

知识表示项目报告

LLM知识图谱

黄子馨，陈思远

032130208，162140117

2024年6月22日

目录

- 1. 项目简介
- 2. 训练数据收集、标注
 - 2.1 数据来源
 - 2.1.1 大模型榜单
 - 2.1.2 维基百科
 - 2.1.3 谷歌浏览器
 - 2.1.4 Arxiv平台论文
 - 2.2 数据标注
 - 2.2.1 命名实体识别数据标注
 - 2.2.2 关系抽取数据标注
- 3. 模型训练
 - 3.1 命名实体识别模型
 - 3.2 关系抽取
- 4. 知识图谱构建
 - 4.1 从论文中获取三元组
 - 4.2 构建图模型
 - 4.3 可视化知识图谱
- 5. 知识图谱嵌入
 - 5.1 向量表示
 - 5.1.1 模型架构
 - 5.1.2 关键代码说明
 - 5.2 知识推理
 - 5.2.1 实现细节
 - 5.2.2 实验结果
 - 5.2.3 未来工作
- 6. 问题分析
 - 6.1 为什么模型实体识别效果差
 - 6.2 Llama3为什么是OpenAI开发的

分工:

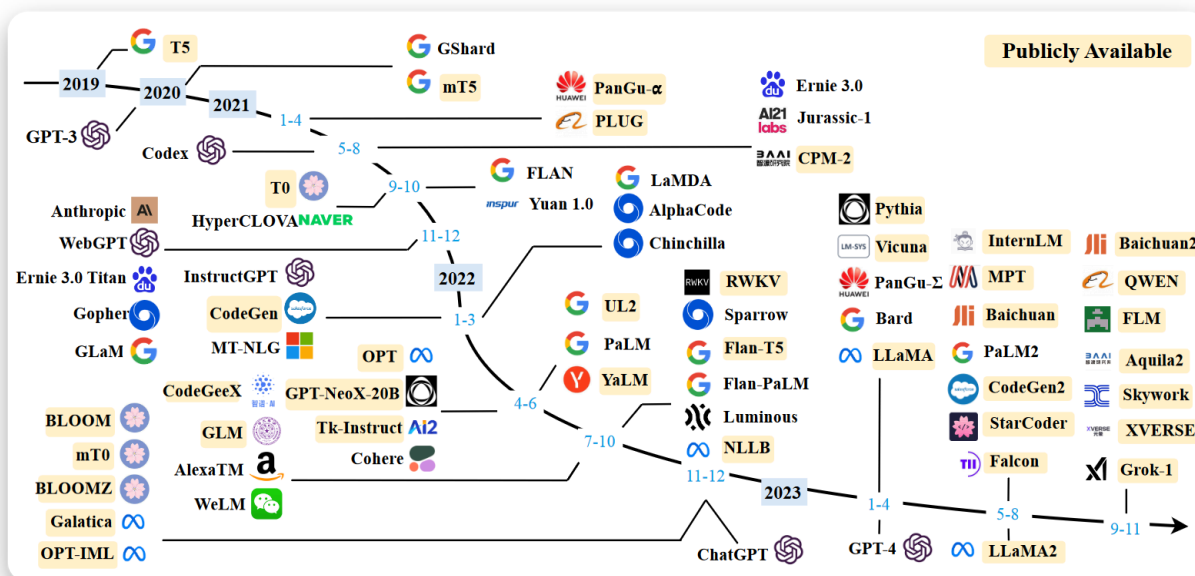
- 陈思远: 训练数据收集及标注, 模型训练, 知识图谱构建
- 黄子馨: 知识图谱嵌入, 向量表示及知识推理

报告使用markdown+html+css撰写。

1. 项目简介

知识图谱（Knowledge Graph）是一种用于组织和表示知识的方法，通常以图的形式非常直观地展示实体（如人、地点、事物）及其相互关系。通过知识图谱，我们能够将信息以一种结构化的形式表示，使得计算机在图的基础和层面上理解这些信息并实现推理。

大语言模型（Large Language Models）是基于深度学习的自然语言处理模型。自在2022年底OpenAI公开发布ChatGPT，实现大语言模型研究“商业化”落地以来，越来越多的公司或研究机构发布了或自研、或套壳的模型。这些模型拥有不同的基础架构（如BERT类利用Transformer编码器，而GPT类利用解码器）、不同的参数量（从1B到上千B）、不同的处理任务（如代码生成、文本生成、摘要生成）、不同的模型使用效果，他们之间可能使用相同的技术路线，可能是由上一代模型迭代改进而来，可能是由相同的公司或研究机构提出发布……



摘自“A Survey of Large Language Models”

在这样的背景下，我们希望构建一个大语言模型知识图谱，用于汇总近些年来大模型的相关信息，并以图的形式展示他们之间的关系（而不是像榜单那样的列表形式）。目前我们构建的知识图谱主要：

- 以模型、公司（研究机构）、任务等作为主要实体
- 以（模型，被开发，公司）、（模型，用于，任务）、（公司，位于，地址）等作为主要关系
- 以[参数，发布时间，是否开源]等作为模型实体的属性，以[创建人员，成立时间]等作为公司实体的属性

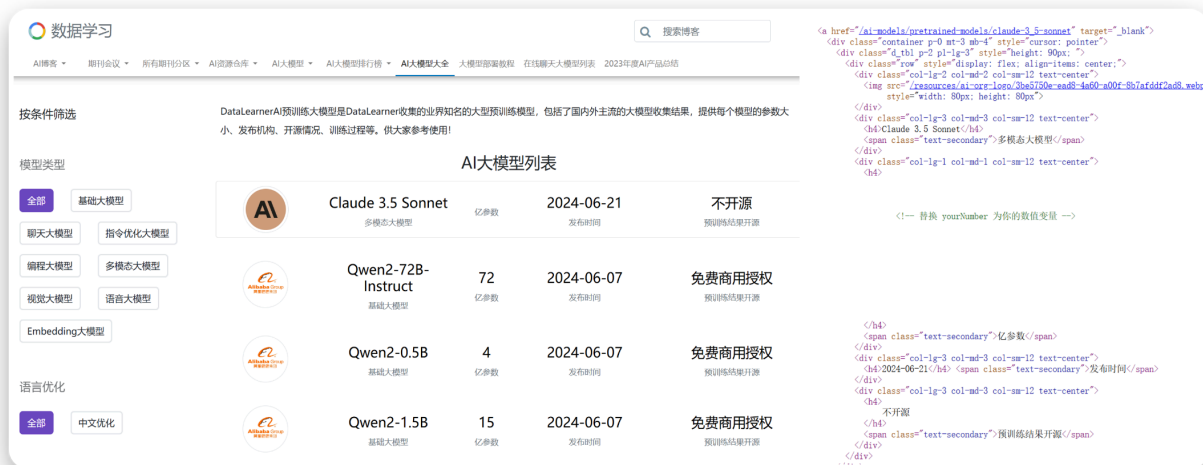
2. 训练数据收集、标注

为了不断从网络信息中自动捕获所需信息，我们希望训练模型用于命名实体识别（自动捕获模型、公司等实体）、关系抽取（自动抽取实体之间的关系）。而为了训练这些模型，需要收集大量的数据并进行标注。目前我们主要是收集结构化数据、半结构化数据，然后再进一步处理得到训练数据。**本项目中爬取数据代码均为我们编写**，以下我们将简要介绍数据来源及处理方法。

2.1 数据来源

2.1.1 大模型榜单

为了对比大模型的性能，网络上存在大量的大模型榜单，其中包含了近些年来发布的众多模型，同时有些榜单也附加了这些模型的基本属性。我们通过爬虫抓取了这些网页的源代码，通过处理，便能够得到拥有基础属性的模型实体。



The screenshot shows the 'AI大模型列表' (AI Large Model List) on the DataLearner website. The table lists several models with their parameters, release dates, and licensing. The models shown are Claude 3.5 Sonnet, Qwen2-72B-Instruct, Qwen2-0.5B, and Qwen2-1.5B.

| 模型名称 | 参数规模 | 发布时间 | 授权方式 |
|--------------------|--------|------------|--------|
| Claude 3.5 Sonnet | 亿参数 | 2024-06-21 | 不开源 |
| Qwen2-72B-Instruct | 72 亿参数 | 2024-06-07 | 免费商用授权 |
| Qwen2-0.5B | 4 亿参数 | 2024-06-07 | 免费商用授权 |
| Qwen2-1.5B | 15 亿参数 | 2024-06-07 | 免费商用授权 |

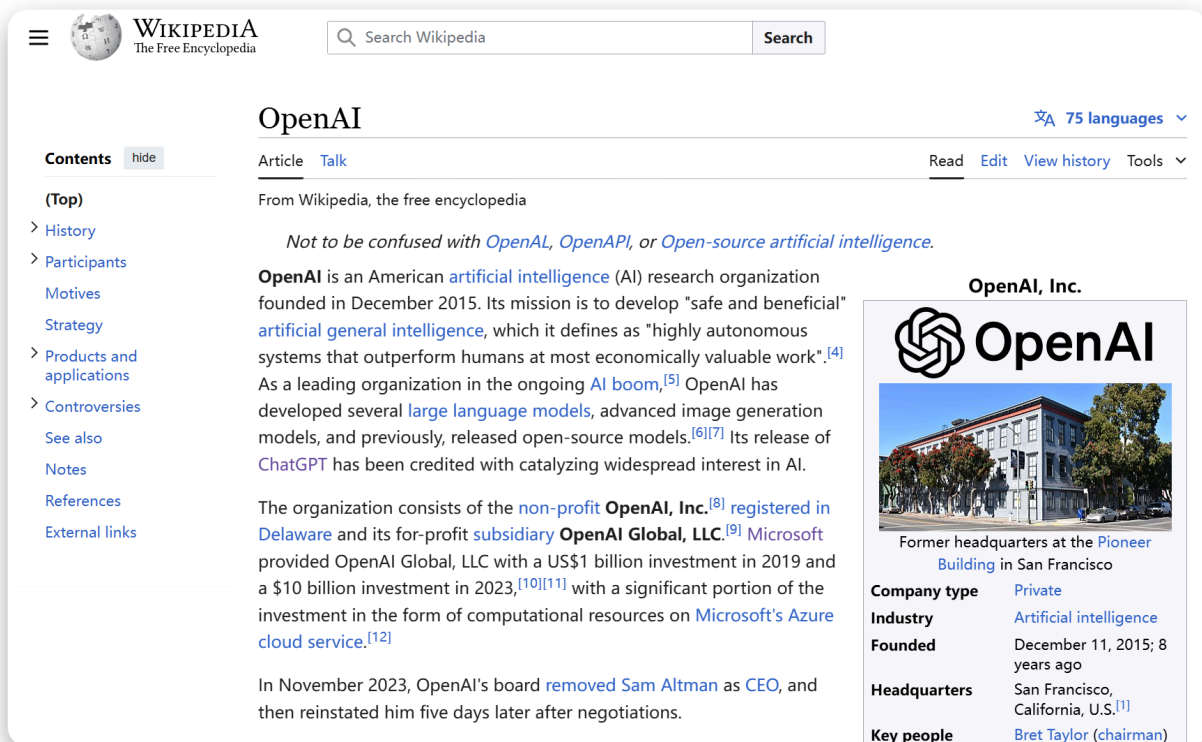
使用正则表达式清洗数据后得到数据，保存为json格式存储

```
1 a_tags_pattern = re.compile(r'<a href=[\s\S]*?>', re.IGNORECASE)
2 model_pattern = re.compile(
3     r'<div class="col-lg-3 col-md-3 col-sm-12 text-center">\s*<h4>(.*?)'
4     r'</h4>.*?<span class="text-secondary">(.*?)</span>.*?'
5     r'<!-- 数值为整数，不显示小数 -->\s*(.*?)\s*</h4>.*?<h4>(.*?)</h4> '
6     r'<span class="text-secondary">发布时间</span>.*?<h4>\s*(.*?)\s*'
7     r'</h4>\s*<span class="text-secondary">(.*?)</span>',
8     re.DOTALL)
```

```
"Codestral": {
  "type": "编程大模型",
  "size": "220",
  "date": "2024-05-29",
  "copyright": "不可以商用",
  "opensource": "预训练结果开源"
}
...
```

2.1.2 维基百科

注意到上述榜单中并不能获得发布这些模型的机构（网页源代码中机构部分均为无有效信息的图像连接），因此通过自主收集并使用chatgpt生成筛选了目前主流公司或研究机构的名称。通过调用维基百科的api，查询这些公司的维基百科主页，并抓取这些公司的简介文本，用于训练文本数据。

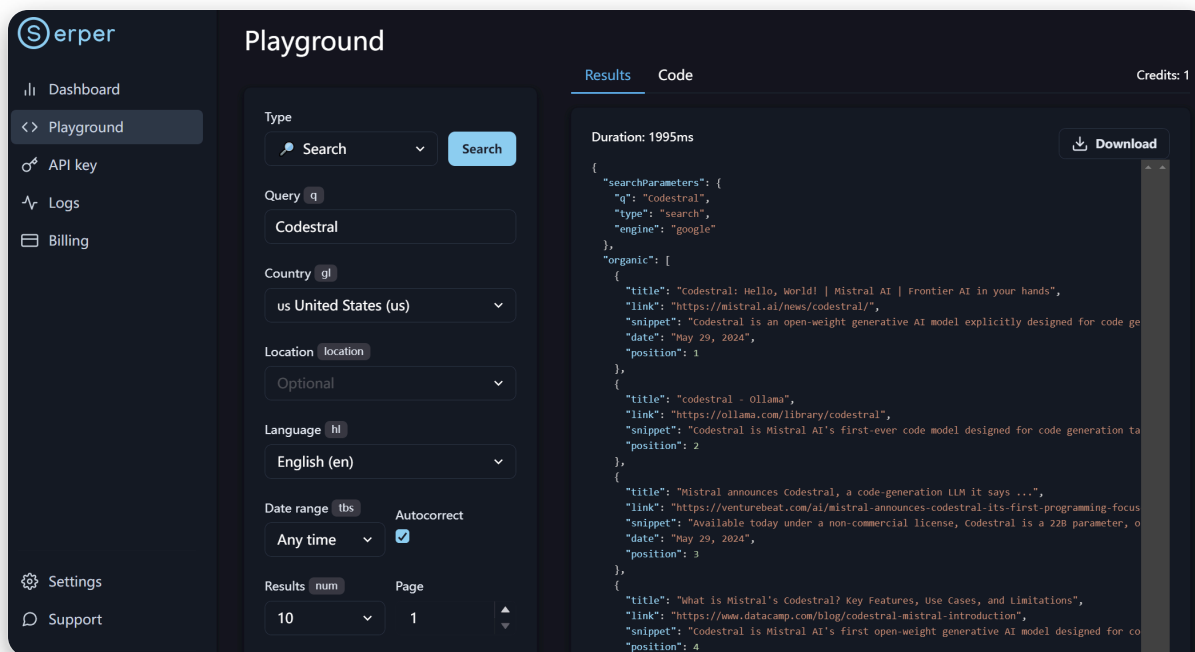


The image shows a screenshot of the Wikipedia article for OpenAI. The page title is "OpenAI" and it includes a search bar, navigation links, and a table of contents. The main text describes OpenAI as an American artificial intelligence (AI) research organization founded in December 2015. It mentions its mission to develop "safe and beneficial" artificial general intelligence and lists its products and applications. A sidebar on the right provides information about OpenAI, Inc., including its company type (Private), industry (Artificial intelligence), founding date (December 11, 2015), and headquarters (San Francisco, California, U.S.).

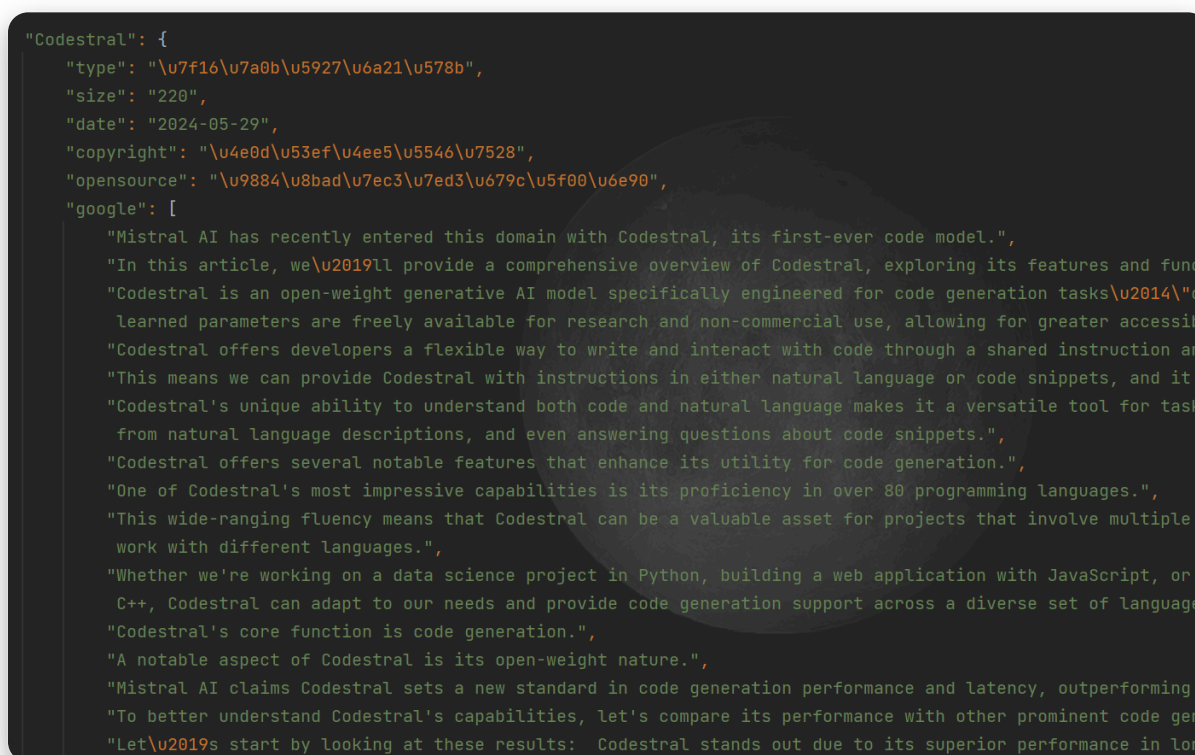
| Company type | Private |
|--------------|--|
| Industry | Artificial intelligence |
| Founded | December 11, 2015; 8 years ago |
| Headquarters | San Francisco, California, U.S. ^[1] |
| Key people | Bret Taylor (chairman) |

2.1.3 谷歌浏览器

通过大模型榜单获取的大模型缺乏包含这些大模型的语料，因此希望从网页中抓取这些模型相关文本。利用serper平台提供的谷歌搜索接口，通过 requests 库抓取谷歌搜索返回的top 10搜索结果网址，再利用 requests 和 BeautifulSoup 提取这个网址对应网页的文本。



通过Python程序自动获取到这些文本后，利用 `nltk.tokenize.sent_tokenize` 将这些长文本切割为句子，作为每个模型的语料。



2.1.4 Arxiv平台论文

在训练完命名实体识别、关系抽取的模型后，需要大量未标注文本，进一步提取模型、公司、任务等实体及它们之间的关系。项目的初衷是能够实时地、自动地捕获新模型、新公司、新任务，

对知识图谱进行更新，因此不能完全依靠网页内容（网页文本大部分不是一手内容，而二手内容，如论文解读、报道等，缺乏实时性、准确性）。针对大模型研究的特点，在模型发布时，往往会有相应的论文发表在arxiv平台上，因此我们从arxiv上自动爬取LLMs相关的论文，将论文中的文本输入模型，进行命名实体识别与关系分类，再将利用结果更新知识图谱。

具体来说，

- 首先通过 `requests` 连接到arxiv平台，根据输入的查询词和其他查询条件（例如年份），获取查询结果
- 再使用 `feedparser` 对查询结果进行解析，获得每篇论文的连接
- 利用 `requests.get()` 从指定网址下载pdf写入本地文件，重复下载完成相关论文的爬取
- 使用 `pdfplumber` 对pdf进行读取，获取文章中的文本信息

2.2 数据标注

通过初步的数据抓取，目前主要获得了如下两种形式的原始数据。

```
"OpenAI": [  
  "Generative Pre-trained Transformer 3 (GPT-3) is a large language model relea...",  
  "On June 11, 2018, OpenAI researchers and engineers published a paper introdu...",  
  "Text generated by Mike Sharples[11] On May 28, 2020, an arXiv preprint by a ...",  
  "OpenAI has implemented several strategies to limit the amount of toxic langu...",  
  ...  
]
```

```
"Codestral": {  
  "type": "编程大模型",  
  "size": "220",  
  "date": "2024-05-29",  
  "copyright": "不可以商用",  
  "opensource": "预训练结果开源",  
  "google": [  
    "Mistral AI has recently entered this domain with Codestral...",  
    "In this article, we'll provide a comprehensive overview of Codestral...",  
  ]  
}
```

2.2.1 命名实体识别数据标注

在上一节中已经通过各种途径获取到了各种模型的名称、公司或研究机构的名称，以及大量的文本（已经经过初步筛选）。获取命名实体识别所需的标注数据，只需在这些文本中找到对应的名称，进行匹配即可。

在实际情况下，预先获得的模型标准名称与文本中的名称不可能完全相同，例如，gpt-3.5, gpt 3.5我, GPT3.5实际指的为同一模型。因此不能简单地通过字符串匹配来标注文本中的实体名称。我们使用 `Levenshtein` 库提供的方法测量文本中词与名称之间的**编辑距离**，若编辑距离小于设置的阈值，则认为词与名称相同。

```
def is_similar(str1, str2, threshold=0.8):
    distance = Levenshtein.distance(str1, str2)
    similarity = 1 - (distance / max(len(str1), len(str2)))
    return similarity >= threshold

is_similar(candidate.lower(), ' '.join(words[i:i + len(candidate_words)]).lower())
```

我们使用BIO (Begin, Inside, Outside) 标注形式，包含模型、公司、任务等实体。标注数据如下：

```
Developed      0
by             0
Alibaba        B-ORG
Cloud,         0
Qwen2          B-MOD
is             0
the            0
next           B-TASK
generation     I-TASK
of             0
the            0
```

2.2.2 关系抽取数据标注

我们注意到在之前收集到的原始数据中，包含了大量同时含有模型-公司、模型-任务的句子，同时，这些句子中同时存在的实体，通常满足实际关系，例如（模型，被开发，公司）。因此我们

采用远程监督的思想，若两个实体同时存在于一个句子中，则任务它们满足预设的关系。例如，如果gpt2和code generation同时存在于一个句子中，我们就认为二者满足（模型，用于，任务）的关系。

此外，为降低错误样本对模型的影响，我们通过chatgpt生成了上百条包含实体和实际关系的句子，加入到训练集中（也可以用于微调）。标注数据如下：

```
{"text": "The ChatGLM model can be called to start a conversation using the following code: The above code will automatically download the model implementation and parameters by transformers.", "labels": ["transformers.", "conversation", "developed_for"]}  
{"text": "The following table provides a comprehensive comparison of WizardLMs and several other LLMs on the code generation task, namely HumanEval.", "labels": ["WizardLMs", "code generation", "developed_for"]}  
{"text": "So Huawei is challenging Apple directly in the same price categories.", "labels": ["same", "Apple", "developed_by"]}
```

3. 模型训练

3.1 命名实体识别模型

命名实体识别部分采用了BERT + BiLSTM + CRF，模型的处理流程如下

- 向文本中插入特殊符号，并将文本转为输入id序列，**分词器**采用预训练模型bert-base-uncased提供的tokenizer
- 将id序列输入BERT模型，获取最后一个隐藏层的输出向量
- 将BERT输出向量传入BiLSTM，将输出向量调整后通过一层全连接层
- 通过CRF层解码，得到最终模型输出

使用Trainer进行封装训练，利用CRF层的负对数似然损失函数来训练模型。在训练过程中，优化器将最小化这个损失函数，从而调整模型的参数，使得模型的预测性能逐渐提高。

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| MOD | 0.73 | 0.25 | 0.37 | 708 |
| ORG | 0.95 | 0.89 | 0.92 | 2428 |
| TASK | 0.84 | 0.65 | 0.73 | 57 |
| micro avg | 0.93 | 0.74 | 0.83 | 3193 |
| macro avg | 0.84 | 0.60 | 0.67 | 3193 |
| weighted avg | 0.90 | 0.74 | 0.79 | 3193 |

BERT + BiLSTM + CRF被封装入BertNer类，整个模型被封装入NERPredictor类，能够实现输入文本，直接输出命名实体识别结果。

3.2 关系抽取

关系抽取部分直接使用BERT，将关系抽取任务看作多分类任务。获得BERT的最后一个隐藏层的输出向量后，通过一层全连接层，再通过softmax得到最终分类结果。

同样使用Trainer进行封装训练，利用交叉熵损失函数计算损失更新模型。

| | precision | recall | f1-score | support |
|---------------|-----------|--------|----------|---------|
| developed_for | 1.00 | 1.00 | 1.00 | 7 |
| developed_by | 1.00 | 1.00 | 1.00 | 13 |
| accuracy | | | 1.00 | 20 |
| macro avg | 1.00 | 1.00 | 1.00 | 20 |
| weighted avg | 1.00 | 1.00 | 1.00 | 20 |

最后对整个模型进行封装，实现输入（文本，实体对），直接输出预测关系。

4. 知识图谱构建

4.1 从论文中获取三元组

利用'Large language model'、模型名称等查询词，从arxiv平台上爬取了100篇相关论文，根据前述的方法，提取出论文中的文本，再将文本分为句子，输入命名实体识别模型，抽取出句子中的实体。

若句子中同时存在两个不同实体，则形成（文本，实体对）输入关系抽取模型，获得实体对之间的关系。此外，为尽可能完善、丰富知识图谱，我们利用了清华NLP实验室开放的OpenNRE进行了额外的关系抽取。

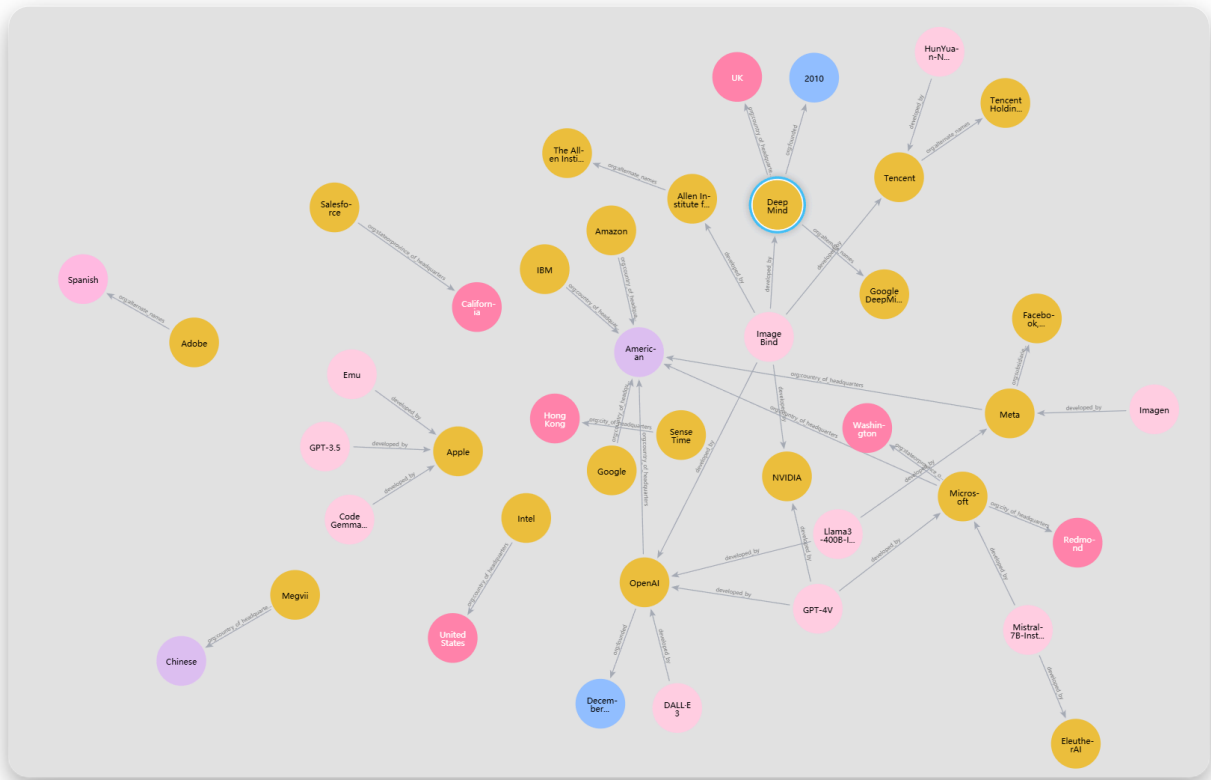
完成上述操作后即可获得（实体，实体，关系）的三元组

4.2 构建图模型

在我们看来，知识图谱本质上是一种异质图（heterogeneous graph），它拥有不同类别的节点、节点间拥有不同类别的边关系。因此，我们采用networkx中的 MultiDiGraph 对知识图谱进行存储。并将知识图谱保存为pkl数据，便于直接读取。

4.3 可视化知识图谱

可以直接使用networkx与matplotlib，两个库直接能非常好的配合。但这样的形成的图缺乏交互性，且不利于共享。因此我们利用 py2neo 连接到neo4j中的数据库，将知识图谱插入到neo4j数据库，进行可视化。



neo4j中构建的知识图谱局部图

5. 知识图谱嵌入

知识图谱嵌入部分基于TransE模型的知识图谱嵌入方法，使用Python实现，利用向量空间中的距离来表示知识图谱中的实体和关系。

5.1 向量表示

5.1.1 模型架构

1. **数据预处理**：读取包含实体、关系和三元组的数据文件，将其转换为适合模型训练的格式。
 - 读取实体和关系的文件，并将每行数据拆分成列表。
 - 读取三元组数据文件，将每行数据拆分为三元组形式。
2. **向量初始化**：使用均匀分布随机初始化实体和关系的向量，并进行归一化处理。
 - 生成均匀分布的随机矩阵。
 - 对随机矩阵进行归一化处理，以保证向量的模为1。
3. **负样本生成**：通过随机替换三元组中的头实体或尾实体生成负样本。
 - 根据一定概率替换头实体或尾实体，生成与正样本对应的负样本。
4. **模型训练**：计算正样本和负样本的距离，并通过合页损失函数更新向量。
 - 计算正样本和负样本的距离，使用L1或L2距离度量。
 - 根据合页损失函数计算误差，并更新实体和关系的向量。
 - 在每个epoch结束时打印损失值，以监控训练过程。
5. **保存模型**：将训练好的实体和关系向量保存，以便在推理阶段使用。

5.1.2 关键代码说明

在实现TransE模型的过程中，最关键的部分是模型训练中的向量更新部分，即update函数。其涉及向量的实际调整和优化，直接影响模型的性能和训练效果。

- 合页损失计算：这部分代码计算了正样本和负样本的距离差，并通过合页损失函数 (hinge loss) 来衡量模型的表现。
- 向量更新规则：根据计算的损失，决定如何调整实体和关系的向量，使得模型在下一次预测时表现更好。
- 梯度计算和更新：通过计算梯度并应用学习率，来逐步调整向量，从而优化模型。

```

def update(self, t_batch):
    for triplet_with_corrupted_triplet in t_batch:
        head_entity_id = triplet_with_corrupted_triplet[0][0]
        tail_entity_id = triplet_with_corrupted_triplet[0][1]
        relation_id = triplet_with_corrupted_triplet[0][2]
        corrupted_head_entity_id = triplet_with_corrupted_triplet[1][0]
        corrupted_tail_entity_id = triplet_with_corrupted_triplet[1][1]
        # tripletWithCorruptedTriplet是原三元组和打碎的三元组的元组tuple
        head_entity_vector = self.entity_vector_dict[head_entity_id]
        tail_entity_vector = self.entity_vector_dict[tail_entity_id]
        relation_vector = self.relation_vector_dict[relation_id]
        corrupted_head_entity_vector = self.entity_vector_dict[corrupted_head_entity_id]
        corrupted_tail_entity_vector = self.entity_vector_dict[corrupted_tail_entity_id]

        # 期望此距离尽可能小
        dist_triplet = distance(head_entity_vector, tail_entity_vector, relation_vector, l1=self.l1)
        # 期望此距离尽可能大
        corrupted_dist_triplet = distance(corrupted_head_entity_vector, corrupted_tail_entity_vector,
                                         relation_vector, l1=self.l1)

        # 合页损失函数
        err = self.margin + dist_triplet - corrupted_dist_triplet
        # 小于0时说明当前向量设置没问题，不需要更新
        if err > 0:
            self.loss += err
            temp_positive = 2 * (tail_entity_vector - head_entity_vector - relation_vector)
            temp_negative = 2 * (corrupted_tail_entity_vector - corrupted_head_entity_vector - relation_vector)
            if self.l1:
                temp_positive = np.array([1 if s >= 0 else -1 for s in temp_positive])
                temp_negative = np.array([1 if s >= 0 else -1 for s in temp_negative])
            temp_positive = temp_positive * self.learning_rate
            temp_negative = temp_negative * self.learning_rate

            head_entity_vector += temp_positive
            tail_entity_vector -= temp_positive
            relation_vector += temp_positive - temp_negative
            corrupted_head_entity_vector -= temp_negative
            corrupted_tail_entity_vector += temp_negative

            self.entity_vector_dict[head_entity_id] = head_entity_vector / np.linalg.norm(head_entity_vector)
            self.entity_vector_dict[tail_entity_id] = tail_entity_vector / np.linalg.norm(tail_entity_vector)
            self.relation_vector_dict[relation_id] = relation_vector / np.linalg.norm(relation_vector)
            self.entity_vector_dict[corrupted_head_entity_id] = corrupted_head_entity_vector / np.linalg.norm(
                corrupted_head_entity_vector)
            self.entity_vector_dict[corrupted_tail_entity_id] = corrupted_tail_entity_vector / np.linalg.norm(
                corrupted_tail_entity_vector)

```

5.2 知识推理

本部分通过预测关系中的尾实体来测试TransE模型的性能，读取实体和关系的ID映射表及其向量表示，并利用余弦相似度计算来预测尾实体。

- 输入：头实体和关系的名称
- 输出：预测的尾实体

5.2.1 实现细节

1. 读取数据

- 实体和关系的ID映射表，将每一行解析成实体或关系与其对应的ID，并存储在字典中。
- 实体和关系的向量表示，每一行包含一个实体或关系及其对应的向量。通过解析这些文件，我们将向量存储在字典中，字典的键为实体或关系的名称，值为其对应的向量。这

一步确保我们能够快速获取每个实体和关系的向量表示。

2. 计算余弦相似度

为了比较向量之间的相似度，我们定义了一个函数来计算两个向量之间的余弦相似度。

余弦相似度是通过计算两个向量的点积并除以它们的范数之积来实现的。这个度量标准在高维空间中非常有效，特别适用于嵌入向量的相似度计算。

3. 预测尾实体

通过使用头实体和关系的向量，我们可以预测尾实体的向量。具体步骤如下：

- 获取头实体和关系的向量。
- 将头实体向量与关系向量相加，得到预测的尾实体向量。
- 遍历所有实体向量，计算每个实体向量与预测尾实体向量的余弦相似度。
- 选出相似度最高的实体，作为预测的尾实体。

5.2.2 实验结果

通过输入特定的头实体和关系，我们能够预测出尾实体。实验验证了该模型的有效性，即通过余弦相似度计算，能够准确找到最相似的实体向量，从而实现关系预测的目标。

```
(vllm) ubuntu@ubuntu:/mnt/workspace/llm_kg$ ^C
(vllm) ubuntu@ubuntu:/mnt/workspace/llm_kg$ cd /mnt/workspace/llm_kg ; /mnt/anaconda3/envs/vllm/bin/python /home/ubuntu/.vscode-server/extensions/ms-python.debugpy-2024.6.8-linux-x64/bundled/libs/debugpy/..../debugpy/launcher 42889 -- /mnt/workspace/llm_kg/transe_test.py
Mistral-7B-Instruct-v0.3
is open?
预测结果开源
(vllm) ubuntu@ubuntu:/mnt/workspace/llm_kg$
```

本实验成功实现了基于TransE模型的关系预测。通过读取实体和关系的向量表示，并使用余弦相似度计算，代码能够有效地预测给定头实体和关系的尾实体。这验证了TransE模型在关系预测任务中的有效性。

5.2.3 未来工作

未来可以进一步优化和扩展该模型，例如：

- 使用更复杂的向量表示方法，提高预测准确性。
- 集成更多的预处理步骤，提升数据质量。
- 在更大规模的数据集上进行实验，验证模型的扩展性。
- 通过以上改进，可以进一步提升模型的性能和实用性。

6. 问题分析

6.1 为什么模型实体识别效果差

从命名实体识别模型那节中的识别结果可以发现，MOD，即模型，识别的精准度、召回率都比较低。

我们猜测是由于直接采用bert-base-uncased的分词器，并使用了它的原始词表。分词器是自然语言处理中的一个关键组件，用于将文本分割成更小的单元，如单词、子词或字符。而BERT全系的分词器都使用了WordPiece分词方法，这种分词方法会将输入文本分解成子词单元。对于未知词，分词器会将其分解成已知子词。例如，词汇表中没有“unhappiness”，但有“un”，“##happy”和“##ness”。而模型名称通常是一些组合词，甚至是非常偏僻、新造的词，例如“PaliGemma-3B”、“Yi-1.5-34B”两个模型的分词结果如下：

```
['pali', '##ge', '##mma', '-', '3', '##b']  
['yi', '-', '1', '.', '5', '-', '34', '##b']
```

绝大多数模型的名称都不在词表中，这导致在处理这些名称时，分词器会将其切割为很细碎、毫无语义特征的小块，这对于后续的处理无疑是不利的。

重新构建一个词表可行吗？或许可行，许多模型在迭代过程中都只更改模型名称后面的数字，前面的字符一般不会改变。但对于新的模型，拥有新的名称，同样会遭遇上述问题，这对于一个自动更新的、希望一劳永逸的知识图谱是不利的。

6.2 Llama3为什么是OpenAI开发的

在生成的知识图谱中，我们发现会出现一些错误的关系，例如（Llama3, OpenAI, developed by）。

我们猜测有两种可能，

- 远程监督时由于两者在同一个句子中，从而产生的错误关系（我们将远程监督产生的训练样本同样用于了构建知识图谱）
- 模型训练损失函数不完善

对于第二点，主要是因为关系抽取是，我们将问题视为多分类任务，并简单地使用了交叉熵损失。对于模型来说，我们提供给它的只有正样本，即正确的关系三元组。训练结束后，它能很好地将之前它看见过的三元组抽取出来。但是在推理过程中，产生 (Llama3, OpenAI, developed by) 这样的三元组，对于模型来说并不奇怪。训练数据中不存在负样本，模型没有因为产生这样的三元组而得到什么惩罚，或者损失。

缺乏负样本，使得模型可能陷入一个误区，即对于 (模型, 公司) 实体对，它只需要输出‘被开发’即可，并不会学习到‘这个模型不是被这个公司开发’的知识。