目录

[一、数据准备 1](#_Toc174089645)

[1.Doccano介绍 1](#_Toc174089646)

[2.Uie介绍 2](#_Toc174089647)

[3.模型部署步骤： 2](#_Toc174089648)

[二、数据处理 3](#_Toc174089649)

[三、预训练模型准备 5](#_Toc174089650)

[1.环境准备 5](#_Toc174089651)

[2.模型导出 6](#_Toc174089652)

[四、模型代码 7](#_Toc174089653)

[1.doccano.py: 8](#_Toc174089654)

[2.finetune.py: 13](#_Toc174089655)

[3.utils.py: 20](#_Toc174089656)

[五、模型训练 40](#_Toc174089657)

[1.Uie模型微调 40](#_Toc174089658)

[2.运行截图： 41](#_Toc174089659)

[六、实现效果 45](#_Toc174089660)

[1.test.py代码: 45](#_Toc174089661)

[2.运行截图： 48](#_Toc174089662)

[七、报错分析 50](#_Toc174089663)

[1.最初数据标注启动失败 50](#_Toc174089664)

[2.微调启动失败 51](#_Toc174089665)

[3.数据集格式错误 52](#_Toc174089666)

[4.微调参数错误 53](#_Toc174089667)

[5.选错数据标注类型 56](#_Toc174089668)

[6.运行模型未保存 57](#_Toc174089669)

[7.运行一半模型自动停止运行 57](#_Toc174089670)

# 一、数据准备

## 1.Doccano介绍

doccano是一个面向人类的开源文本注释工具。它为文本分类、序列标记和从序列到序列的任务提供了注释功能。可以为情绪分析、命名实体识别、文本摘要等创建标记数据。只要创建一个项目，上传数据，并开始进行注释，该工具可以在数小时内构建一个数据集。



## 2.Uie介绍

[UIE(Universal Information Extraction)](https://arxiv.org/pdf/2203.12277.pdf)：Yaojie Lu等人在ACL-2022中提出了通用信息抽取统一框架UIE。该框架实现了实体抽取、关系抽取、事件抽取、情感分析等任务的统一建模，并使得不同任务间具备良好的迁移和泛化能力。为了方便大家使用UIE的强大能力，PaddleNLP借鉴该论文的方法，基于ERNIE 3.0知识增强预训练模型，训练并开源了首个中文通用信息抽取模型UIE。该模型可以支持不限定行业领域和抽取目标的关键信息抽取，实现零样本快速冷启动，并具备优秀的小样本微调能力，快速适配特定的抽取目标。

**UIE的优势**

* **使用简单**：用户可以使用自然语言自定义抽取目标，无需训练即可统一抽取输入文本中的对应信息。**实现开箱即用，并满足各类信息抽取需求**。
* **降本增效**：以往的信息抽取技术需要大量标注数据才能保证信息抽取的效果，为了提高开发过程中的开发效率，减少不必要的重复工作时间，开放域信息抽取可以实现零样本（zero-shot）或者少样本（few-shot）抽取，**大幅度降低标注数据依赖，在降低成本的同时，还提升了效果**。
* **效果领先**：开放域信息抽取在多种场景，多种任务上，均有不俗的表现。

## 3.模型部署步骤：

1. 配置doccano数据标注环境
2. 进行数据标注
3. 导出数据
4. 将数据格式转换为prompt
5. 通过doccano.py进行数据集分批等操作
6. 配置模型运行环境
7. 下载uie-base模型
8. 测试测试样例
9. 修改参数微调模型
10. 成功保存微调模型
11. 编写实例代码，调用微调模型实现项目要求

# 二、数据处理

**利用doccano进行数据标注，通过docker容器运行doccano。**

作为一次性设置，创建如下的Docker容器：

docker pull doccano/doccano

docker container create --name doccano \

-e "ADMIN\_USERNAME=admin" \

-e "ADMIN\_EMAIL=admin@example.com" \

-e "ADMIN\_PASSWORD=password" \

-v doccano-db:/data \

-p 8000:8000 doccano/doccano

接下来，通过运行容器来启动doccano：

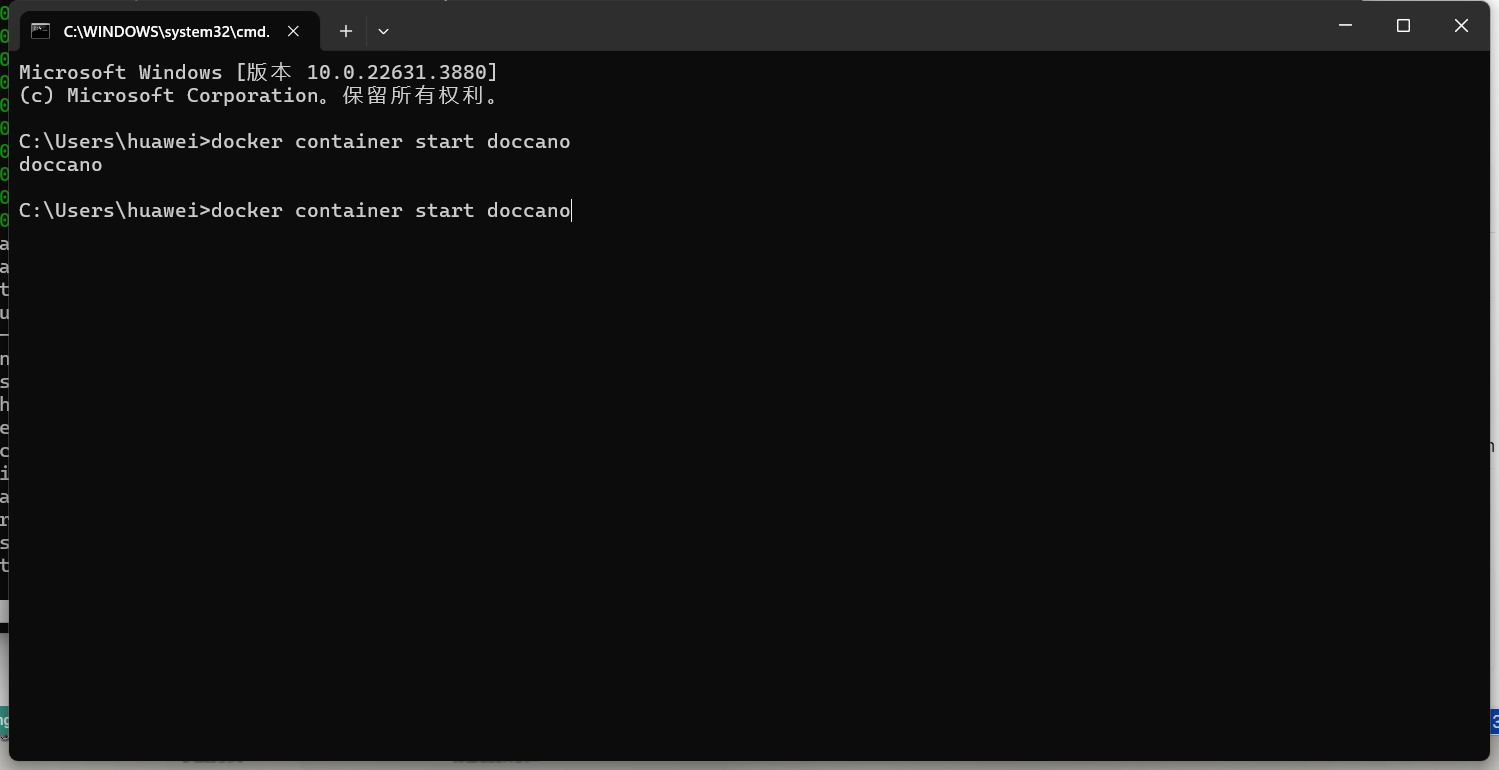
docker container start doccano

网页访问 <http://127.0.0.1:8000/>.

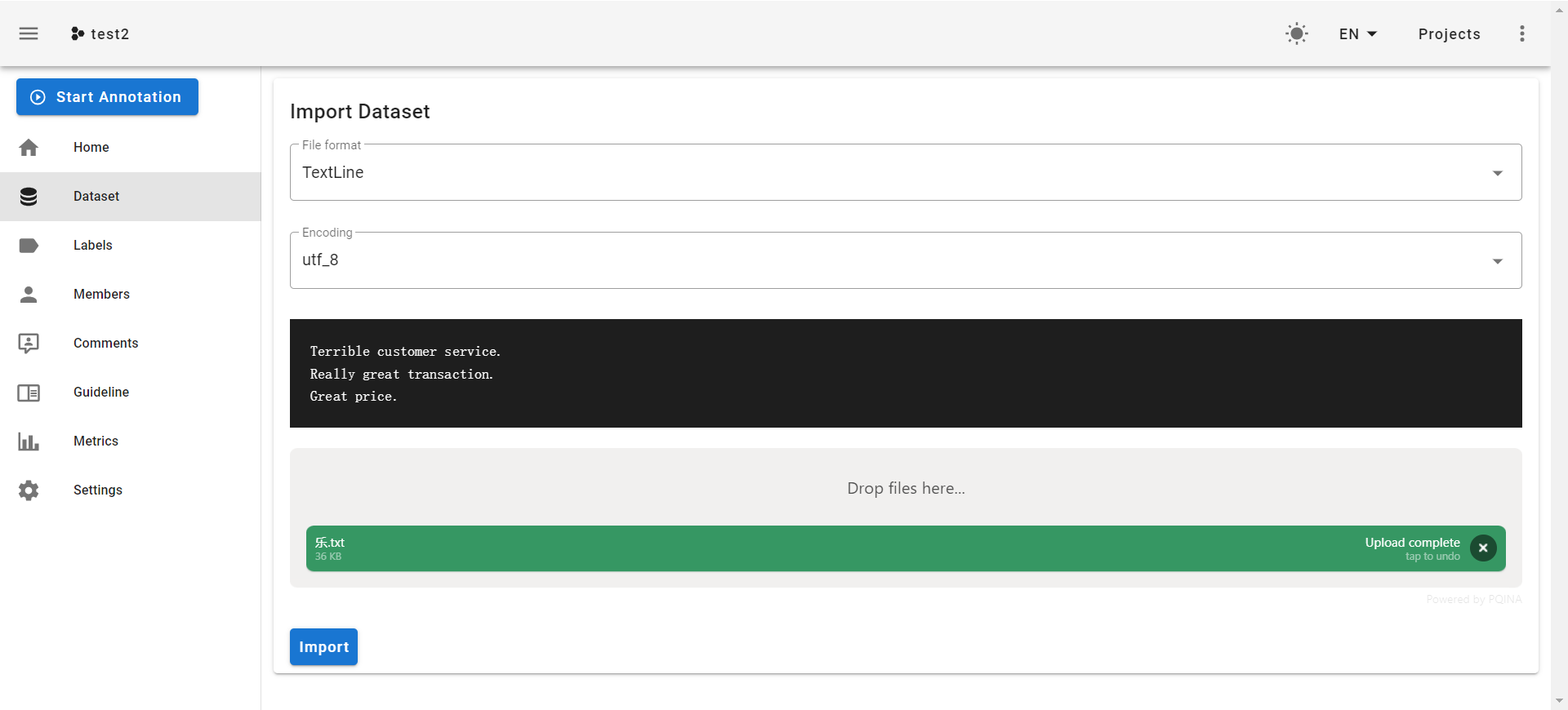
要停止容器，请执行以下操作：

docker container stop doccano -t 5

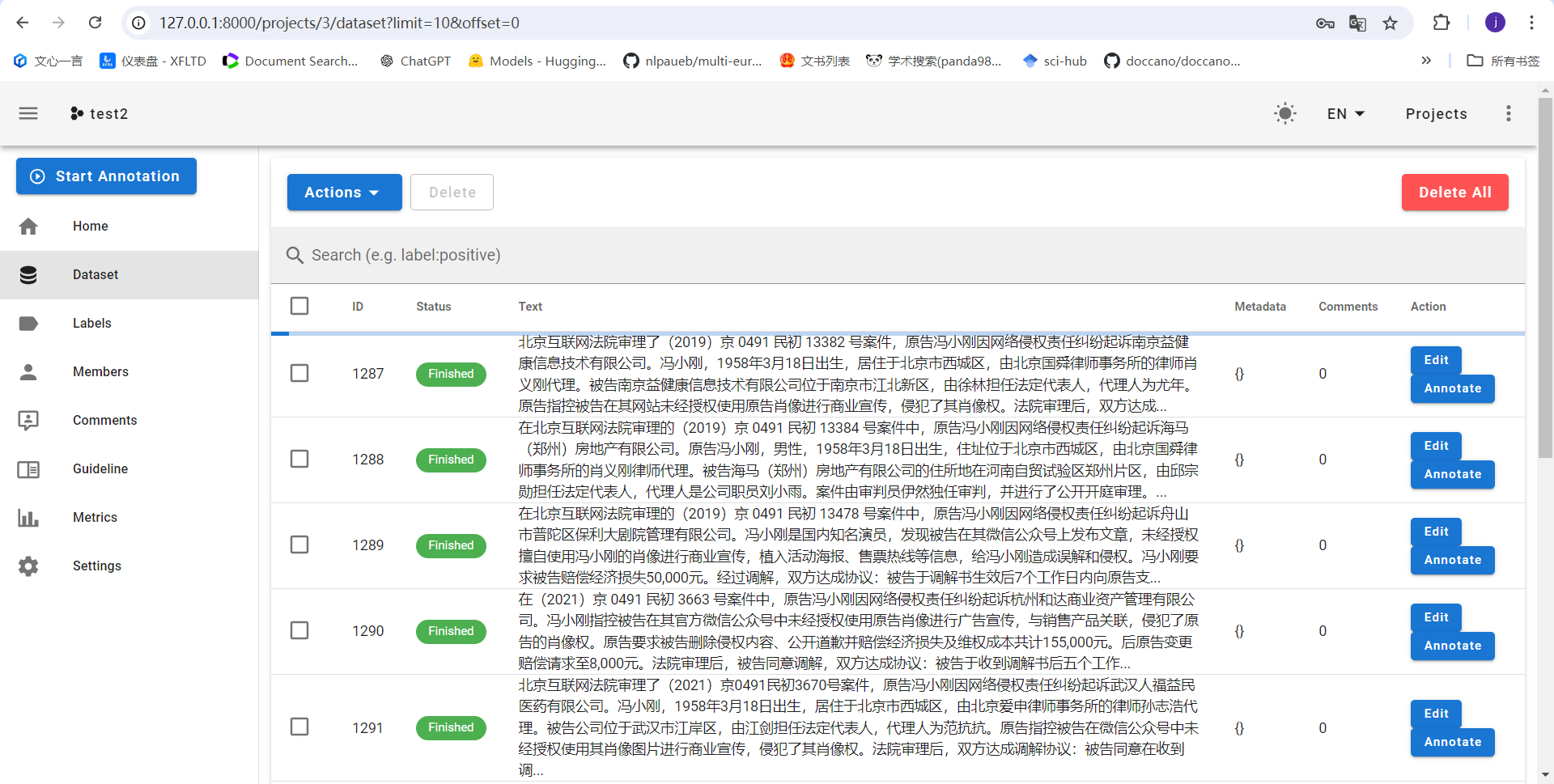
在容器中创建的所有数据都将在重新启动期间持续存在。



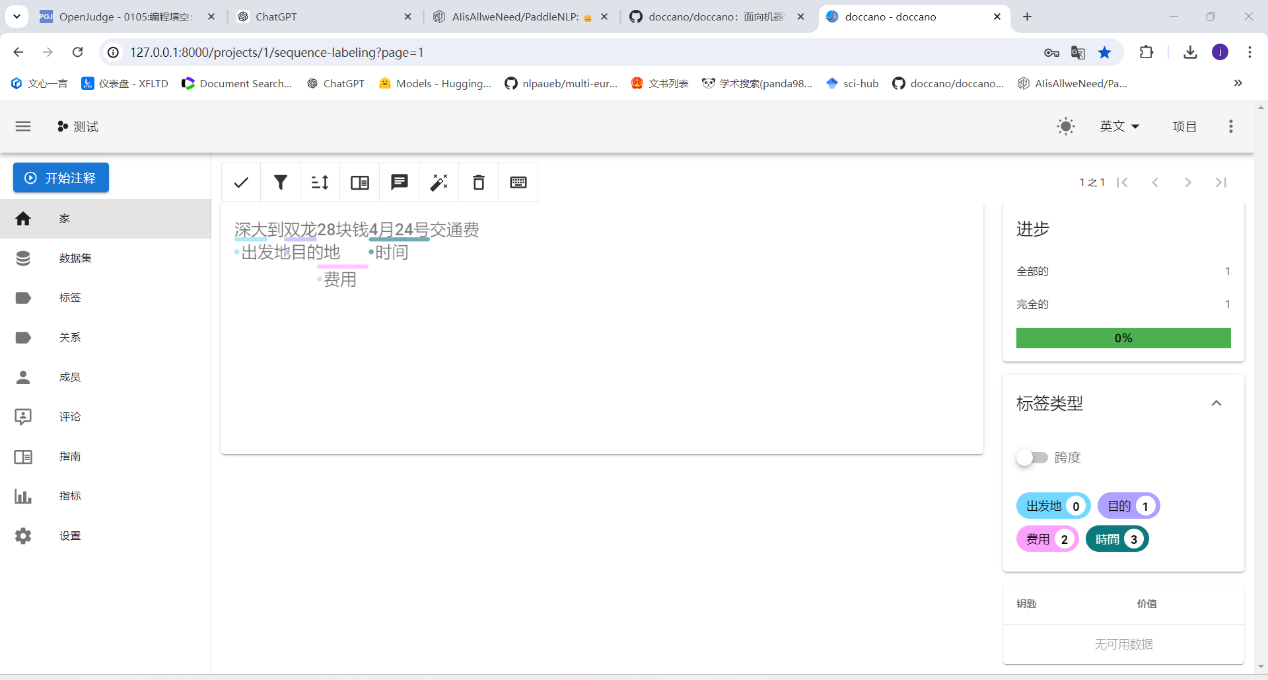
启动docker



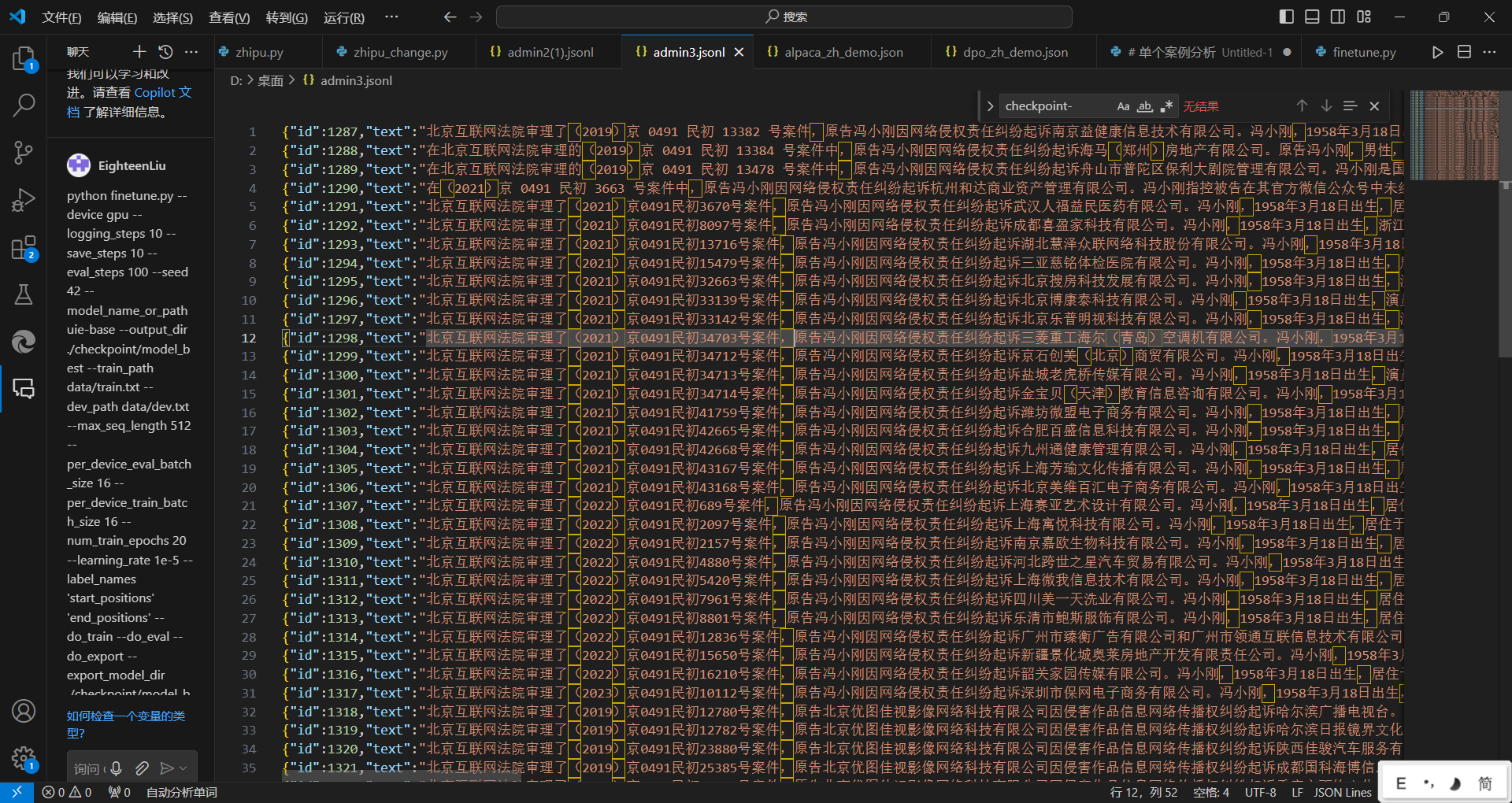
数据集导入页面



数据标注页面



单条数据标注页面



标注导出后数据集

# 三、预训练模型准备

以下是 UIE Python 端的部署流程，包括环境准备、模型导出和使用示例。

## 1.环境准备

UIE的部署分为 CPU 和 GPU 两种情况，请根据你的部署环境安装对应的依赖。

**CPU端**

CPU端的部署请使用如下命令安装所需依赖：

pip install -r deploy/python/requirements\_cpu.txt

**GPU端**

为了在 GPU 上获得最佳的推理性能和稳定性，请先确保机器已正确安装 NVIDIA 相关驱动和基础软件，确保 CUDA >= 11.2，cuDNN >= 8.1.1，并使用以下命令安装所需依赖

pip install -r deploy/python/requirements\_gpu.txt

如果有模型推理加速、内存显存占用优化的需求，并且 GPU 设备的 CUDA 计算能力 (CUDA Compute Capability) 大于等于 7.0，例如 V100、T4、A10、A100/GA100、Jetson AGX Xavier 等显卡，推荐使用半精度（FP16）部署。直接使用微调后导出的 FP32 模型，运行时设置 --use\_fp16 即可。

如果 GPU 设备的 CUDA 计算能力较低，低于 7.0，只支持 FP32 部署，微调后导出模型直接部署即可。

## 2.模型导出

模型训练、压缩时已经自动进行了静态图的导出，保存路径${finetuned\_model} 下应该有 \*.pdimodel、\*.pdiparams 模型文件可用于推理。

**推理**

**CPU端推理样例**

在CPU端，请使用如下命令进行部署

python deploy/python/infer\_cpu.py --model\_path\_prefix ${finetuned\_model}/model

部署UIE-M模型

python deploy/python/infer\_cpu.py --model\_path\_prefix ${finetuned\_model}/model --multilingual

可配置参数说明：

model\_path\_prefix: 用于推理的Paddle模型文件路径，需加上文件前缀名称。例如模型文件路径为./export/model.pdiparams，则传入./export/model。

position\_prob：模型对于span的起始位置/终止位置的结果概率 0~1 之间，返回结果去掉小于这个阈值的结果，默认为 0.5，span 的最终概率输出为起始位置概率和终止位置概率的乘积。

max\_seq\_len: 文本最大切分长度，输入超过最大长度时会对输入文本进行自动切分，默认为 512。

batch\_size: 批处理大小，请结合机器情况进行调整，默认为 4。

multilingual：是否是跨语言模型，用 "uie-m-base", "uie-m-large" 等模型进微调得到的模型是多语言模型，需要设置为 True；默认为 False。

**GPU端推理样例**

在GPU端，请使用如下命令进行部署

python deploy/python/infer\_gpu.py --model\_path\_prefix ${finetuned\_model}/model --use\_fp16 --device\_id 0

部署UIE-M模型

python deploy/python/infer\_gpu.py --model\_path\_prefix ${finetuned\_model}/model --use\_fp16 --device\_id 0 --multilingual

可配置参数说明：

model\_path\_prefix: 用于推理的 Paddle 模型文件路径，需加上文件前缀名称。例如模型文件路径为./export/model.pdiparams，则传入./export/model。

use\_fp16: FP32 模型是否使用 FP16 进行加速，使用 FP32、INT8 推理时不需要设置，默认关闭。

position\_prob：模型对于span的起始位置/终止位置的结果概率0~1之间，返回结果去掉小于这个阈值的结果，默认为 0.5，span 的最终概率输出为起始位置概率和终止位置概率的乘积。

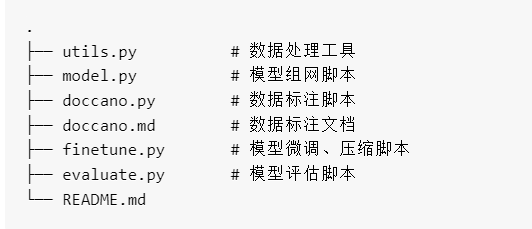
max\_seq\_len: 文本最大切分长度，输入超过最大长度时会对输入文本进行自动切分，默认为 512。

batch\_size: 批处理大小，请结合机器情况进行调整，默认为 4。

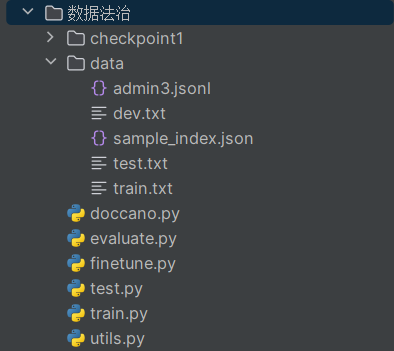
device\_id: GPU 设备 ID，默认为 0。

multilingual：是否是跨语言模型，用 "uie-m-base", "uie-m-large" 等模型进微调得到的模型是多语言模型，需要设置为 True；默认为 False。

# 四、模型代码



代码结构



实际代码结构

## 1.doccano.py:

import argparse  
import json  
import os  
import time  
from decimal import Decimal  
  
import numpy as np  
from utils import convert\_cls\_examples, convert\_ext\_examples, set\_seed  
#convert\_cls\_examples,  
  
from paddlenlp.trainer.argparser import strtobool  
from paddlenlp.utils.log import logger  
  
  
def do\_convert():  
 set\_seed(args.seed)  
  
 tic\_time = time.time()  
 if not os.path.exists(args.doccano\_file):  
 raise ValueError("Please input the correct path of doccano file.")  
  
 if not os.path.exists(args.save\_dir):  
 os.makedirs(args.save\_dir)  
  
 if len(args.splits) != 0 and len(args.splits) != 3:  
 raise ValueError("Only []/ len(splits)==3 accepted for splits.")  
  
 def \_check\_sum(splits):  
 return Decimal(str(splits[0])) + Decimal(str(splits[1])) + Decimal(str(splits[2])) == Decimal("1")  
  
 if len(args.splits) == 3 and not \_check\_sum(args.splits):  
 raise ValueError("Please set correct splits, sum of elements in splits should be equal to 1.")  
  
 with open(args.doccano\_file, "r", encoding="utf-8") as f:  
 raw\_examples = f.readlines()  
  
 def \_create\_ext\_examples(  
 examples,  
 negative\_ratio,  
 prompt\_prefix="情感倾向",  
 options=["正向", "负向"],  
 separator="##",  
 shuffle=False,  
 is\_train=True,  
 schema\_lang="ch",  
 ):  
 entities, relations, aspects = convert\_ext\_examples(  
 examples, negative\_ratio, prompt\_prefix, options, separator, is\_train, schema\_lang  
 )  
 examples = entities + relations + aspects  
 if shuffle:  
 indexes = np.random.permutation(len(examples))  
 examples = [examples[i] for i in indexes]  
 return examples  
  
 def \_create\_cls\_examples(examples, prompt\_prefix, options, shuffle=False):  
 examples = convert\_cls\_examples(examples, prompt\_prefix, options)  
 if shuffle:  
 indexes = np.random.permutation(len(examples))  
 examples = [examples[i] for i in indexes]  
 return examples  
  
 def \_save\_examples(save\_dir, file\_name, examples):  
 count = 0  
 save\_path = os.path.join(save\_dir, file\_name)  
 with open(save\_path, "w", encoding="utf-8") as f:  
 for example in examples:  
 f.write(json.dumps(example, ensure\_ascii=False) + "\n")  
 count += 1  
 logger.info("Save %d examples to %s." % (count, save\_path))  
  
 if len(args.splits) == 0:  
 if args.task\_type == "ext":  
 examples = \_create\_ext\_examples(  
 raw\_examples,  
 args.negative\_ratio,  
 args.prompt\_prefix,  
 args.options,  
 args.separator,  
 args.is\_shuffle,  
 schema\_lang=args.schema\_lang,  
 )  
 else:  
 examples = \_create\_cls\_examples(raw\_examples, args.prompt\_prefix, args.options, args.is\_shuffle)  
 \_save\_examples(args.save\_dir, "train.txt", examples)  
 else:  
 if args.is\_shuffle:  
 indexes = np.random.permutation(len(raw\_examples))  
 index\_list = indexes.tolist()  
 raw\_examples = [raw\_examples[i] for i in indexes]  
 else:  
 index\_list = list(range(len(raw\_examples)))  
  
 i1, i2, \_ = args.splits  
 p1 = int(len(raw\_examples) \* i1)  
 p2 = int(len(raw\_examples) \* (i1 + i2))  
  
 train\_ids = index\_list[:p1]  
 dev\_ids = index\_list[p1:p2]  
 test\_ids = index\_list[p2:]  
  
 with open(os.path.join(args.save\_dir, "sample\_index.json"), "w") as fp:  
 maps = {"train\_ids": train\_ids, "dev\_ids": dev\_ids, "test\_ids": test\_ids}  
 fp.write(json.dumps(maps))  
  
 if args.task\_type == "ext":  
 train\_examples = \_create\_ext\_examples(  
 raw\_examples[:p1],  
 args.negative\_ratio,  
 args.prompt\_prefix,  
 args.options,  
 args.separator,  
 args.is\_shuffle,  
 schema\_lang=args.schema\_lang,  
 )  
 dev\_examples = \_create\_ext\_examples(  
 raw\_examples[p1:p2],  
 -1,  
 args.prompt\_prefix,  
 args.options,  
 args.separator,  
 is\_train=False,  
 schema\_lang=args.schema\_lang,  
 )  
 test\_examples = \_create\_ext\_examples(  
 raw\_examples[p2:],  
 -1,  
 args.prompt\_prefix,  
 args.options,  
 args.separator,  
 is\_train=False,  
 schema\_lang=args.schema\_lang,  
 )  
 else:  
 train\_examples = \_create\_cls\_examples(raw\_examples[:p1], args.prompt\_prefix, args.options)  
 dev\_examples = \_create\_cls\_examples(raw\_examples[p1:p2], args.prompt\_prefix, args.options)  
 test\_examples = \_create\_cls\_examples(raw\_examples[p2:], args.prompt\_prefix, args.options)  
  
 \_save\_examples(args.save\_dir, "train.txt", train\_examples)  
 \_save\_examples(args.save\_dir, "dev.txt", dev\_examples)  
 \_save\_examples(args.save\_dir, "test.txt", test\_examples)  
  
 logger.info("Finished! It takes %.2f seconds" % (time.time() - tic\_time))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # yapf: disable  
 parser = argparse.ArgumentParser()  
  
 parser.add\_argument("--doccano\_file", default="./data/admin3.jsonl", type=str, help="The doccano file exported from doccano platform.")  
 parser.add\_argument("--save\_dir", default="./data", type=str, help="The path of data that you wanna save.")  
 parser.add\_argument("--negative\_ratio", default=5, type=int, help="Used only for the extraction task, the ratio of positive and negative samples, number of negtive samples = negative\_ratio \* number of positive samples")  
 parser.add\_argument("--splits", default=[0.8, 0.1, 0.1], type=float, nargs="\*", help="The ratio of samples in datasets. [0.6, 0.2, 0.2] means 60% samples used for training, 20% for evaluation and 20% for test.")  
 parser.add\_argument("--task\_type", choices=['ext', 'cls'], default="ext", type=str, help="Select task type, ext for the extraction task and cls for the classification task, defaults to ext.")  
 parser.add\_argument("--options", default=["正向", "负向"], type=str, nargs="+", help="Used only for the classification task, the options for classification")  
 parser.add\_argument("--prompt\_prefix", default="情感倾向", type=str, help="Used only for the classification task, the prompt prefix for classification")  
 parser.add\_argument("--is\_shuffle", default="True", type=strtobool, help="Whether to shuffle the labeled dataset, defaults to True.")  
 parser.add\_argument("--seed", type=int, default=1000, help="Random seed for initialization")  
 parser.add\_argument("--separator", type=str, default='##', help="Used only for entity/aspect-level classification task, separator for entity label and classification label")  
 parser.add\_argument("--schema\_lang", choices=["ch", "en"], default="ch", help="Select the language type for schema.")  
  
 args = parser.parse\_args()  
 # yapf: enable  
  
 do\_convert()

## 2.finetune.py:

import os  
from dataclasses import dataclass, field  
from functools import partial  
from typing import List, Optional  
  
import paddle  
from utils import convert\_example, reader  
  
from paddlenlp.data import DataCollatorWithPadding  
from paddlenlp.datasets import load\_dataset  
from paddlenlp.metrics import SpanEvaluator  
from paddlenlp.trainer import (  
 CompressionArguments,  
 PdArgumentParser,  
 Trainer,  
 get\_last\_checkpoint,  
)  
from paddlenlp.transformers import UIE, UIEM, AutoTokenizer, export\_model  
from paddlenlp.utils.log import logger  
  
  
@dataclass  
class DataArguments:  
 *"""  
 Arguments pertaining to what data we are going to input our model for training and eval.  
 Using `PdArgumentParser` we can turn this class into argparse arguments to be able to  
 specify them on the command line.  
 """* train\_path: str = field(  
 default=None, metadata={"help": "The name of the dataset to use (via the datasets library)."}  
 )  
  
 dev\_path: str = field(  
 default=None, metadata={"help": "The name of the dataset to use (via the datasets library)."}  
 )  
  
 max\_seq\_length: Optional[int] = field(  
 default=512,  
 metadata={  
 "help": "The maximum total input sequence length after tokenization. Sequences longer "  
 "than this will be truncated, sequences shorter will be padded."  
 },  
 )  
  
 dynamic\_max\_length: Optional[List[int]] = field(  
 default=None,  
 metadata={"help": "dynamic max length from batch, it can be array of length, eg: 16 32 64 128"},  
 )  
  
  
@dataclass  
class ModelArguments:  
 *"""  
 Arguments pertaining to which model/config/tokenizer we are going to fine-tune from.  
 """* model\_name\_or\_path: Optional[str] = field(  
 default="uie-base",  
 metadata={  
 "help": "Path to pretrained model, such as 'uie-base', 'uie-tiny', "  
 "'uie-medium', 'uie-mini', 'uie-micro', 'uie-nano', 'uie-base-en', "  
 "'uie-m-base', 'uie-m-large', or finetuned model path."  
 },  
 )  
 export\_model\_dir: Optional[str] = field(  
 default=None,  
 metadata={"help": "Path to directory to store the exported inference model."},  
 )  
 multilingual: bool = field(default=False, metadata={"help": "Whether the model is a multilingual model."})  
  
  
def main():  
 parser = PdArgumentParser((ModelArguments, DataArguments, CompressionArguments))  
 model\_args, data\_args, training\_args = parser.parse\_args\_into\_dataclasses()  
 training\_args.resume\_from\_checkpoint = 'D:/python/bert/数据法治/checkpoint1/model\_best/checkpoint-70'  
  
 if model\_args.model\_name\_or\_path in ["uie-m-base", "uie-m-large"]:  
 model\_args.multilingual = True  
  
 # Log model and data config  
 training\_args.print\_config(model\_args, "Model")  
 training\_args.print\_config(data\_args, "Data")  
  
 paddle.set\_device(training\_args.device)  
  
 # Log on each process the small summary:  
 logger.warning(  
 f"Process rank: {training\_args.local\_rank}, device: {training\_args.device}, world\_size: {training\_args.world\_size}, "  
 + f"distributed training: {bool(training\_args.local\_rank != -1)}, 16-bits training: {training\_args.fp16}"  
 )  
  
 # Detecting last checkpoint.  
 last\_checkpoint = None  
 if os.path.isdir(training\_args.output\_dir) and training\_args.do\_train and not training\_args.overwrite\_output\_dir:  
 last\_checkpoint = get\_last\_checkpoint(training\_args.output\_dir)  
 if last\_checkpoint is None and len(os.listdir(training\_args.output\_dir)) > 0:  
 raise ValueError(  
 f"Output directory ({training\_args.output\_dir}) already exists and is not empty. "  
 "Use --overwrite\_output\_dir to overcome."  
 )  
 elif last\_checkpoint is not None and training\_args.resume\_from\_checkpoint is None:  
 logger.info(  
 f"Checkpoint detected, resuming training at {last\_checkpoint}. To avoid this behavior, change "  
 "the `--output\_dir` or add `--overwrite\_output\_dir` to train from scratch."  
 )  
 checkpoint = 'D:/python/bert/数据法治/checkpoint1/model\_best/checkpoint-70'  
 if training\_args.resume\_from\_checkpoint is not None:  
 checkpoint = training\_args.resume\_from\_checkpoint  
 elif last\_checkpoint is not None:  
 checkpoint = last\_checkpoint  
  
 tokenizer = AutoTokenizer.from\_pretrained(model\_args.model\_name\_or\_path)  
 if model\_args.multilingual:  
 model = UIEM.from\_pretrained(model\_args.model\_name\_or\_path)  
 else:  
 model = UIE.from\_pretrained(model\_args.model\_name\_or\_path)  
  
 train\_ds = load\_dataset(reader, data\_path=data\_args.train\_path, max\_seq\_len=data\_args.max\_seq\_length, lazy=False)  
 dev\_ds = load\_dataset(reader, data\_path=data\_args.dev\_path, max\_seq\_len=data\_args.max\_seq\_length, lazy=False)  
  
 trans\_fn = partial(  
 convert\_example,  
 tokenizer=tokenizer,  
 max\_seq\_len=data\_args.max\_seq\_length,  
 multilingual=model\_args.multilingual,  
 dynamic\_max\_length=data\_args.dynamic\_max\_length,  
 )  
  
 train\_ds = train\_ds.map(trans\_fn)  
 dev\_ds = dev\_ds.map(trans\_fn)  
  
 if training\_args.device == "npu":  
 data\_collator = DataCollatorWithPadding(tokenizer, padding="longest")  
 else:  
 data\_collator = DataCollatorWithPadding(tokenizer)  
  
 criterion = paddle.nn.BCELoss()  
  
 def uie\_loss\_func(outputs, labels):  
 start\_ids, end\_ids = labels  
 start\_prob, end\_prob = outputs  
 start\_ids = paddle.cast(start\_ids, "float32")  
 end\_ids = paddle.cast(end\_ids, "float32")  
 loss\_start = criterion(start\_prob, start\_ids)  
 loss\_end = criterion(end\_prob, end\_ids)  
 loss = (loss\_start + loss\_end) / 2.0  
 return loss  
  
 def compute\_metrics(p):  
 metric = SpanEvaluator()  
 start\_prob, end\_prob = p.predictions  
 start\_ids, end\_ids = p.label\_ids  
 metric.reset()  
  
 num\_correct, num\_infer, num\_label = metric.compute(start\_prob, end\_prob, start\_ids, end\_ids)  
 metric.update(num\_correct, num\_infer, num\_label)  
 precision, recall, f1 = metric.accumulate()  
 metric.reset()  
  
 return {"precision": precision, "recall": recall, "f1": f1}  
  
 trainer = Trainer(  
 model=model,  
 criterion=uie\_loss\_func,  
 args=training\_args,  
 data\_collator=data\_collator,  
 train\_dataset=train\_ds if training\_args.do\_train or training\_args.do\_compress else None,  
 eval\_dataset=dev\_ds if training\_args.do\_eval or training\_args.do\_compress else None,  
 tokenizer=tokenizer,  
 compute\_metrics=compute\_metrics,  
 )  
  
 trainer.optimizer = paddle.optimizer.AdamW(  
 learning\_rate=training\_args.learning\_rate, parameters=model.parameters()  
 )  
 checkpoint = None  
 if training\_args.resume\_from\_checkpoint is not None:  
 checkpoint = training\_args.resume\_from\_checkpoint  
 elif last\_checkpoint is not None:  
 checkpoint = last\_checkpoint  
  
 # Training  
 if training\_args.do\_train:  
 train\_result = trainer.train(resume\_from\_checkpoint=checkpoint)  
 metrics = train\_result.metrics  
 trainer.save\_model()  
 trainer.log\_metrics("train", metrics)  
 trainer.save\_metrics("train", metrics)  
 trainer.save\_state()  
  
 # Evaluate and tests model  
 if training\_args.do\_eval:  
 eval\_metrics = trainer.evaluate()  
 trainer.log\_metrics("eval", eval\_metrics)  
  
 # export inference model  
 if training\_args.do\_export:  
 # You can also load from certain checkpoint  
 # trainer.load\_state\_dict\_from\_checkpoint("/path/to/checkpoint/")  
 if training\_args.device == "npu":  
 # npu will transform int64 to int32 for internal calculation.  
 # To reduce useless transformation, we feed int32 inputs.  
 input\_spec\_dtype = "int32"  
 else:  
 input\_spec\_dtype = "int64"  
 if model\_args.multilingual:  
 input\_spec = [  
 paddle.static.InputSpec(shape=[None, None], dtype=input\_spec\_dtype, name="input\_ids"),  
 paddle.static.InputSpec(shape=[None, None], dtype=input\_spec\_dtype, name="position\_ids"),  
 ]  
 else:  
 input\_spec = [  
 paddle.static.InputSpec(shape=[None, None], dtype=input\_spec\_dtype, name="input\_ids"),  
 paddle.static.InputSpec(shape=[None, None], dtype=input\_spec\_dtype, name="token\_type\_ids"),  
 paddle.static.InputSpec(shape=[None, None], dtype=input\_spec\_dtype, name="position\_ids"),  
 paddle.static.InputSpec(shape=[None, None], dtype=input\_spec\_dtype, name="attention\_mask"),  
 ]  
 if model\_args.export\_model\_dir is None:  
 model\_args.export\_model\_dir = os.path.join(training\_args.output\_dir, "export")  
 export\_model(model=trainer.model, input\_spec=input\_spec, path=model\_args.export\_model\_dir)  
 if training\_args.do\_compress:  
  
 @paddle.no\_grad()  
 def custom\_evaluate(self, model, data\_loader):  
 metric = SpanEvaluator()  
 model.eval()  
 metric.reset()  
 for batch in data\_loader:  
 if model\_args.multilingual:  
 logits = model(input\_ids=batch["input\_ids"], position\_ids=batch["position\_ids"])  
 else:  
 logits = model(  
 input\_ids=batch["input\_ids"],  
 token\_type\_ids=batch["token\_type\_ids"],  
 position\_ids=batch["position\_ids"],  
 attention\_mask=batch["attention\_mask"],  
 )  
 start\_prob, end\_prob = logits  
 start\_ids, end\_ids = batch["start\_positions"], batch["end\_positions"]  
 num\_correct, num\_infer, num\_label = metric.compute(start\_prob, end\_prob, start\_ids, end\_ids)  
 metric.update(num\_correct, num\_infer, num\_label)  
 precision, recall, f1 = metric.accumulate()  
 logger.info("f1: %s, precision: %s, recall: %s" % (f1, precision, f1))  
 model.train()  
 return f1  
  
 trainer.compress(custom\_evaluate=custom\_evaluate)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

## 3.utils.py:

import json  
import math  
import random  
import re  
from typing import List, Optional  
  
import numpy as np  
import paddle  
from tqdm import tqdm  
  
from paddlenlp.utils.log import logger  
  
  
def set\_seed(seed):  
 paddle.seed(seed)  
 random.seed(seed)  
 np.random.seed(seed)  
  
  
def create\_data\_loader(dataset, mode="train", batch\_size=1, trans\_fn=None):  
 *"""  
 Create dataloader.  
 Args:  
 dataset(obj:`paddle.io.Dataset`): Dataset instance.  
 mode(obj:`str`, optional, defaults to obj:`train`): If mode is 'train', it will shuffle the dataset randomly.  
 batch\_size(obj:`int`, optional, defaults to 1): The sample number of a mini-batch.  
 trans\_fn(obj:`callable`, optional, defaults to `None`): function to convert a data sample to input ids, etc.  
 Returns:  
 dataloader(obj:`paddle.io.DataLoader`): The dataloader which generates batches.  
 """* if trans\_fn:  
 dataset = dataset.map(trans\_fn)  
  
 shuffle = True if mode == "train" else False  
 if mode == "train":  
 sampler = paddle.io.DistributedBatchSampler(dataset=dataset, batch\_size=batch\_size, shuffle=shuffle)  
 else:  
 sampler = paddle.io.BatchSampler(dataset=dataset, batch\_size=batch\_size, shuffle=shuffle)  
 dataloader = paddle.io.DataLoader(dataset, batch\_sampler=sampler, return\_list=True)  
 return dataloader  
  
  
def map\_offset(ori\_offset, offset\_mapping):  
 *"""  
 map ori offset to token offset  
 """* for index, span in enumerate(offset\_mapping):  
 if span[0] <= ori\_offset < span[1]:  
 return index  
 return -1  
  
  
def reader(data\_path, max\_seq\_len=512):  
 *"""  
 read json  
 """* with open(data\_path, "r", encoding="utf-8") as f:  
 for line in f:  
 json\_line = json.loads(line)  
 content = json\_line["content"].strip()  
 prompt = json\_line["prompt"]  
 # Model Input is aslike: [CLS] Prompt [SEP] Content [SEP]  
 # It include three summary tokens.  
 if max\_seq\_len <= len(prompt) + 3:  
 raise ValueError("The value of max\_seq\_len is too small, please set a larger value")  
 max\_content\_len = max\_seq\_len - len(prompt) - 3  
 if len(content) <= max\_content\_len:  
 yield json\_line  
 else:  
 result\_list = json\_line["result\_list"]  
 json\_lines = []  
 accumulate = 0  
 while True:  
 cur\_result\_list = []  
 for result in result\_list:  
 if result["end"] - result["start"] > max\_content\_len:  
 logger.warning(  
 "result['end'] - result ['start'] exceeds max\_content\_len, which will result in no valid instance being returned"  
 )  
 if (  
 result["start"] + 1 <= max\_content\_len < result["end"]  
 and result["end"] - result["start"] <= max\_content\_len  
 ):  
 max\_content\_len = result["start"]  
 break  
  
 cur\_content = content[:max\_content\_len]  
 res\_content = content[max\_content\_len:]  
  
 while True:  
 if len(result\_list) == 0:  
 break  
 elif result\_list[0]["end"] <= max\_content\_len:  
 if result\_list[0]["end"] > 0:  
 cur\_result = result\_list.pop(0)  
 cur\_result\_list.append(cur\_result)  
 else:  
 cur\_result\_list = [result for result in result\_list]  
 break  
 else:  
 break  
  
 json\_line = {"content": cur\_content, "result\_list": cur\_result\_list, "prompt": prompt}  
 json\_lines.append(json\_line)  
  
 for result in result\_list:  
 if result["end"] <= 0:  
 break  
 result["start"] -= max\_content\_len  
 result["end"] -= max\_content\_len  
 accumulate += max\_content\_len  
 max\_content\_len = max\_seq\_len - len(prompt) - 3  
 if len(res\_content) == 0:  
 break  
 elif len(res\_content) < max\_content\_len:  
 json\_line = {"content": res\_content, "result\_list": result\_list, "prompt": prompt}  
 json\_lines.append(json\_line)  
 break  
 else:  
 content = res\_content  
  
 for json\_line in json\_lines:  
 yield json\_line  
  
  
def unify\_prompt\_name(prompt):  
 # The classification labels are shuffled during finetuning, so they need  
 # to be unified during evaluation.  
 if re.search(r"\[.\*?\]$", prompt):  
 prompt\_prefix = prompt[: prompt.find("[", 1)]  
 cls\_options = re.search(r"\[.\*?\]$", prompt).group()[1:-1].split(",")  
 cls\_options = sorted(list(set(cls\_options)))  
 cls\_options = ",".join(cls\_options)  
 prompt = prompt\_prefix + "[" + cls\_options + "]"  
 return prompt  
 return prompt  
  
  
def get\_relation\_type\_dict(relation\_data, schema\_lang="ch"):  
 def compare(a, b, schema\_lang="ch"):  
 if schema\_lang == "ch":  
 a = a[::-1]  
 b = b[::-1]  
  
 res = ""  
 for i in range(min(len(a), len(b))):  
 if a[i] == b[i]:  
 res += a[i]  
 else:  
 break  
 if res == "":  
 return res  
 if schema\_lang == "ch" and res[::-1][0] == "的":  
 return res[::-1][1:]  
 elif schema\_lang == "en" and res[-3:] == " of":  
 return res[:-3]  
 return ""  
  
 relation\_type\_dict = {}  
 added\_list = []  
 for i in range(len(relation\_data)):  
 added = False  
 if relation\_data[i][0] not in added\_list:  
 for j in range(i + 1, len(relation\_data)):  
 match = compare(relation\_data[i][0], relation\_data[j][0], schema\_lang=schema\_lang)  
 if match != "":  
 match = unify\_prompt\_name(match)  
 if relation\_data[i][0] not in added\_list:  
 added\_list.append(relation\_data[i][0])  
 relation\_type\_dict.setdefault(match, []).append(relation\_data[i][1])  
 added\_list.append(relation\_data[j][0])  
 relation\_type\_dict.setdefault(match, []).append(relation\_data[j][1])  
 added = True  
 if not added:  
 added\_list.append(relation\_data[i][0])  
 if schema\_lang == "ch":  
 suffix = relation\_data[i][0].rsplit("的", 1)[1]  
 suffix = unify\_prompt\_name(suffix)  
 relation\_type = suffix  
 else:  
 prefix = relation\_data[i][0].split(" of ", 1)[0]  
 prefix = unify\_prompt\_name(prefix)  
 relation\_type = prefix  
 relation\_type\_dict.setdefault(relation\_type, []).append(relation\_data[i][1])  
 return relation\_type\_dict  
  
  
def add\_entity\_negative\_example(examples, texts, prompts, label\_set, negative\_ratio):  
 negative\_examples = []  
 positive\_examples = []  
 with tqdm(total=len(prompts)) as pbar:  
 for i, prompt in enumerate(prompts):  
 redundants = list(set(label\_set) ^ set(prompt))  
 redundants.sort()  
  
 num\_positive = len(examples[i])  
 if num\_positive != 0:  
 actual\_ratio = math.ceil(len(redundants) / num\_positive)  
 else:  
 # Set num\_positive to 1 for text without positive example  
 num\_positive, actual\_ratio = 1, 0  
  
 if actual\_ratio <= negative\_ratio or negative\_ratio == -1:  
 idxs = [k for k in range(len(redundants))]  
 else:  
 idxs = random.sample(range(0, len(redundants)), negative\_ratio \* num\_positive)  
  
 for idx in idxs:  
 negative\_result = {"content": texts[i], "result\_list": [], "prompt": redundants[idx]}  
 negative\_examples.append(negative\_result)  
 positive\_examples.extend(examples[i])  
 pbar.update(1)  
 return positive\_examples, negative\_examples  
  
  
def add\_relation\_negative\_example(redundants, text, num\_positive, ratio):  
 added\_example = []  
 rest\_example = []  
  
 if num\_positive != 0:  
 actual\_ratio = math.ceil(len(redundants) / num\_positive)  
 else:  
 # Set num\_positive to 1 for text without positive example  
 num\_positive, actual\_ratio = 1, 0  
  
 all\_idxs = [k for k in range(len(redundants))]  
 if actual\_ratio <= ratio or ratio == -1:  
 idxs = all\_idxs  
 rest\_idxs = []  
 else:  
 idxs = random.sample(range(0, len(redundants)), ratio \* num\_positive)  
 rest\_idxs = list(set(all\_idxs) ^ set(idxs))  
  
 for idx in idxs:  
 negative\_result = {"content": text, "result\_list": [], "prompt": redundants[idx]}  
 added\_example.append(negative\_result)  
  
 for rest\_idx in rest\_idxs:  
 negative\_result = {"content": text, "result\_list": [], "prompt": redundants[rest\_idx]}  
 rest\_example.append(negative\_result)  
  
 return added\_example, rest\_example  
  
  
def add\_full\_negative\_example(examples, texts, relation\_prompts, predicate\_set, subject\_goldens, schema\_lang="ch"):  
 with tqdm(total=len(relation\_prompts)) as pbar:  
 for i, relation\_prompt in enumerate(relation\_prompts):  
 negative\_sample = []  
 for subject in subject\_goldens[i]:  
 for predicate in predicate\_set:  
 # The relation prompt is constructed as follows:  
 # subject + "的" + predicate -> Chinese  
 # predicate + " of " + subject -> English  
 if schema\_lang == "ch":  
 prompt = subject + "的" + predicate  
 else:  
 prompt = predicate + " of " + subject  
 if prompt not in relation\_prompt:  
 negative\_result = {"content": texts[i], "result\_list": [], "prompt": prompt}  
 negative\_sample.append(negative\_result)  
 examples[i].extend(negative\_sample)  
 pbar.update(1)  
 return examples  
  
  
def generate\_cls\_example(text, labels, prompt\_prefix, options):  
 random.shuffle(options)  
 cls\_options = ",".join(options)  
 prompt = prompt\_prefix + "[" + cls\_options + "]"  
  
 result\_list = []  
 example = {"content": text, "result\_list": result\_list, "prompt": prompt}  
 for label in labels:  
 start = prompt.rfind(label) - len(prompt) - 1  
 end = start + len(label)  
 result = {"text": label, "start": start, "end": end}  
 example["result\_list"].append(result)  
 return example  
  
  
def convert\_cls\_examples(raw\_examples, prompt\_prefix="情感倾向", options=["正向", "负向"]):  
 *"""  
 Convert labeled data export from doccano for classification task.  
 """* examples = []  
 logger.info("Converting doccano data...")  
 with tqdm(total=len(raw\_examples)):  
 for line in raw\_examples:  
 items = json.loads(line)  
 # Compatible with doccano >= 1.6.2  
 if "data" in items.keys():  
 text, labels = items["data"], items["label"]  
 else:  
 text, labels = items["text"], items["label"]  
 example = generate\_cls\_example(text, labels, prompt\_prefix, options)  
 examples.append(example)  
 return examples  
  
  
def convert\_ext\_examples(  
 raw\_examples,  
 negative\_ratio,  
 prompt\_prefix="情感倾向",  
 options=["正向", "负向"],  
 separator="##",  
 is\_train=True,  
 schema\_lang="ch",  
):  
 *"""  
 Convert labeled data export from doccano for extraction and aspect-level classification task.  
 """* def \_sep\_cls\_label(label, separator):  
 label\_list = label.split(separator)  
 if len(label\_list) == 1:  
 return label\_list[0], None  
 return label\_list[0], label\_list[1:]  
  
 texts = []  
 entity\_examples = []  
 relation\_examples = []  
 entity\_cls\_examples = []  
 entity\_prompts = []  
 relation\_prompts = []  
 entity\_label\_set = []  
 entity\_name\_set = []  
 predicate\_set = []  
 subject\_goldens = []  
 inverse\_relation\_list = []  
 predicate\_list = []  
  
 logger.info("Converting doccano data...")  
 with tqdm(total=len(raw\_examples)) as pbar:  
 for line in raw\_examples:  
 items = json.loads(line)  
 entity\_id = 0  
 if "data" in items.keys():  
 relation\_mode = False  
 if isinstance(items["label"], dict) and "entities" in items["label"].keys():  
 relation\_mode = True  
 text = items["data"]  
 entities = []  
 relations = []  
 if not relation\_mode:  
 # Export file in JSONL format which doccano < 1.7.0  
 # e.g. {"data": "", "label": [ [0, 2, "ORG"], ... ]}  
 for item in items["label"]:  
 entity = {"id": entity\_id, "start\_offset": item[0], "end\_offset": item[1], "label": item[2]}  
 entities.append(entity)  
 entity\_id += 1  
 else:  
 # Export file in JSONL format for relation labeling task which doccano < 1.7.0  
 # e.g. {"data": "", "label": {"relations": [ {"id": 0, "start\_offset": 0, "end\_offset": 6, "label": "ORG"}, ... ], "entities": [ {"id": 0, "from\_id": 0, "to\_id": 1, "type": "foundedAt"}, ... ]}}  
 entities.extend([entity for entity in items["label"]["entities"]])  
 if "relations" in items["label"].keys():  
 relations.extend([relation for relation in items["label"]["relations"]])  
 else:  
 # Export file in JSONL format which doccano >= 1.7.0  
 # e.g. {"text": "", "label": [ [0, 2, "ORG"], ... ]}  
 if "label" in items.keys():  
 text = items["text"]  
 entities = []  
 for item in items["label"]:  
 entity = {"id": entity\_id, "start\_offset": item[0], "end\_offset": item[1], "label": item[2]}  
 entities.append(entity)  
 entity\_id += 1  
 relations = []  
 else:  
 # Export file in JSONL (relation) format  
 # e.g. {"text": "", "relations": [ {"id": 0, "start\_offset": 0, "end\_offset": 6, "label": "ORG"}, ... ], "entities": [ {"id": 0, "from\_id": 0, "to\_id": 1, "type": "foundedAt"}, ... ]}  
 text, relations, entities = items["text"], items["relations"], items["entities"]  
 texts.append(text)  
  
 entity\_example = []  
 entity\_prompt = []  
 entity\_example\_map = {}  
 entity\_map = {} # id to entity name  
 for entity in entities:  
 entity\_name = text[entity["start\_offset"] : entity["end\_offset"]]  
 entity\_map[entity["id"]] = {  
 "name": entity\_name,  
 "start": entity["start\_offset"],  
 "end": entity["end\_offset"],  
 }  
  
 entity\_label, entity\_cls\_label = \_sep\_cls\_label(entity["label"], separator)  
  
 # Define the prompt prefix for entity-level classification  
 # xxx + "的" + 情感倾向 -> Chinese  
 # Sentiment classification + " of " + xxx -> English  
 if schema\_lang == "ch":  
 entity\_cls\_prompt\_prefix = entity\_name + "的" + prompt\_prefix  
 else:  
 entity\_cls\_prompt\_prefix = prompt\_prefix + " of " + entity\_name  
 if entity\_cls\_label is not None:  
 entity\_cls\_example = generate\_cls\_example(  
 text, entity\_cls\_label, entity\_cls\_prompt\_prefix, options  
 )  
  
 entity\_cls\_examples.append(entity\_cls\_example)  
  
 result = {"text": entity\_name, "start": entity["start\_offset"], "end": entity["end\_offset"]}  
 if entity\_label not in entity\_example\_map.keys():  
 entity\_example\_map[entity\_label] = {  
 "content": text,  
 "result\_list": [result],  
 "prompt": entity\_label,  
 }  
 else:  
 entity\_example\_map[entity\_label]["result\_list"].append(result)  
  
 if entity\_label not in entity\_label\_set:  
 entity\_label\_set.append(entity\_label)  
 if entity\_name not in entity\_name\_set:  
 entity\_name\_set.append(entity\_name)  
 entity\_prompt.append(entity\_label)  
  
 for v in entity\_example\_map.values():  
 entity\_example.append(v)  
  
 entity\_examples.append(entity\_example)  
 entity\_prompts.append(entity\_prompt)  
  
 subject\_golden = [] # Golden entity inputs  
 relation\_example = []  
 relation\_prompt = []  
 relation\_example\_map = {}  
 inverse\_relation = []  
 predicates = []  
 for relation in relations:  
 predicate = relation["type"]  
 subject\_id = relation["from\_id"]  
 object\_id = relation["to\_id"]  
 # The relation prompt is constructed as follows:  
 # subject + "的" + predicate -> Chinese  
 # predicate + " of " + subject -> English  
 if schema\_lang == "ch":  
 prompt = entity\_map[subject\_id]["name"] + "的" + predicate  
 inverse\_negative = entity\_map[object\_id]["name"] + "的" + predicate  
 else:  
 prompt = predicate + " of " + entity\_map[subject\_id]["name"]  
 inverse\_negative = predicate + " of " + entity\_map[object\_id]["name"]  
  
 if entity\_map[subject\_id]["name"] not in subject\_golden:  
 subject\_golden.append(entity\_map[subject\_id]["name"])  
 result = {  
 "text": entity\_map[object\_id]["name"],  
 "start": entity\_map[object\_id]["start"],  
 "end": entity\_map[object\_id]["end"],  
 }  
  
 inverse\_relation.append(inverse\_negative)  
 predicates.append(predicate)  
  
 if prompt not in relation\_example\_map.keys():  
 relation\_example\_map[prompt] = {"content": text, "result\_list": [result], "prompt": prompt}  
 else:  
 relation\_example\_map[prompt]["result\_list"].append(result)  
  
 if predicate not in predicate\_set:  
 predicate\_set.append(predicate)  
 relation\_prompt.append(prompt)  
  
 for v in relation\_example\_map.values():  
 relation\_example.append(v)  
  
 relation\_examples.append(relation\_example)  
 relation\_prompts.append(relation\_prompt)  
 subject\_goldens.append(subject\_golden)  
 inverse\_relation\_list.append(inverse\_relation)  
 predicate\_list.append(predicates)  
 pbar.update(1)  
  
 logger.info("Adding negative samples for first stage prompt...")  
 positive\_examples, negative\_examples = add\_entity\_negative\_example(  
 entity\_examples, texts, entity\_prompts, entity\_label\_set, negative\_ratio  
 )  
 if len(positive\_examples) == 0:  
 all\_entity\_examples = []  
 else:  
 all\_entity\_examples = positive\_examples + negative\_examples  
  
 all\_relation\_examples = []  
 if len(predicate\_set) != 0:  
 logger.info("Adding negative samples for second stage prompt...")  
 if is\_train:  
  
 positive\_examples = []  
 negative\_examples = []  
 per\_n\_ratio = negative\_ratio // 3  
  
 with tqdm(total=len(texts)) as pbar:  
 for i, text in enumerate(texts):  
 negative\_example = []  
 collects = []  
 num\_positive = len(relation\_examples[i])  
  
 # 1. inverse\_relation\_list  
 redundants1 = inverse\_relation\_list[i]  
  
 # 2. entity\_name\_set ^ subject\_goldens[i]  
 redundants2 = []  
 if len(predicate\_list[i]) != 0:  
 nonentity\_list = list(set(entity\_name\_set) ^ set(subject\_goldens[i]))  
 nonentity\_list.sort()  
  
 if schema\_lang == "ch":  
 redundants2 = [  
 nonentity + "的" + predicate\_list[i][random.randrange(len(predicate\_list[i]))]  
 for nonentity in nonentity\_list  
 ]  
 else:  
 redundants2 = [  
 predicate\_list[i][random.randrange(len(predicate\_list[i]))] + " of " + nonentity  
 for nonentity in nonentity\_list  
 ]  
  
 # 3. entity\_label\_set ^ entity\_prompts[i]  
 redundants3 = []  
 if len(subject\_goldens[i]) != 0:  
 non\_ent\_label\_list = list(set(entity\_label\_set) ^ set(entity\_prompts[i]))  
 non\_ent\_label\_list.sort()  
  
 if schema\_lang == "ch":  
 redundants3 = [  
 subject\_goldens[i][random.randrange(len(subject\_goldens[i]))] + "的" + non\_ent\_label  
 for non\_ent\_label in non\_ent\_label\_list  
 ]  
 else:  
 redundants3 = [  
 non\_ent\_label + " of " + subject\_goldens[i][random.randrange(len(subject\_goldens[i]))]  
 for non\_ent\_label in non\_ent\_label\_list  
 ]  
  
 redundants\_list = [redundants1, redundants2, redundants3]  
  
 for redundants in redundants\_list:  
 added, rest = add\_relation\_negative\_example(  
 redundants,  
 texts[i],  
 num\_positive,  
 per\_n\_ratio,  
 )  
 negative\_example.extend(added)  
 collects.extend(rest)  
  
 num\_sup = num\_positive \* negative\_ratio - len(negative\_example)  
 if num\_sup > 0 and collects:  
 if num\_sup > len(collects):  
 idxs = [k for k in range(len(collects))]  
 else:  
 idxs = random.sample(range(0, len(collects)), num\_sup)  
 for idx in idxs:  
 negative\_example.append(collects[idx])  
  
 positive\_examples.extend(relation\_examples[i])  
 negative\_examples.extend(negative\_example)  
 pbar.update(1)  
 all\_relation\_examples = positive\_examples + negative\_examples  
 else:  
 relation\_examples = add\_full\_negative\_example(  
 relation\_examples, texts, relation\_prompts, predicate\_set, subject\_goldens, schema\_lang=schema\_lang  
 )  
 all\_relation\_examples = [r for relation\_example in relation\_examples for r in relation\_example]  
 return all\_entity\_examples, all\_relation\_examples, entity\_cls\_examples  
  
  
def get\_dynamic\_max\_length(examples, default\_max\_length: int, dynamic\_max\_length: List[int]) -> int:  
 *"""get max\_length by examples which you can change it by examples in batch"""* cur\_length = len(examples[0]["input\_ids"])  
 max\_length = default\_max\_length  
 for max\_length\_option in sorted(dynamic\_max\_length):  
 if cur\_length <= max\_length\_option:  
 max\_length = max\_length\_option  
 break  
 return max\_length  
  
  
def convert\_example(  
 example, tokenizer, max\_seq\_len, multilingual=False, dynamic\_max\_length: Optional[List[int]] = None  
):  
 *"""  
 example: {  
 title  
 prompt  
 content  
 result\_list  
 }  
 """* if dynamic\_max\_length is not None:  
 temp\_encoded\_inputs = tokenizer(  
 text=[example["prompt"]],  
 text\_pair=[example["content"]],  
 truncation=True,  
 max\_seq\_len=max\_seq\_len,  
 return\_attention\_mask=True,  
 return\_position\_ids=True,  
 return\_dict=False,  
 return\_offsets\_mapping=True,  
 )  
 max\_length = get\_dynamic\_max\_length(  
 examples=temp\_encoded\_inputs, default\_max\_length=max\_seq\_len, dynamic\_max\_length=dynamic\_max\_length  
 )  
 # always pad to max\_length  
 encoded\_inputs = tokenizer(  
 text=[example["prompt"]],  
 text\_pair=[example["content"]],  
 truncation=True,  
 max\_seq\_len=max\_length,  
 pad\_to\_max\_seq\_len=True,  
 return\_attention\_mask=True,  
 return\_position\_ids=True,  
 return\_dict=False,  
 return\_offsets\_mapping=True,  
 )  
 start\_ids = [0.0 for x in range(max\_length)]  
 end\_ids = [0.0 for x in range(max\_length)]  
 else:  
 encoded\_inputs = tokenizer(  
 text=[example["prompt"]],  
 text\_pair=[example["content"]],  
 truncation=True,  
 max\_seq\_len=max\_seq\_len,  
 pad\_to\_max\_seq\_len=True,  
 return\_attention\_mask=True,  
 return\_position\_ids=True,  
 return\_dict=False,  
 return\_offsets\_mapping=True,  
 )  
 start\_ids = [0.0 for x in range(max\_seq\_len)]  
 end\_ids = [0.0 for x in range(max\_seq\_len)]  
  
 encoded\_inputs = encoded\_inputs[0]  
 offset\_mapping = [list(x) for x in encoded\_inputs["offset\_mapping"]]  
 bias = 0  
 for index in range(1, len(offset\_mapping)):  
 mapping = offset\_mapping[index]  
 if mapping[0] == 0 and mapping[1] == 0 and bias == 0:  
 bias = offset\_mapping[index - 1][1] + 1 # Includes [SEP] token  
 if mapping[0] == 0 and mapping[1] == 0:  
 continue  
 offset\_mapping[index][0] += bias  
 offset\_mapping[index][1] += bias  
 for item in example["result\_list"]:  
 start = map\_offset(item["start"] + bias, offset\_mapping)  
 end = map\_offset(item["end"] - 1 + bias, offset\_mapping)  
 start\_ids[start] = 1.0  
 end\_ids[end] = 1.0  
 if multilingual:  
 tokenized\_output = {  
 "input\_ids": encoded\_inputs["input\_ids"],  
 "position\_ids": encoded\_inputs["position\_ids"],  
 "start\_positions": start\_ids,  
 "end\_positions": end\_ids,  
 }  
 else:  
 tokenized\_output = {  
 "input\_ids": encoded\_inputs["input\_ids"],  
 "token\_type\_ids": encoded\_inputs["token\_type\_ids"],  
 "position\_ids": encoded\_inputs["position\_ids"],  
 "attention\_mask": encoded\_inputs["attention\_mask"],  
 "start\_positions": start\_ids,  
 "end\_positions": end\_ids,  
 }  
 return tokenized\_output

# 五、模型训练

## 1.Uie模型微调

将data文件下的doccano导出标注文件doccano.json转化到data文件中

python doccano.py --doccano\_file ./data/admin3.jsonl --task\_type ext --save\_dir ./data --splits 0.8 0.2 0 --schema\_lang ch

使用下面的命令，使用 uie-base 作为预训练模型进行模型微调，将微调后的模型保存至./checkpoint/model\_best

python finetune.py --device cpu --logging\_steps 10 --save\_steps 10 --eval\_steps 10 --seed 50 --model\_name\_or\_path uie-base --output\_dir ./checkpoint1/model\_best --train\_path data/train.txt --dev\_path data/dev.txt --max\_seq\_length 512 --per\_device\_eval\_batch\_size 4 --per\_device\_train\_batch\_size 4 --num\_train\_epochs 20 --learning\_rate 1e-5 --label\_names 'start\_positions' 'end\_positions' --do\_train --do\_export --export\_model\_dir ./checkpoint1/model\_best --overwrite\_output\_dir --disable\_tqdm False --save\_total\_limit 2

GPU

python finetune.py --device gpu --logging\_steps 10 --save\_steps 10 --eval\_steps 100 --seed 42 --model\_name\_or\_path uie-base --output\_dir ./checkpoint/model\_best --train\_path data/train.txt --dev\_path data/dev.txt --max\_seq\_length 512 --per\_device\_eval\_batch\_size 16 --per\_device\_train\_batch\_size 16 --num\_train\_epochs 20 --learning\_rate 1e-5 --label\_names 'start\_positions' 'end\_positions' --do\_train --do\_eval --do\_export --export\_model\_dir ./checkpoint/model\_best --overwrite\_output\_dir --disable\_tqdm True --metric\_for\_best\_model eval\_f1 --load\_best\_model\_at\_end True --save\_total\_limit 1

python -u -m paddle.distributed.launch --gpus "1,2,3" finetune.py --device gpu --logging\_steps 10 --save\_steps 100 --eval\_steps 100 --seed 42 --model\_name\_or\_path uie-m-large --output\_dir ./checkpoint/model\_best --train\_path data/train.txt --dev\_path data/dev.txt --max\_seq\_length 512 --per\_device\_eval\_batch\_size 16 --per\_device\_train\_batch\_size 16 --num\_train\_epochs 100 --learning\_rate 1e-5 --do\_train --do\_export --export\_model\_dir ./checkpoint/model\_best --label\_names 'start\_positions' 'end\_positions'--overwrite\_output\_dir --disable\_tqdm True --metric\_for\_best\_model eval\_f1 --load\_best\_model\_at\_end True --save\_total\_limit 2 –multilingual True

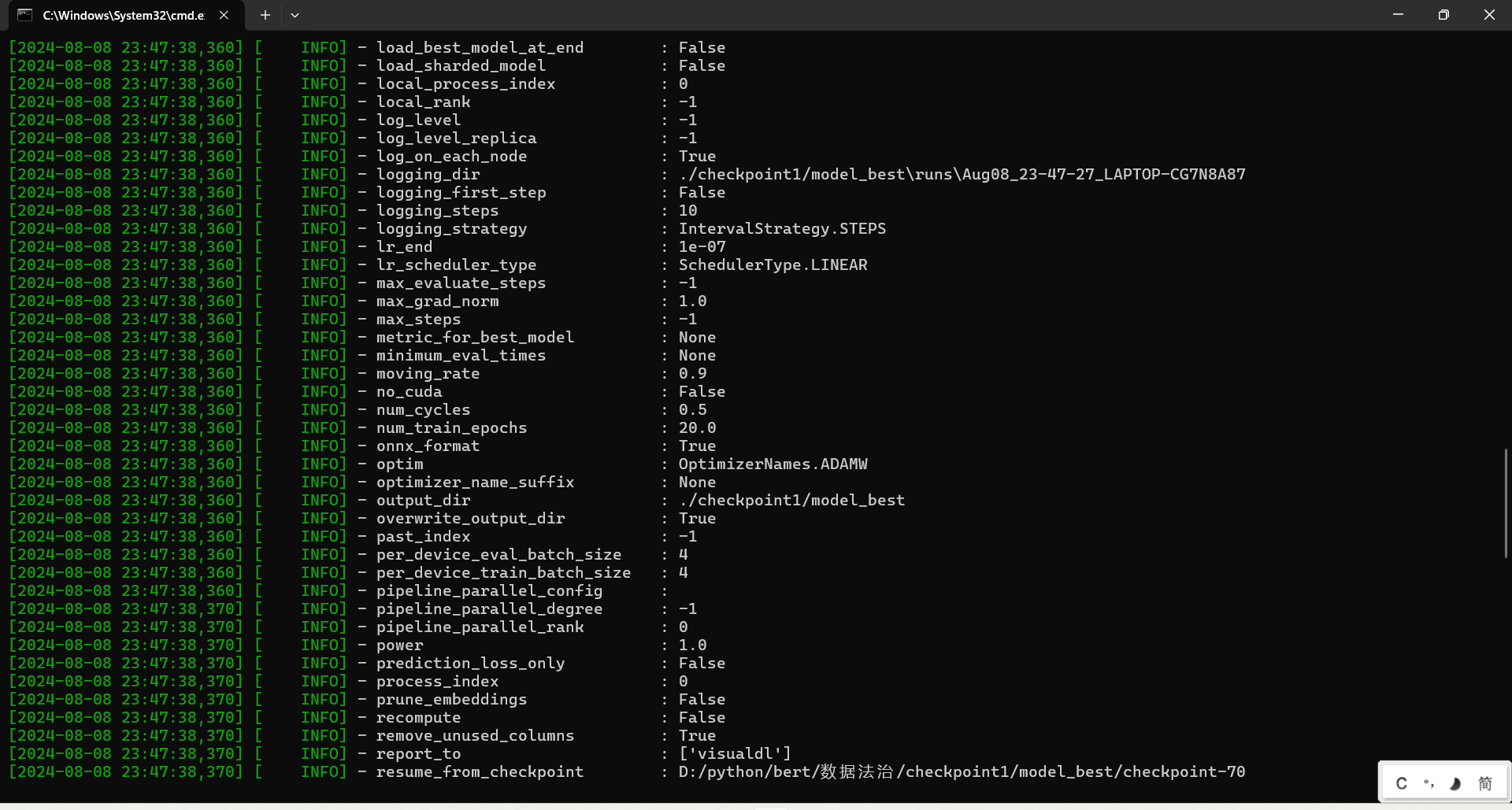
## 2.运行截图：

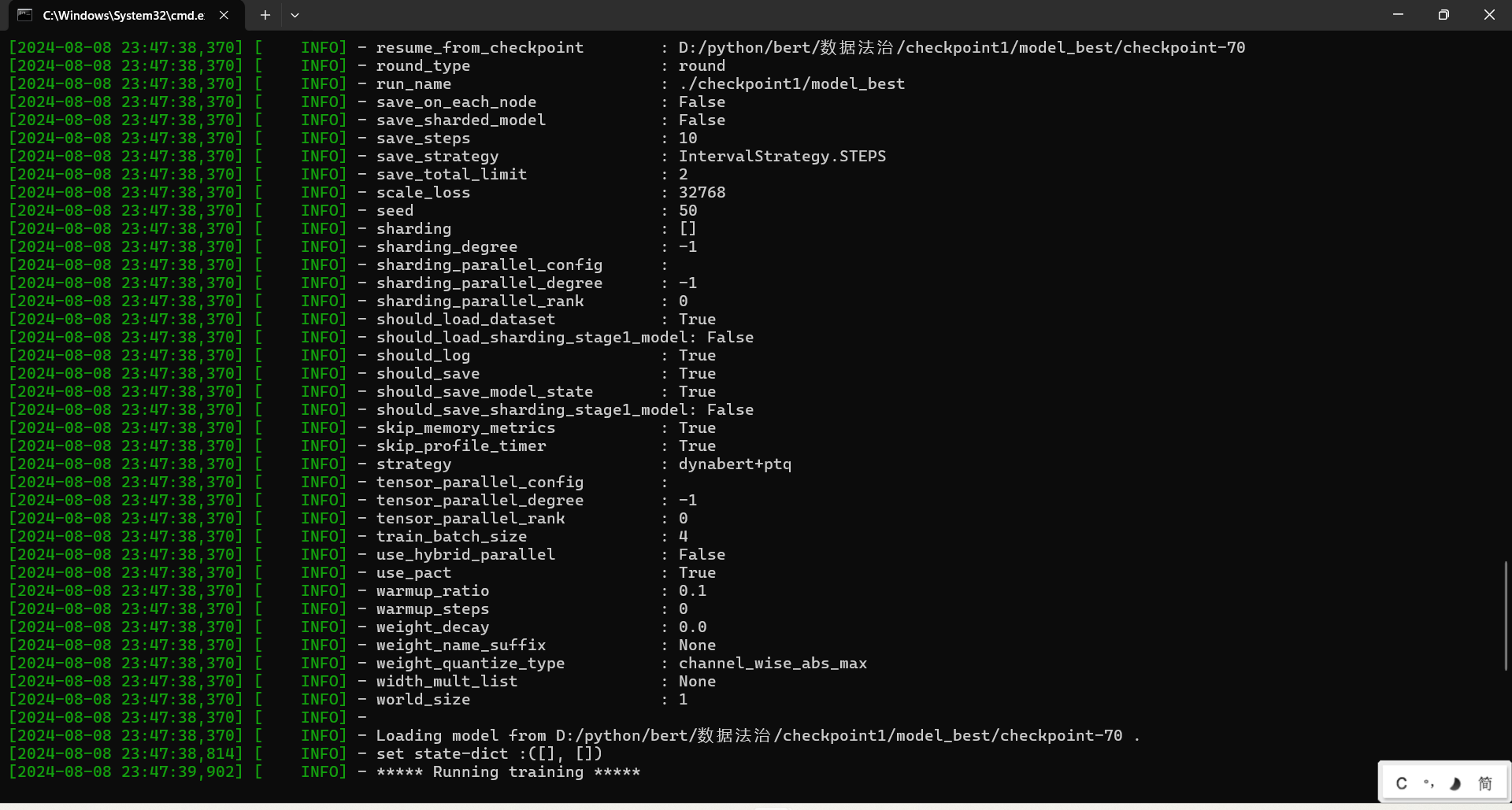


训练页1

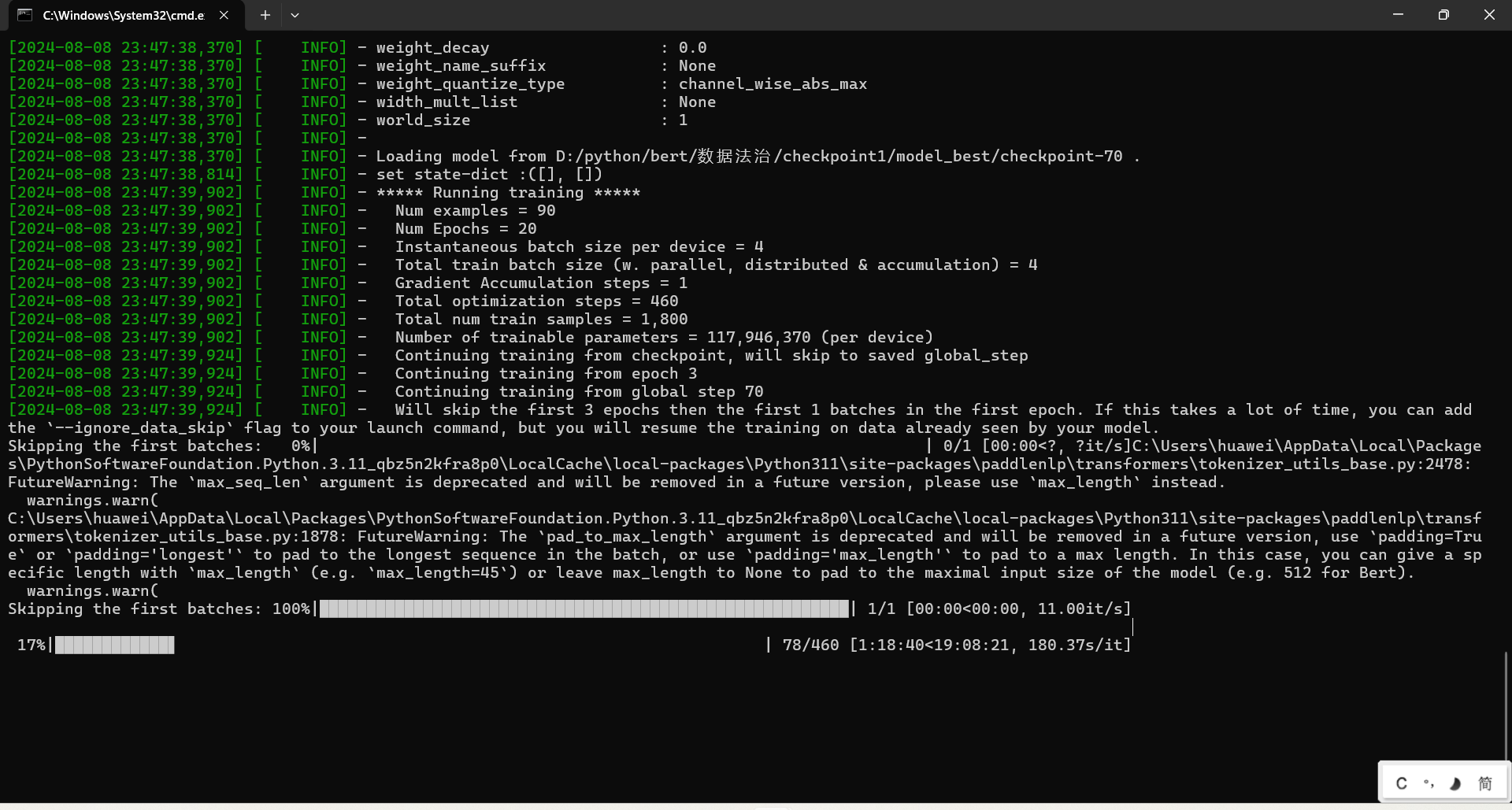


训练页2

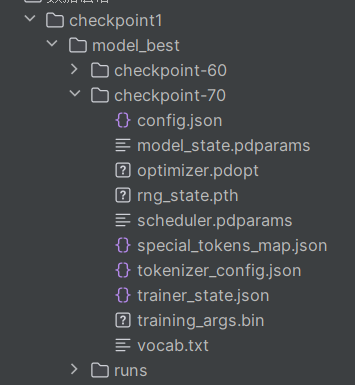
训练页3



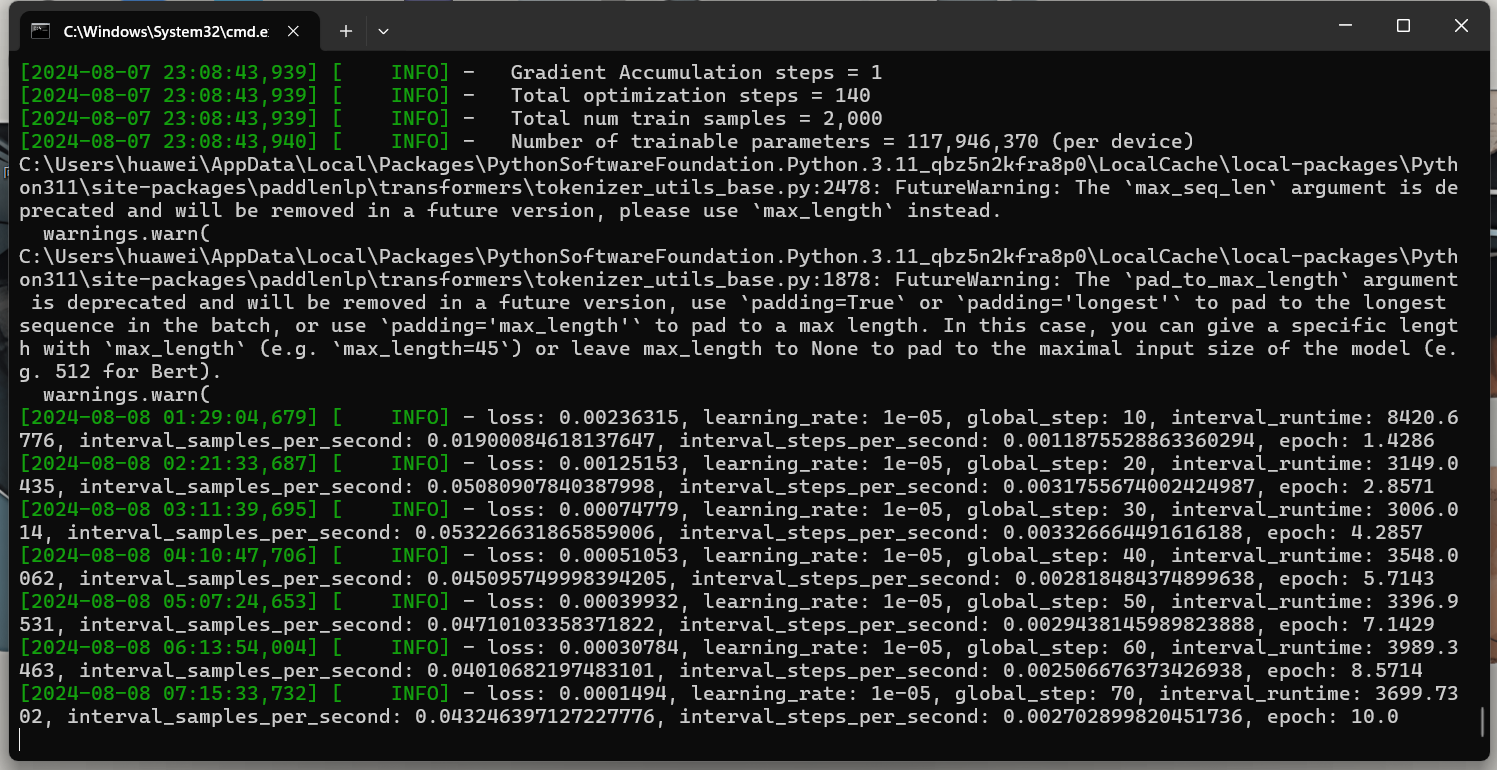
训练页4



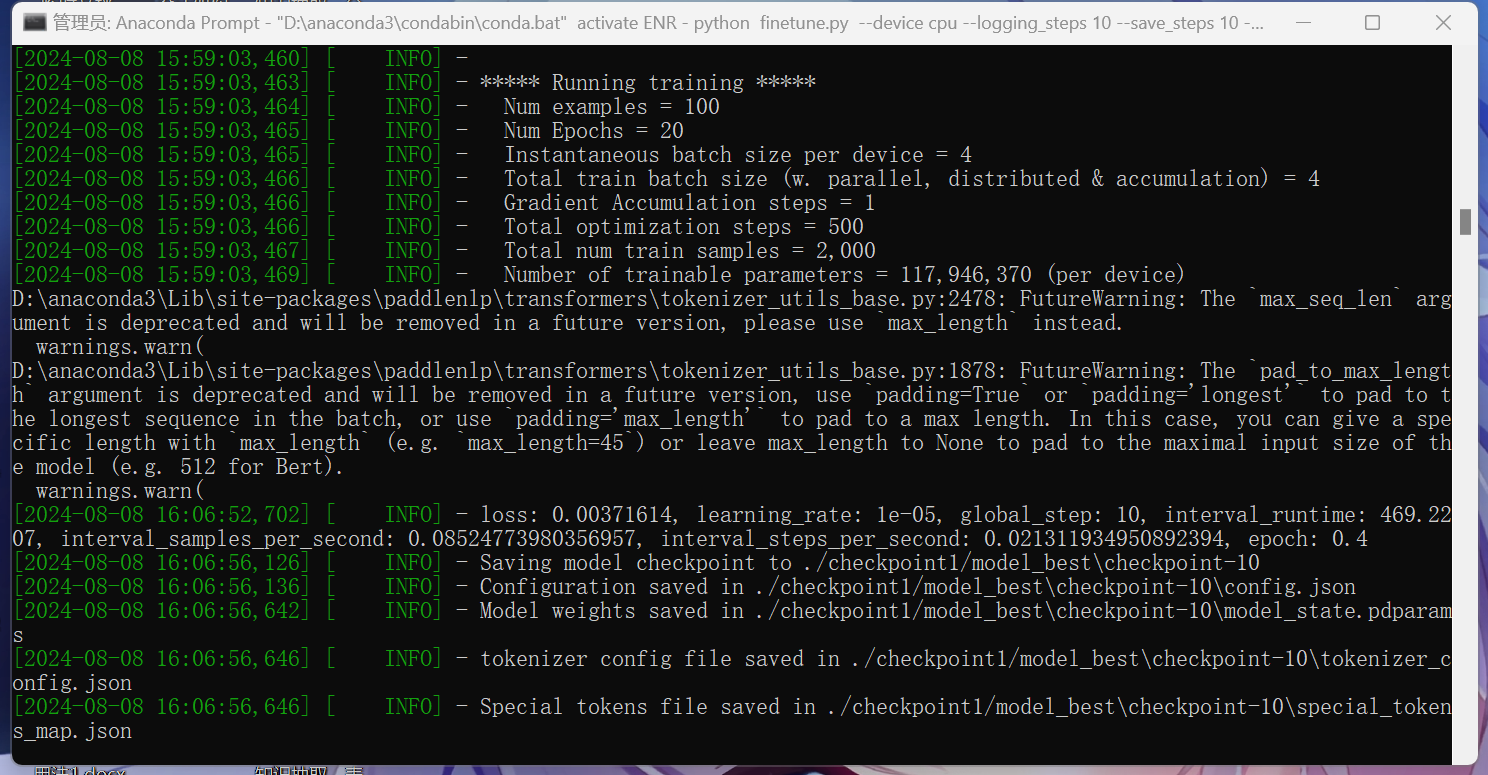
训练页5



训练后模型文件



运行五轮次效果

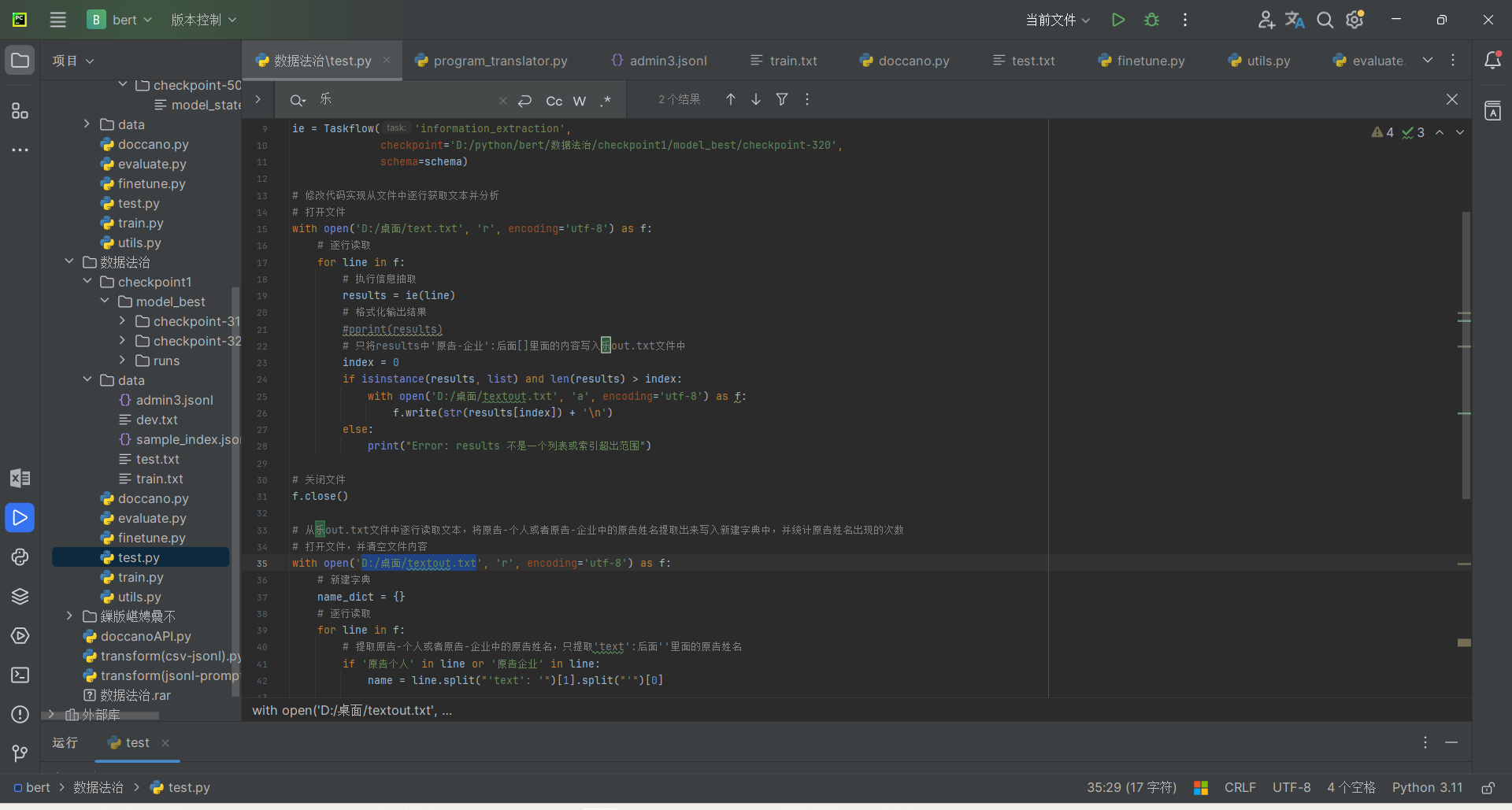
第二次训练模型成功

# 六、实现效果

## 1.test.py代码:

# 单个案例分析  
from pprint import pprint  
from paddlenlp import Taskflow  
  
# 定义schema，分开原告和被告的身份信息  
schema = ['原告-个人', '原告-企业', '被告-个人', '被告-企业']  
  
# 创建信息抽取任务，加载指定的checkpoint  
ie = Taskflow('information\_extraction',  
 checkpoint='D:/python/bert/数据法治/checkpoint1/model\_best/checkpoint-60',  
 schema=schema)  
  
# 修改代码实现从文件中逐行获取文本并分析  
# 打开文件  
with open('D:/桌面/乐.txt', 'r', encoding='utf-8') as f:  
 # 逐行读取  
 for line in f:  
 # 执行信息抽取  
 results = ie(line)  
 # 格式化输出结果  
 pprint(results)  
 # 只将results中'原告-企业':后面[]里面的内容写入乐out.txt文件中  
 index = 0  
 if isinstance(results, list) and len(results) > index:  
 with open('D:/桌面/乐out.txt', 'a', encoding='utf-8') as f:  
 f.write(str(results[index]) + '\n')  
 else:  
 print("Error: results 不是一个列表或索引超出范围")  
  
# 关闭文件  
f.close()  
  
# 从乐out.txt文件中逐行读取文本，将原告-个人或者原告-企业中的原告姓名提取出来写入新建字典中，并统计原告姓名出现的次数  
# 打开文件，并清空文件内容  
with open('D:/桌面/乐out.txt', 'r', encoding='utf-8') as f:  
 # 新建字典  
 name\_dict = {}  
 # 逐行读取  
 for line in f:  
 # 提取原告-个人或者原告-企业中的原告姓名，只提取'text':后面''里面的原告姓名  
 if '原告-个人' in line or '原告-企业' in line:  
 name = line.split("'text': '")[1].split("'")[0]  
  
 # 统计原告姓名出现的次数  
 if name in name\_dict:  
 name\_dict[name] += 1  
 else:  
 name\_dict[name] = 1  
 # 格式化输出结果  
 pprint(name\_dict)  
 #从name\_dict字典中找出出现次数大于10的原告姓名并将姓名输出到屏幕上  
 for key, value in name\_dict.items():  
 if value > 10:  
 print("该人/业为职业打假人/企业:", key)  
# 关闭文件  
f.close()

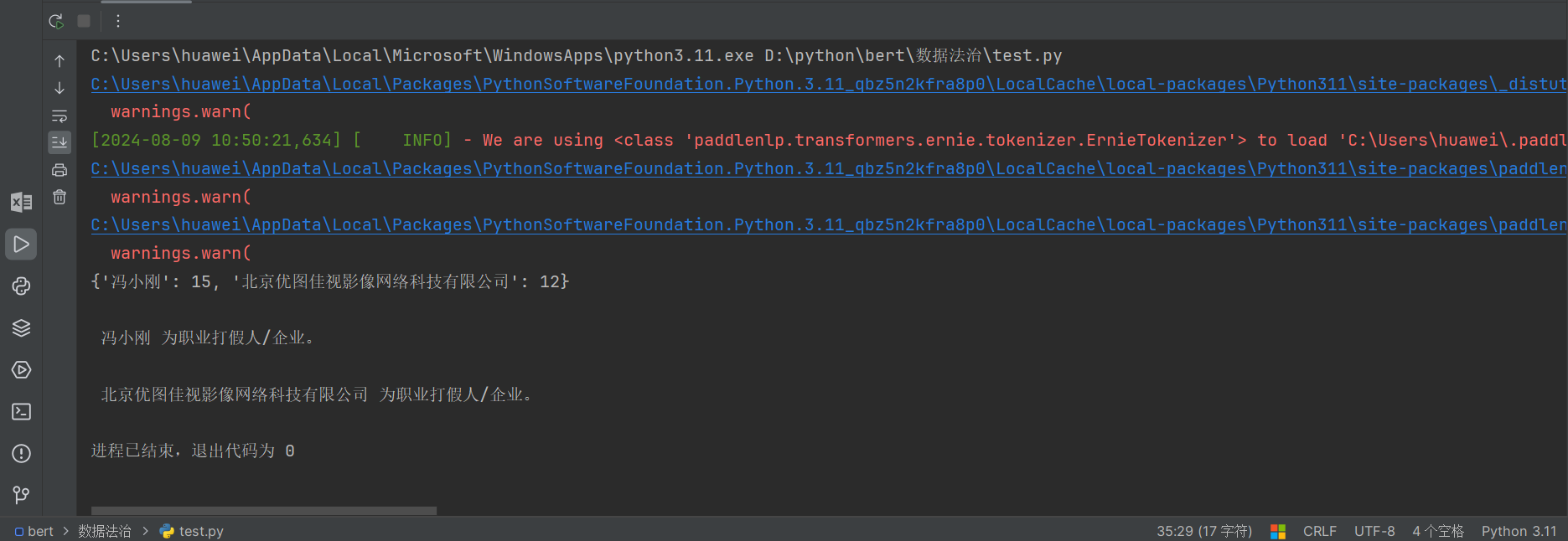
## 2.运行截图：



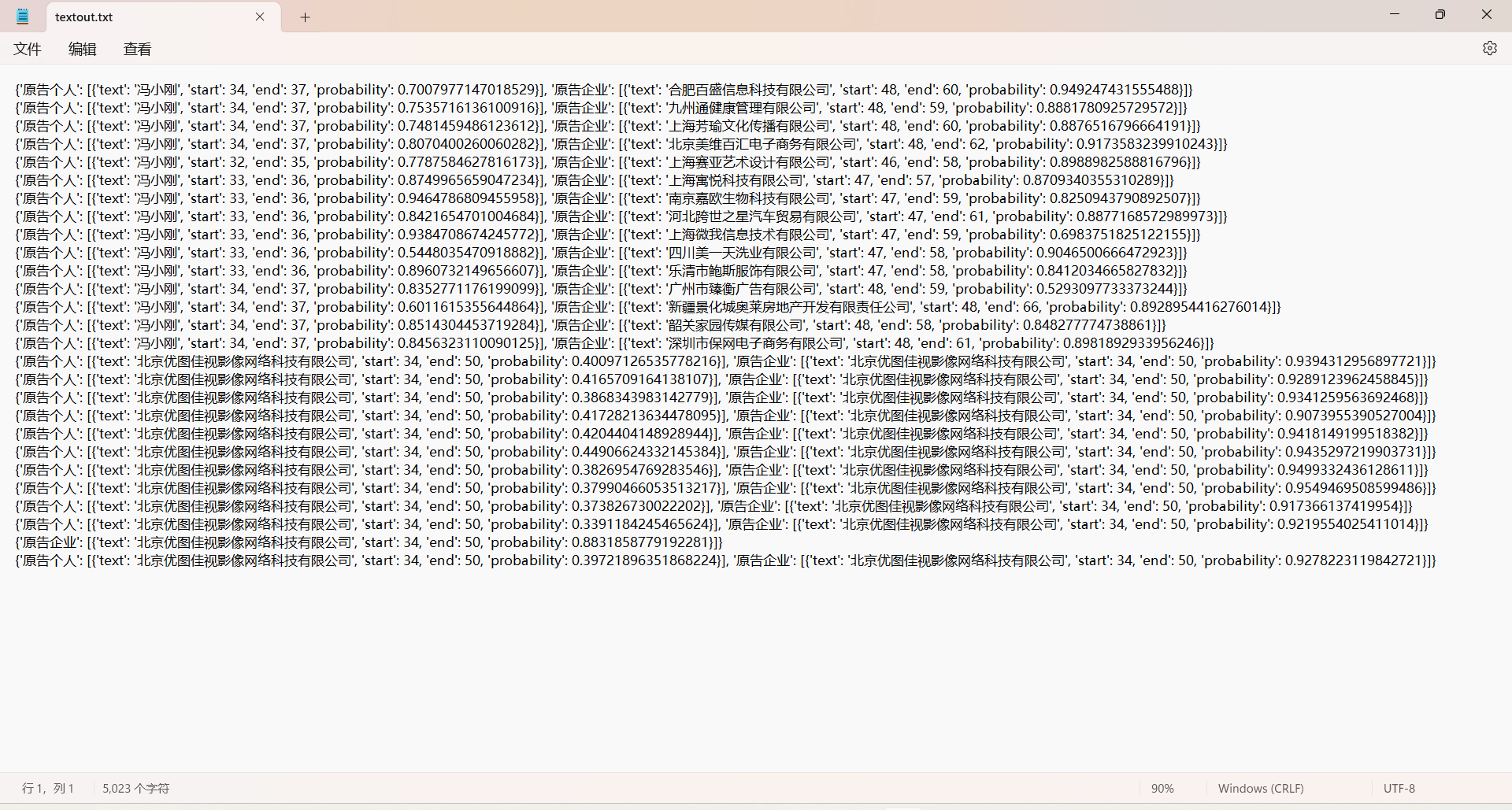
代码页



测试数据



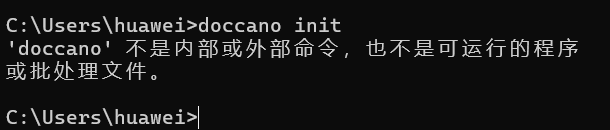
运行结果



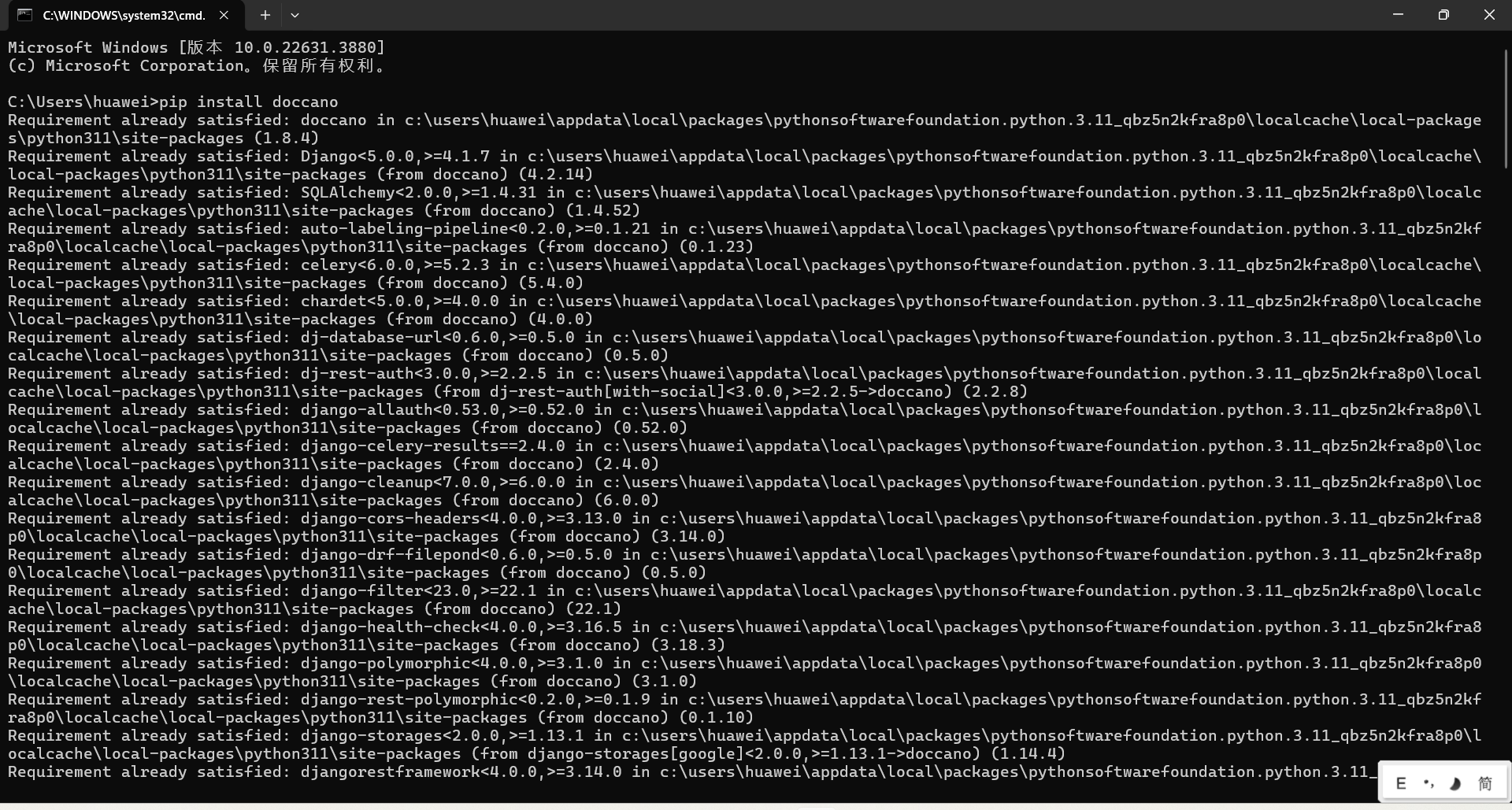
运行中间结果文件

# 七、报错分析

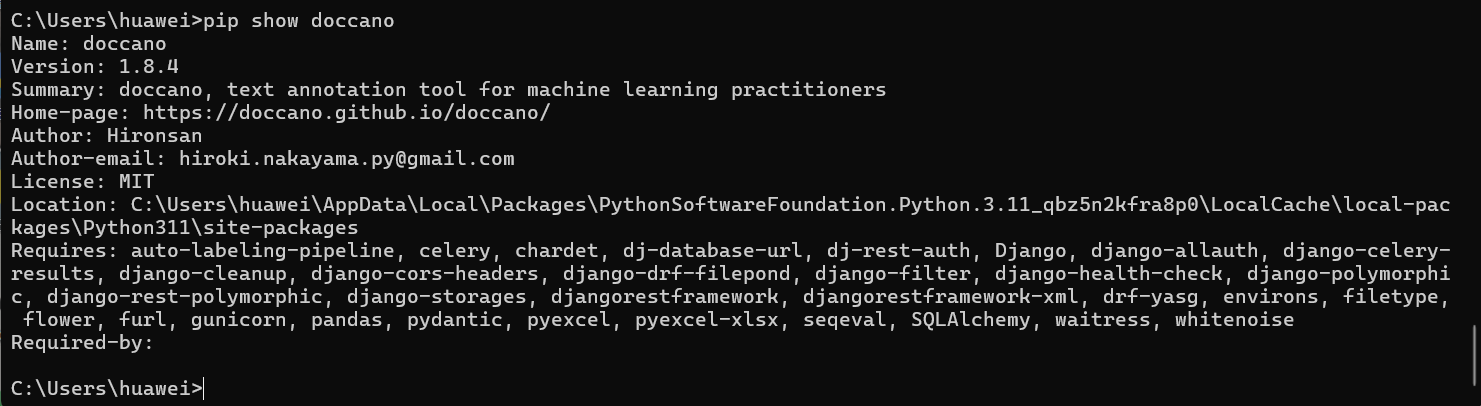
## 1.最初数据标注启动失败



报错信息



下载doccano过程



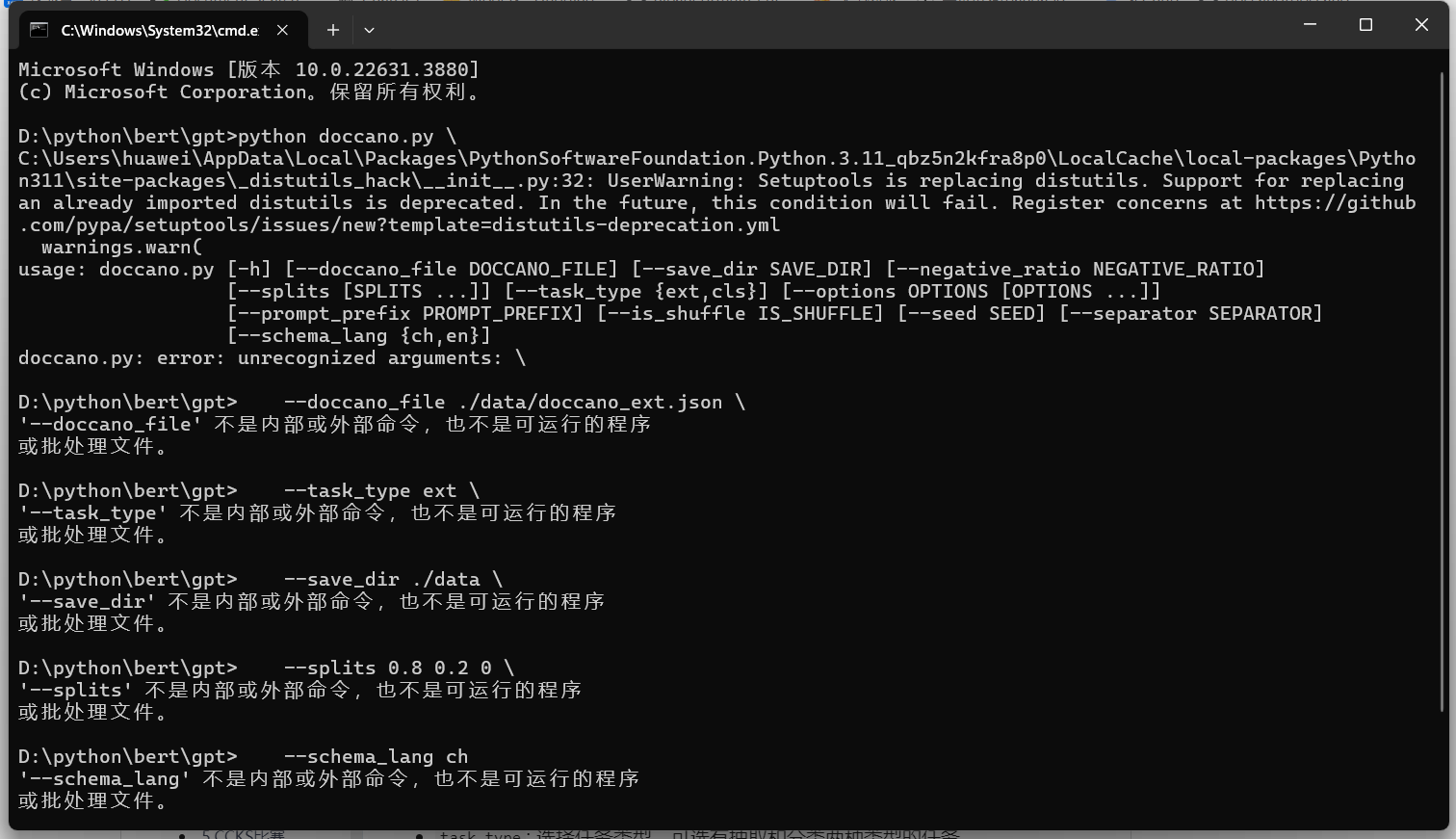
下载doccano成功



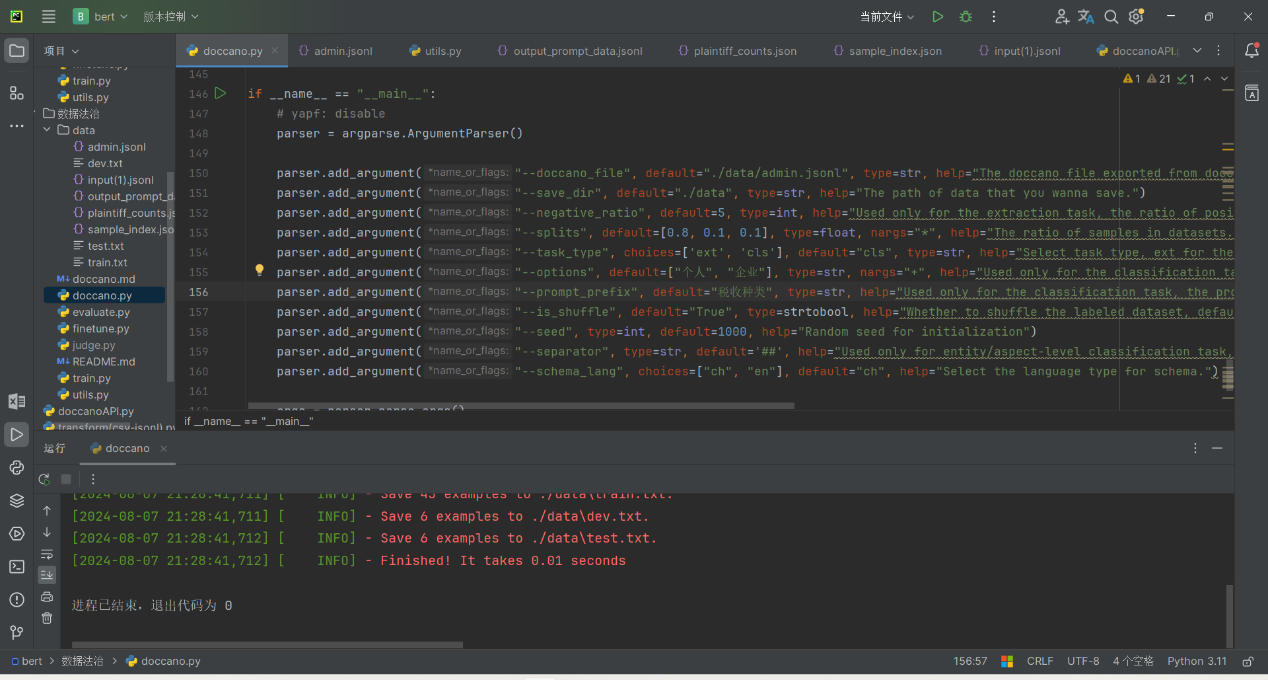
依旧启动doccano失败

解决方式：更换数据标注方式

## 2.微调启动失败



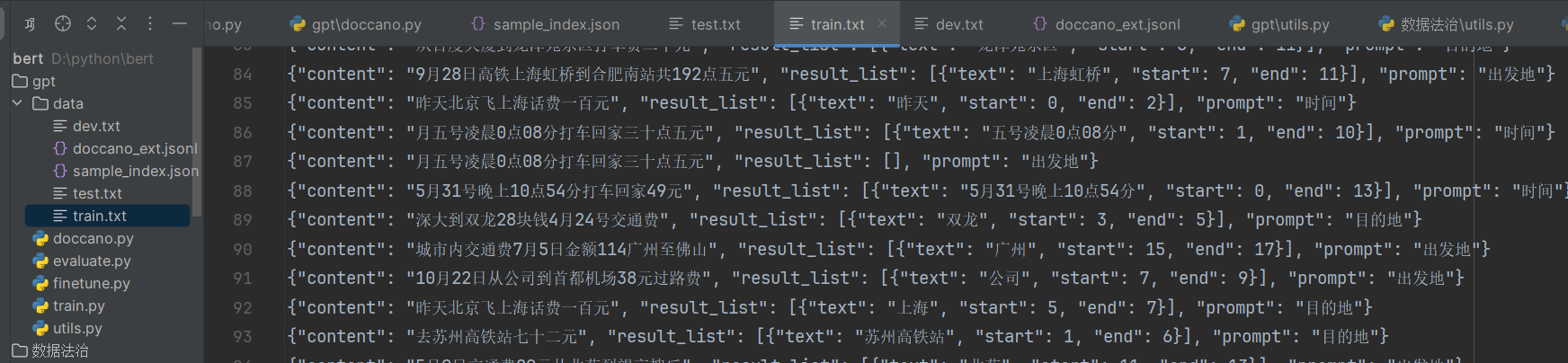
微调启动失败页面



分析启动代码

解决方式，询问有经验的同学，在同学帮助下成功启动

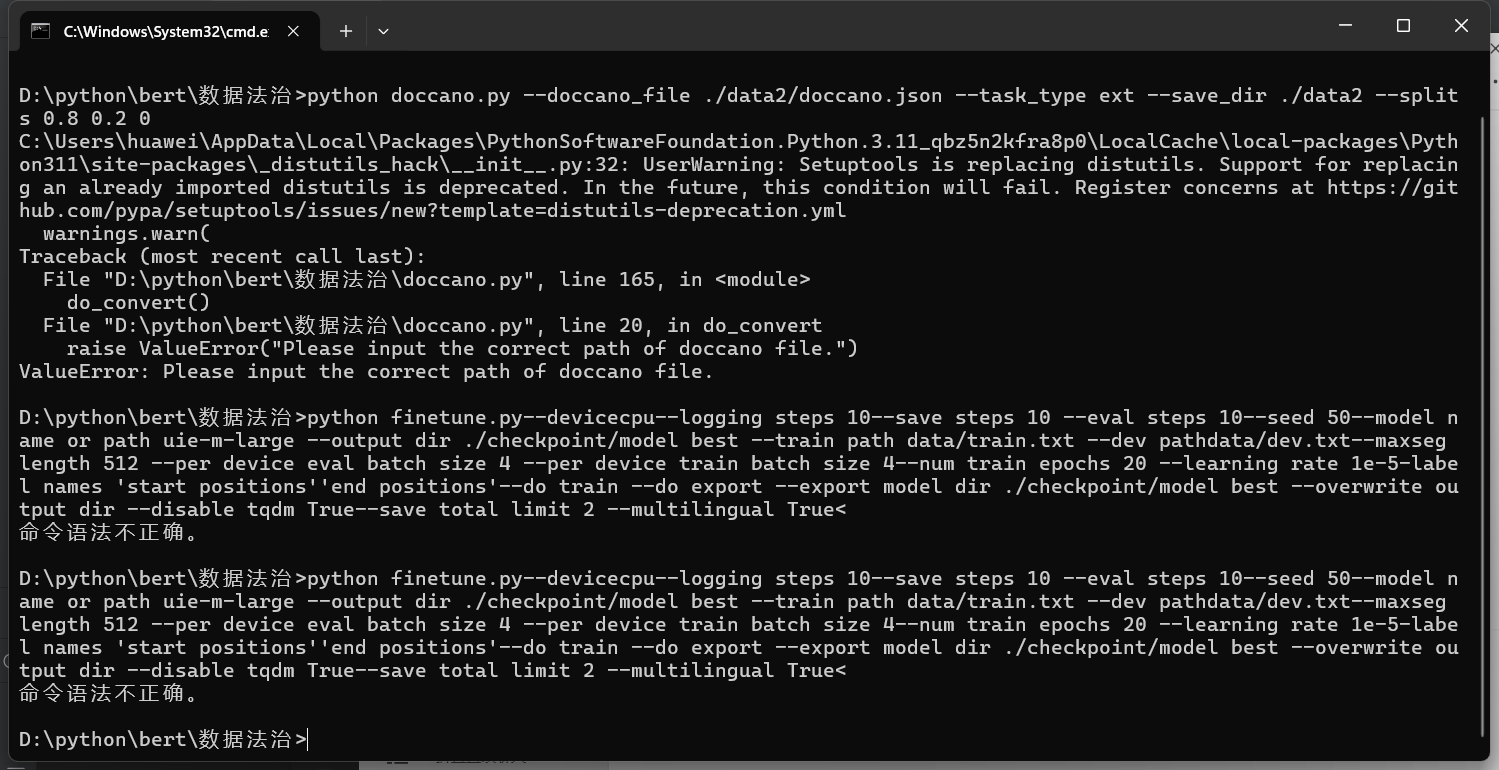
## 3.数据集格式错误



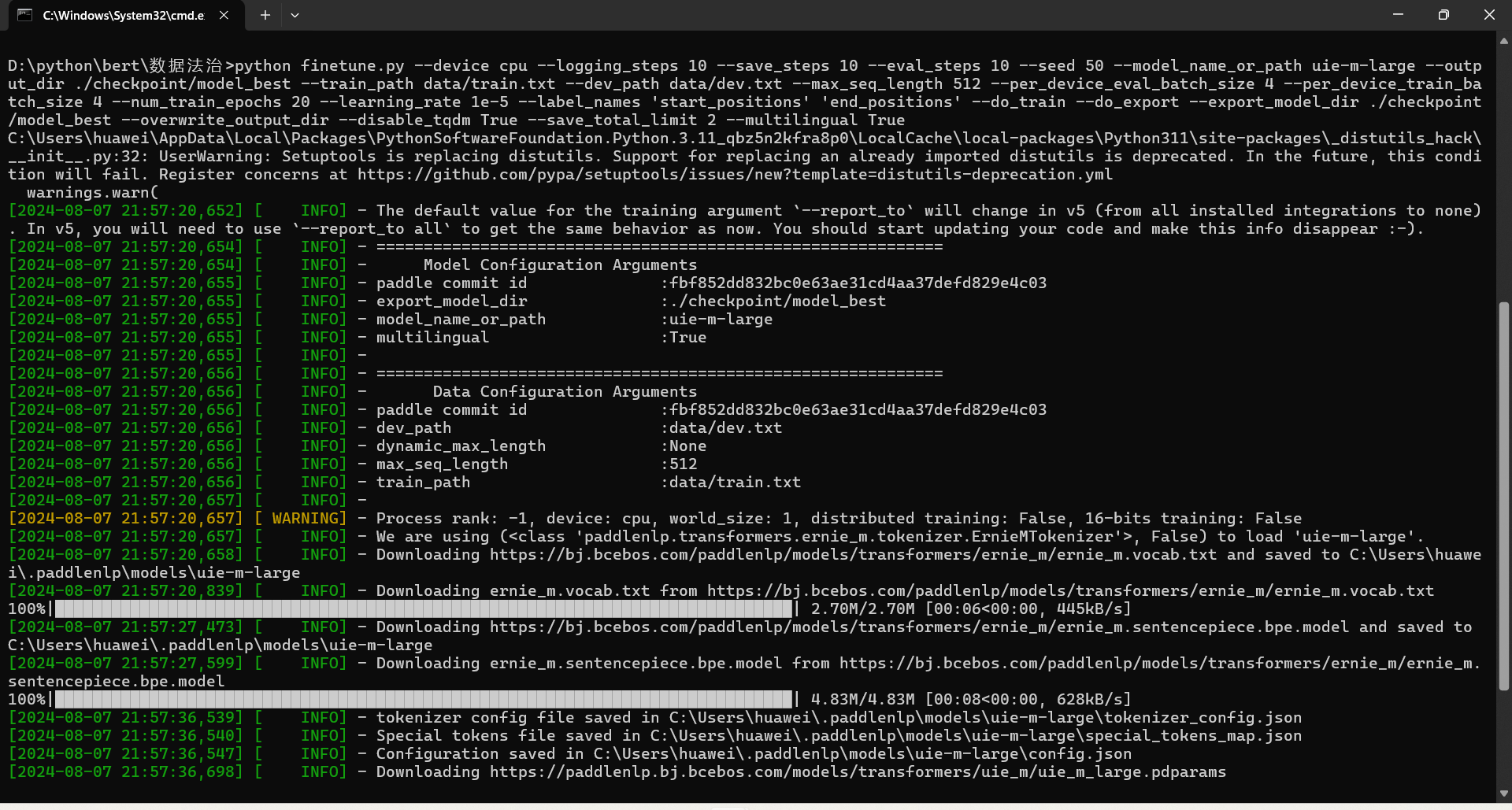
数据集

解决方式：多次尝试不同格式后，成功运行程序

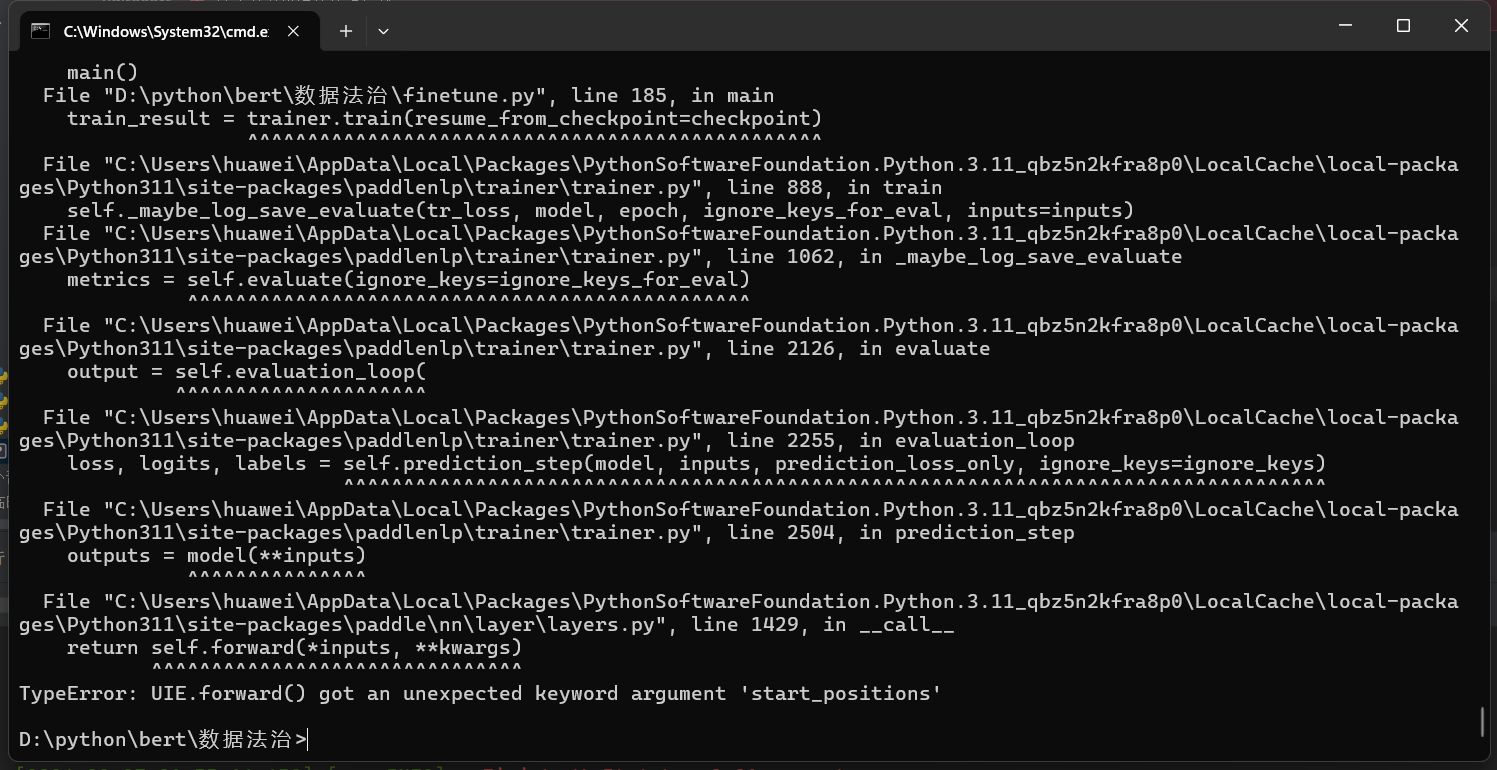
## 4.微调参数错误



错误页面



成功一半



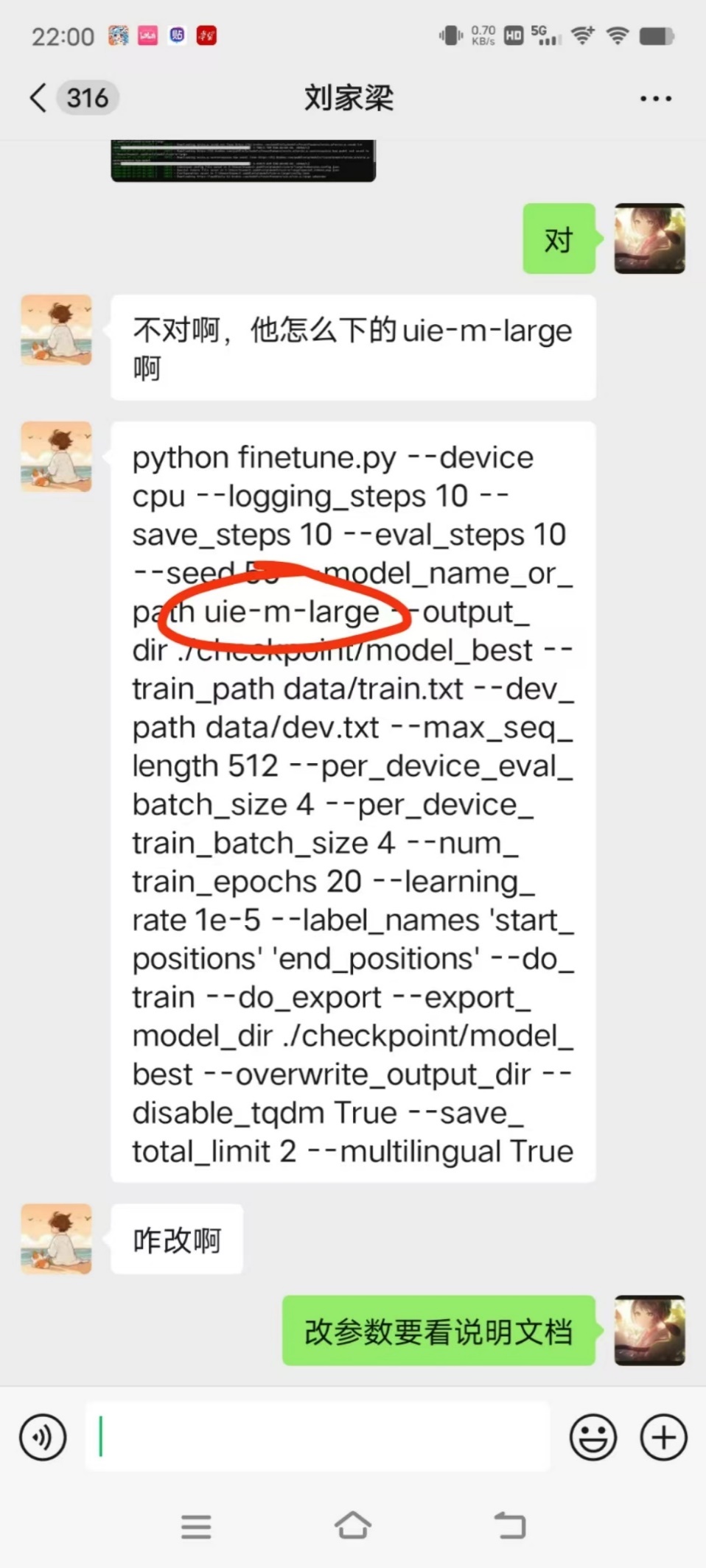
参数错误无法运行



参数错误无法运行\*2



参数错误无法运行\*3



询问同学



询问chatgpt

解决方式：失败多次，观阅uie官方文档、询问chatgpt并在同学帮助下成功运行

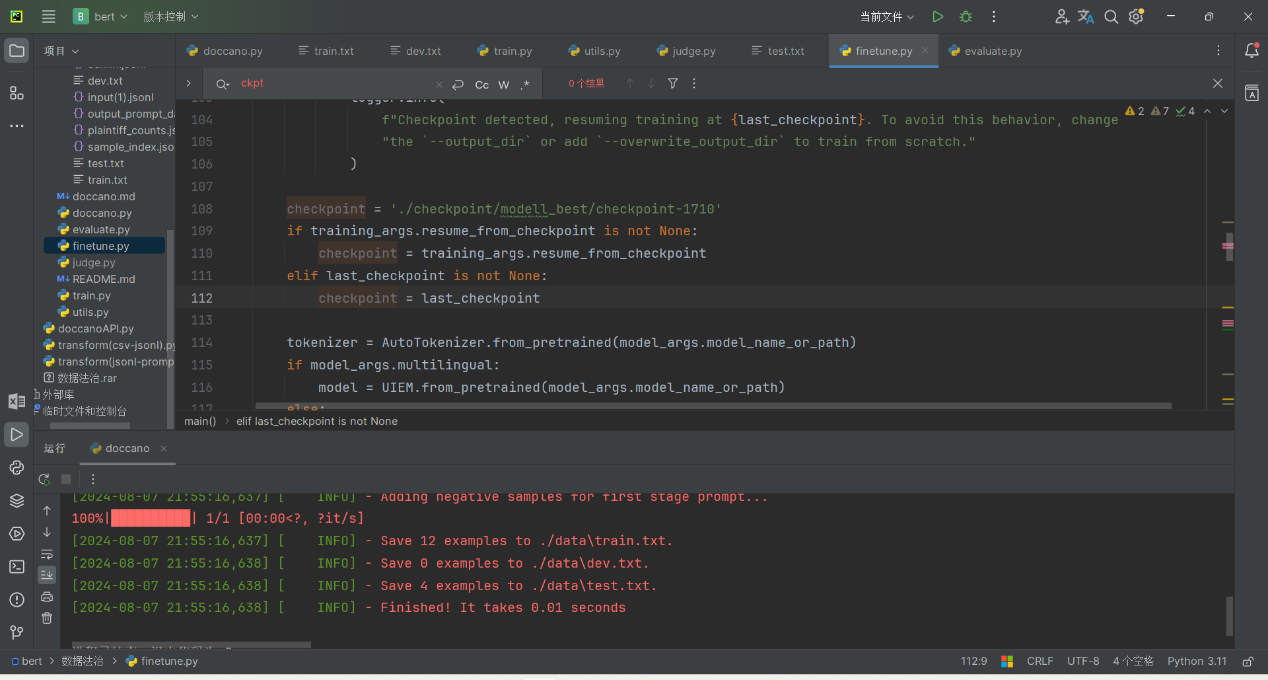
## 5.选错数据标注类型



错误类型

解决方式：失败一次后重新选择类型标注，最终成功

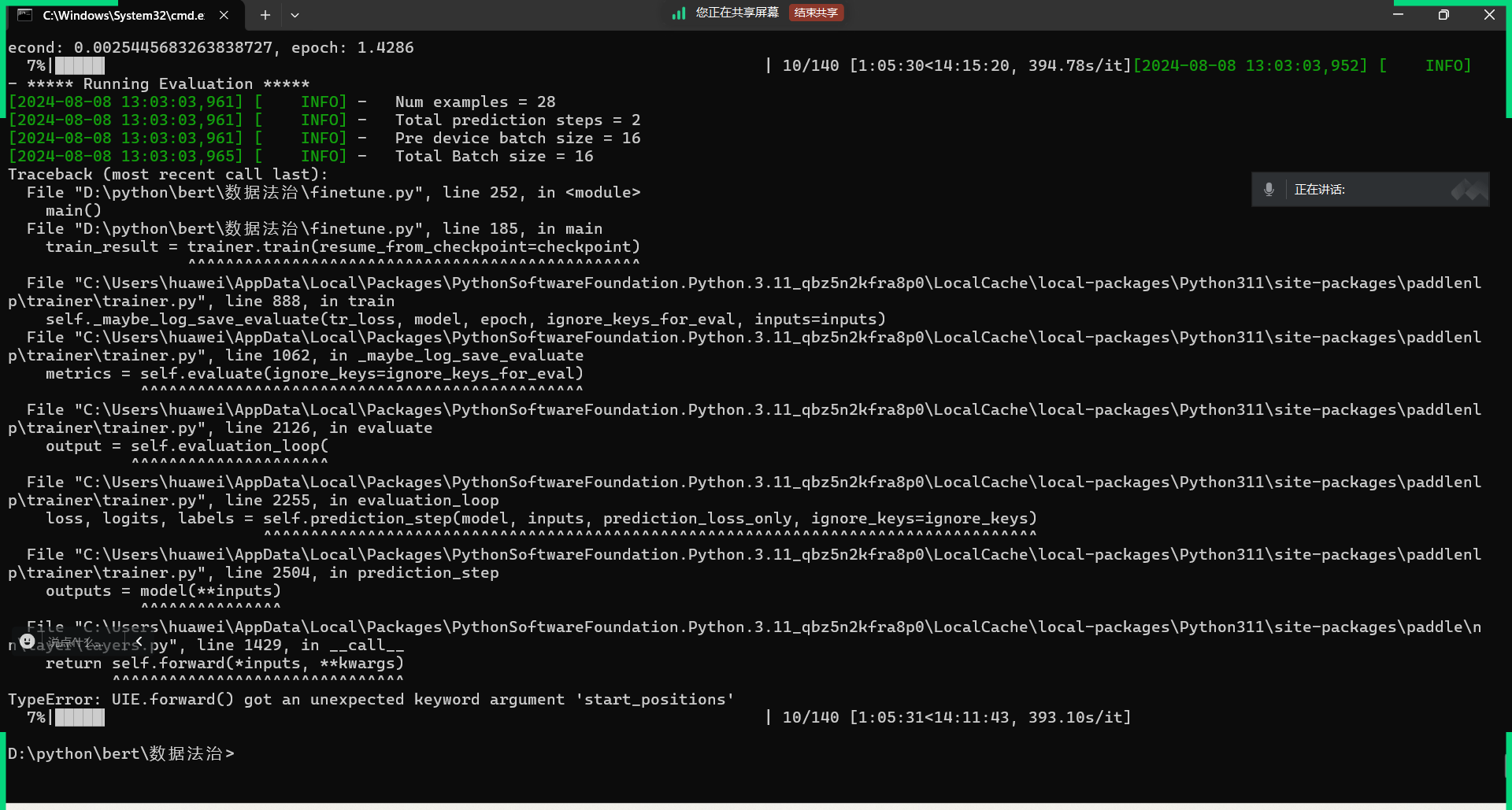
## 6.运行模型未保存



发现模型内地址忘记修改

解决方式：白白训练12小时后修改路径重新训练

## 7.运行一半模型自动停止运行



自动停止页

解决方式：参数设置有误，修改参数重新开始