocog\_val.jsonl',  
 'refcocog\_test': 'data/refcoco/refcocog\_test.jsonl',  
 }  
   
 # 确保输出目录存在  
 os.makedirs(self.out\_dir, exist\_ok=True)  
   
 # 加载模型  
 self.load\_model()  
   
 def load\_model(self):  
 """加载模型和分词器"""  
 from internvl.model import load\_model\_and\_tokenizer  
 self.model, self.tokenizer = load\_model\_and\_tokenizer(self.model)  
   
 # 模型配置  
 self.image\_size = self.model.config.force\_image\_size or self.model.config.vision\_config.image\_size  
 self.use\_thumbnail = self.model.config.use\_thumbnail  
 self.prompt = 'Please provide the bounding box coordinate of the region this sentence describes: <ref>{}</ref>'  
   
 # 打印模型信息  
 total\_params = sum(p.numel() for p in self.model.parameters()) / 1e9  
 if total\_params > 20 or self.dynamic:  
 self.num\_beams = 1  
 print(f'[Model Info] Total params: {total\_params:.1f}B')  
 print(f'[Config] Image size: {self.image\_size}')  
 print(f'[Config] Dynamic image size: {self.dynamic}')  
 print(f'[Config] Use thumbnail: {self.use\_thumbnail}')  
 print(f'[Config] Max num: {self.max\_num}')  
   
 def build\_dataset(self, dataset\_name):  
 """构建数据集"""  
 return RefCOCODataset(  
 test\_file=self.ds\_collections[dataset\_name],  
 prompt=self.prompt,  
 input\_size=self.image\_size,  
 dynamic\_image\_size=self.dynamic,  
 use\_thumbnail=self.use\_thumbnail,  
 max\_num=self.max\_num  
 )  
   
 def generate(self, model, pixel\_values, question):  
 """生成模型预测"""  
 generation\_config = dict(  
 num\_beams=self.num\_beams,  
 max\_new\_tokens=100,  
 min\_new\_tokens=1,  
 do\_sample=True if self.temperature > 0 else False,  
 temperature=self.temperature,  
 )  
 return model.chat(  
 tokenizer=self.tokenizer,  
 pixel\_values=pixel\_values,  
 question=question,  
 generation\_config=generation\_config  
 )  
   
 def extract\_bbox(self, text):  
 """从文本中提取边界框坐标"""  
 predict\_bbox = re.findall(PATTERN, text)  
 try:  
 bbox = (  
 float(predict\_bbox[0][0]),   
 float(predict\_bbox[0][1]),   
 float(predict\_bbox[0][2]),   
 float(predict\_bbox[0][3])  
 )  
 except:  
 bbox = (0., 0., 0., 0.)  
 return bbox  
   
 def calculate\_iou(self, pred\_bbox, gt\_bbox, hw):  
 """计算交并比(IoU)"""  
 pred\_bbox = torch.tensor(pred\_bbox, dtype=torch.float32).view(-1, 4)  
   
 # 如果预测值总和大于4，说明是百分比值（0-1000）  
 if pred\_bbox.sum() >= 4:  
 pred\_bbox = pred\_bbox / 1000  
   
 # 将百分比坐标转换为实际像素坐标  
 pred\_bbox[:, 0::2] \*= hw[1] # 宽度  
 pred\_bbox[:, 1::2] \*= hw[0] # 高度  
   
 gt\_bbox = torch.tensor(gt\_bbox, dtype=torch.float32).view(-1, 4)  
 iou, \_ = box\_iou(pred\_bbox, gt\_bbox)  
 return iou.item()  
   
 def evaluate(self, dataset\_name):  
 """评估单个数据集"""  
 print(f'Evaluating {dataset\_name}...')  
 dataset = self.build\_dataset(dataset\_name)  
 dataloader = torch.utils.data.DataLoader(  
 dataset=dataset,  
 batch\_size=self.batch\_size,  
 num\_workers=self.num\_workers,  
 collate\_fn=partial(collate\_fn, tokenizer=self.tokenizer),  
 shuffle=False  
 )  
   
 outputs = []  
 for pixel\_values, texts, bboxes, hws, image\_paths, qids in tqdm(dataloader):  
 pixel\_values = pixel\_values.to(torch.bfloat16).to(self.device)  
   
 for i in range(len(texts)):  
 try:  
 answer = self.generate(  
 model=self.model,   
 pixel\_values=pixel\_values[i].unsqueeze(0),  
 question=texts[i]  
 )  
 except Exception as e:  
 print(f'Error generating answer: {e}')  
 answer = ''  
   
 outputs.append({  
 'question\_id': qids[i],  
 'image\_path': image\_paths[i],  
 'question': texts[i],  
 'answer': answer,  
 'gt\_bbox': bboxes[i],  
 'hw': hws[i]  
 })  
   
 # 计算精度  
 correct = 0  
 for output in outputs:  
 pred\_bbox = self.extract\_bbox(output['answer'])  
 iou = self.calculate\_iou(pred\_bbox, output['gt\_bbox'], output['hw'])  
 output['iou'] = iou  
 output['correct'] = int(iou >= 0.5)  
 if iou >= 0.5:  
 correct += 1  
   
 precision = correct / len(outputs)  
 print(f'Dataset: {dataset\_name}, Precision @ 0.5: {precision:.4f}')  
   
 # 保存结果  
 time\_prefix = time.strftime('%y%m%d%H%M%S', time.localtime())  
 result\_file = os.path.join(self.out\_dir, f'{dataset\_name}\_{time\_prefix}.json')  
 with open(result\_file, 'w') as f:  
 json.dump({  
 'dataset': dataset\_name,  
 'precision': precision,  
 'details': outputs  
 }, f, indent=2)  
   
 return precision  
   
 def run(self, datasets):  
 """运行评估多个数据集"""  
 results = {}  
 for ds\_name in datasets:  
 if ds\_name in self.ds\_collections:  
 precision = self.evaluate(ds\_name)  
 results[ds\_name] = precision  
   
 # 保存总结结果  
 summary\_file = os.path.join(self.out\_dir, 'summary.txt')  
 with open(summary\_file, 'w') as f:  
 for ds, prec in results.items():  
 f.write(f'{ds}: Precision @ 0.5 = {prec:.4f}\n')  
   
 print('Evaluation completed. Results saved to:', self.out\_dir)  
 return results  
  
def parse\_args():  
 """解析命令行参数"""  
 parser = argparse.ArgumentParser()  
 parser.add\_argument('--model', type=str, default='internvl', choices=supported\_VLM)  
 parser.add\_argument('--datasets', type=str, default='refcoco\_val,refcoco\_testA,refcoco\_testB')  
 parser.add\_argument('--batch-size', type=int, default=1)  
 parser.add\_argument('--num-beams', type=int, default=1)  
 parser.add\_argument('--temperature', type=float, default=0.0)  
 parser.add\_argument('--dynamic', action='store\_true')  
 parser.add\_argument('--max-num', type=int, default=6)  
 parser.add\_argument('--out-dir', type=str, default='results/grounding')  
 return parser.parse\_args()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 args = parse\_args()  
 datasets = args.datasets.split(',')  
   
 evaluator = RefCOCOEvaluator(  
 model=args.model,  
 batch\_size=args.batch\_size,  
 num\_beams=args.num\_beams,  
 temperature=args.temperature,  
 dynamic=args.dynamic,  
 max\_num=args.max\_num,  
 out\_dir=args.out\_dir  
 )  
   
 results = evaluator.run(datasets)

### 主要改进和迁移要点：

1. **类结构重构**：
   * 继承 BaseEvaluator 基类
   * 实现了 load\_model, build\_dataset, evaluate, run 等核心方法
   * 将原始脚本中的功能模块化为独立方法



1. **数据集处理**：
   * 保留 RefCOCODataset 类但适配到新框架
   * 使用 VLMEvalKit 的 Dataset 基类（如果可用）



1. **模型加载**：
   * 使用 VLMEvalKit 的模型加载机制
   * 支持命令行指定不同模型



1. **评估流程**：
   * 每个数据集单独评估并保存结果
   * 保留边界框提取和IoU计算逻辑
   * 添加结果验证和错误处理



1. **结果保存**：
   * 每个数据集评估结果单独保存为JSON文件
   * 包含详细预测结果和IoU值
   * 生成总结报告 summary.txt



1. **命令行接口**：
   * 使用 argparse 添加命令行参数
   * 支持选择不同模型和数据集



1. **分布式支持**：
   * 移除了原始脚本的DDP支持（VLMEvalKit通常有自己的分布式机制）
   * 保留批量处理以提高效率



### 使用方法：

Bashpython evaluate\_grounding.py \  
 --model=qwen-vl \  
 --datasets=refcoco\_val,refcoco\_testA \  
 --batch-size=4 \  
 --out-dir=results/grounding\_qwen

### 关键优势：

1. **与VLMEvalKit集成**：
   * 支持框架内的所有预训练模型
   * 统一的结果格式和报告生成



1. **可扩展性**：
   * 易于添加新模型或数据集
   * 模块化设计便于维护



1. **错误处理**：
   * 添加了异常捕获防止单次失败影响整体评估
   * 详细的预测结果记录便于调试



1. **结果可重现**：
   * 完整保存每次评估的设置和结果
   * 包含时间戳防止覆盖



这个实现保留了原始脚本的核心功能，同时符合 VLMEvalKit 框架的设计规范，使得 grounding 任务可以无缝集成到统一的评估流程中。