AFAC 实验报告

1 概述

本文档是该项目的技术文档。主要包含PiVe训练与调用、MindMap使用、MetaGPT使用的技术细节。

任务1中使用了PiVe模型修正搜索到的三元组，任务2中使用了MindMap进行KG-RAG，任务3使用MetaGPT自动生成研报，同时，任务4中使用Gradio设计便于操作与展示的前端。

后续披露了我在完成该任务时遇到的困难与解决方法，以及一些个人新的体会

有关PiVe、MindMap的原理、原论文的讲解，在另一份提交文档中。

2 任务0：服务器配置

由于一些模型的训练、推理需要在gpu进行，所以选择在AutoDL网站租用显卡进行任务（https://www.autodl.com）而该服务器具有 无独立公网IP、代理地址受限等一系列与任务无关的问题，所以在任务开始前 / 任务进行时，我们需要处理这些问题

2.1 配置SSH隧道

由于该服务器不具有独立公网IP，因此，在服务器配置好Neo4j后，无法通过本地浏览器通过http协议进行访问。根据官方文档，我们通过提供的程序，配置SSH隧道代理。将代理端口设置为Neo4j与Gradio所在的启动端口，现在可以正常访问

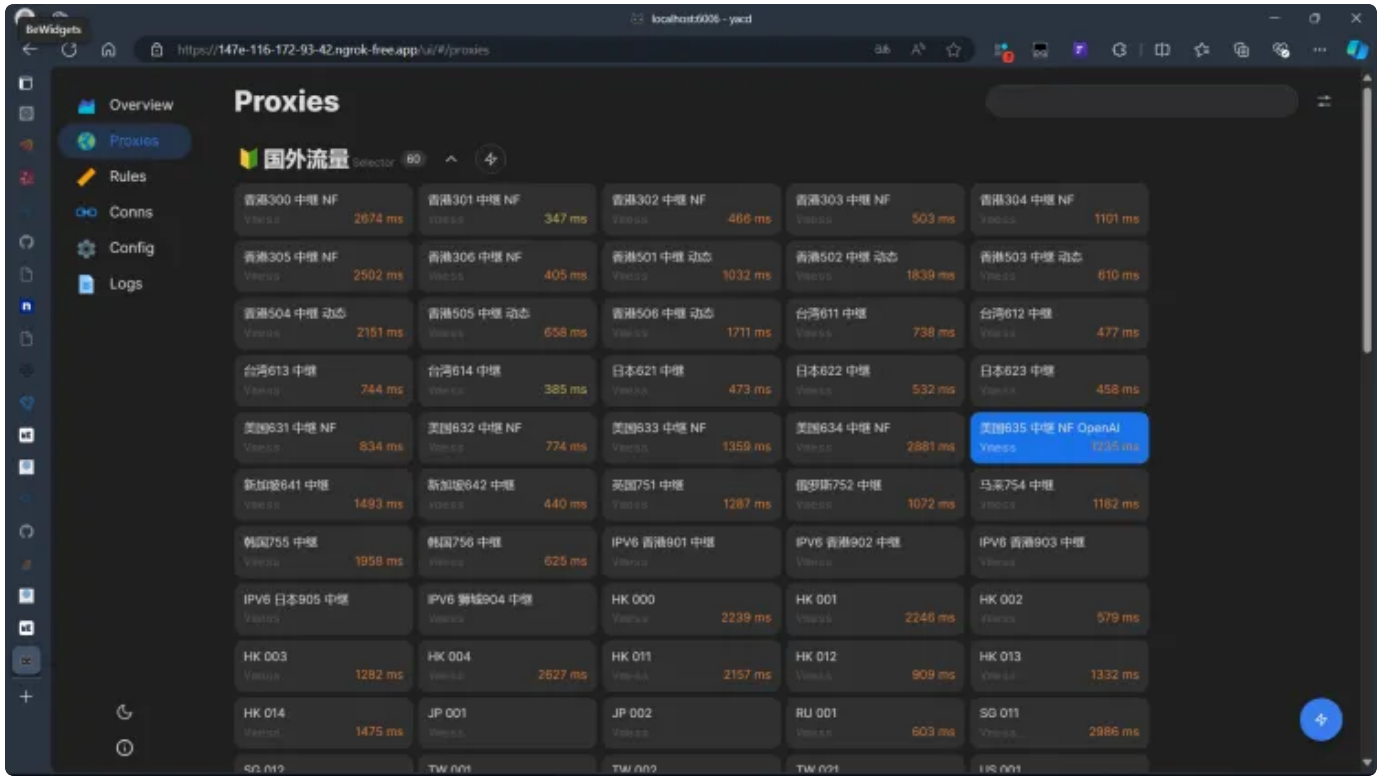
2.2 代理网站范围受限

以下为可以加速访问的学术资源地址：

- github.com
- githubusercontent.com
- githubassets.com
- huggingface.co

以上是该服务器能通过代理访问的网站，然而，在我们访问OpenAI相关的地址时，就会因链接超时引发一系列错误。

GitHub – VocabVictor/clash-for-AutoDL: AutoDL平台服务器适配梯子， 使用 Clash 作为代理工具
该项目通过Nrgok对服务器进行内网穿透，可以在服务器上对Clash进行部署。此时，导入购买的配置url，AutoDL可以通过代理正常访问其他海外网站。



选择对应的海外全局代理节点，才可访问OpenAI相应接口

3 任务1：构建知识图谱

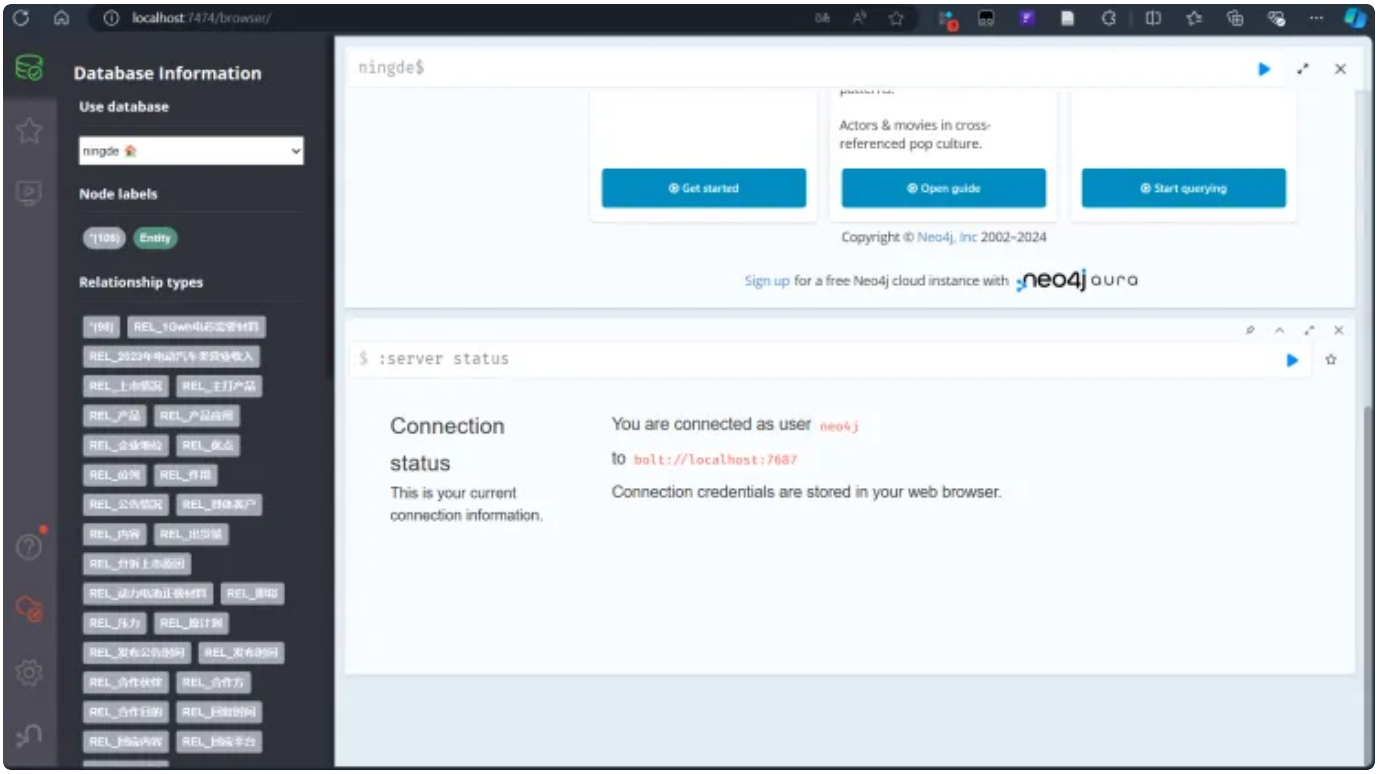
构建知识图谱的过程，就是将文本中的 实体–关系–实体 三元组抽取并插入到图数据库的过程。

3.1 基本思路

我们使用Neo4J构建图数据库，通过修改配置文件，选定要使用的数据库。

```python
dbms.default_database=ningde
```

设置用户名、密码后成功登入数据库



根据PiVe模型文档中的指示，我们首先使用大语言模型构建三元组。由于数据集的语言全部为中文，故使用中文大语言模型 `ERINE 4.0-8K` 进行推理。

> 2024年07月02日 23:33 来自 来源： 中国基金报 资讯：“（公司）整体排产情况良好，近期及第三季度排产环比呈增长态势。”7月2日，宁德时代针对近期市场传言，现场回应多家知名投资机构。7月1日，市场传言锂电行业7月排产数据普遍下滑，... ... <消息来源>: http://finance.eastmoney.com/a/202407023119921425.html

利用大语言模型对其进行直接推理，获得三元组如下：

> ["宁德时代", "回应", "整体排产情况良好，近期及第三季度排产环比呈增长态势"]["宁德时代", "回应时间", "7月2日"]
>
> ["宁德时代", "回应地点", "宁德时代总部"]
>
> ... ...

而后，我们需要利用训练后的 univerifier 对三元组进行修正，确保没有遗漏三元组

3.2 Univerifier 训练 与 推理

我们使用 `flan-t5-xxl-sharded-fp16` 模型进行训练、推理。在训练时，我们使用LoRa方法进行微调，超参数设置如下：

attention dimension	lora_alpha	lora_dropout	target_modules	task_type
4	16	0.05	q, v	SEQ_2_SEQ_LM

在加载预训练模型时，精度选择 `torch.float16` ，以 INT `8` 进行Tensor量化。除从原始状态开始训练外，也设置了从检查点加载模型，进行进一步的训练。

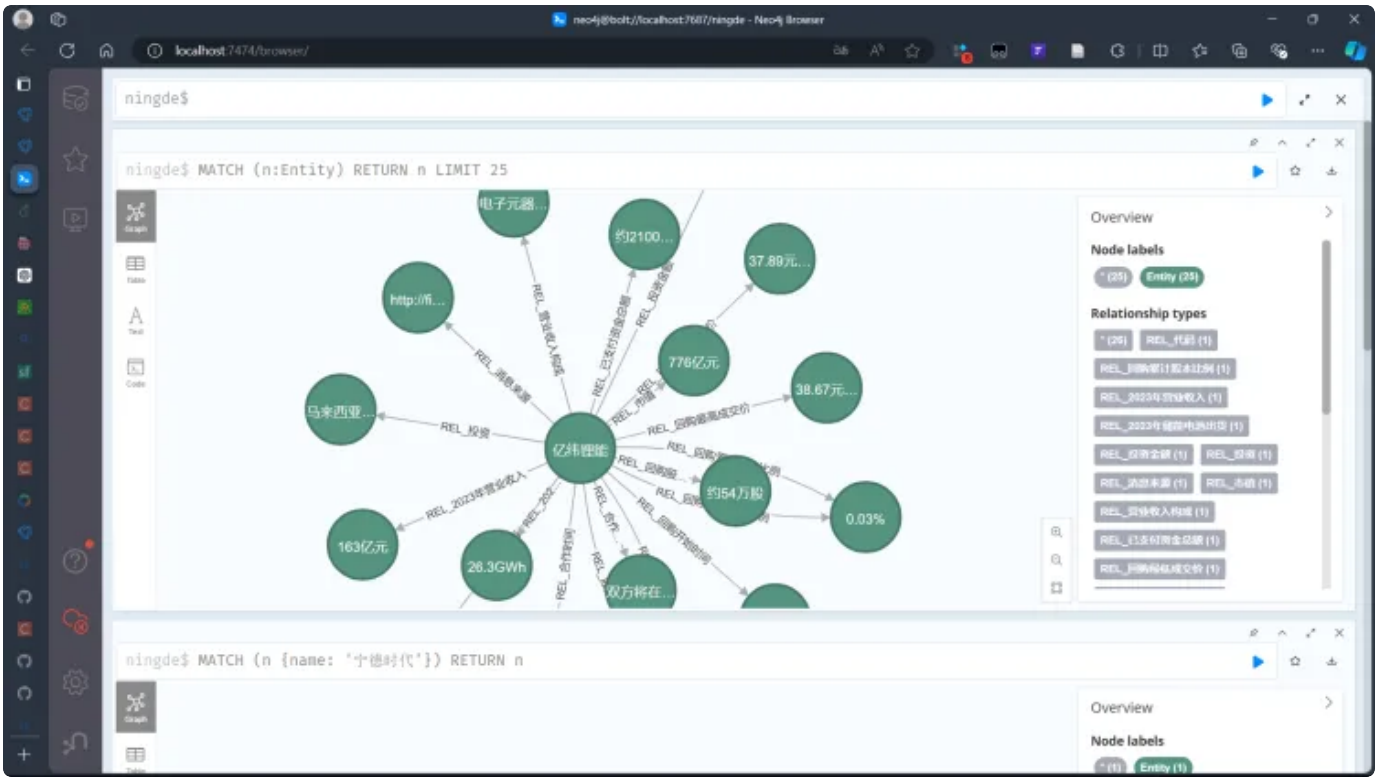
而后，训练配置的关键步骤，配置Trainer中的config参数：

batch_size	warmup_step	learning_rate	epoch	opim	eval_step	save_step
40	100	3e-4	2	admaw	200	200

训练后，推理时直接加载保存好的 `checkpoints` ，获得推理结果。如果大语言模型成功的找到了全部三元组，则返回 `Correct` ，否则返回遗漏的三元组

3.3 效果

我们将修正后的三元组全部插入到Neo4j中，我们已亿纬锂能为例，获得如下知识图谱（部分）：



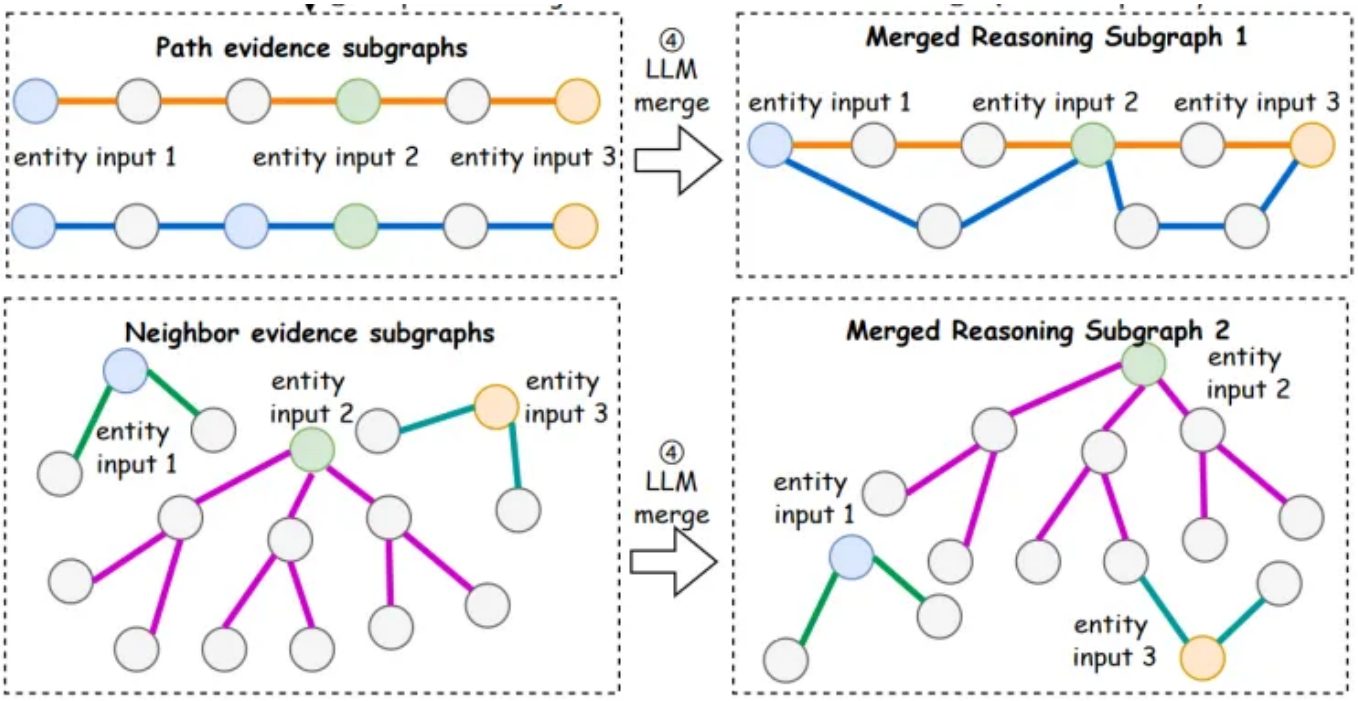
4 任务2：利用MindMap实现KG-RAG

4.1 实现流程

在MindMap中，共设置了4种方法得到答案：

- 通过大模型直接获取答案
- 通过BM25检索文档获得答案
- 通过Embedding直接检索答案
- 通过知识图谱检索答案

Step 1：读取图数据库中的节点，并将输入中提及的节点进行合并（Merge）




```python
query = (
    "MERGE (h:Entity { name: $head_name }) "
    "MERGE (t:Entity { name: $tail_name }) "
    "MERGE (h)-[r:" + relation_name + "]->(t)"
)
```

Step 2: 我们需要在知识图谱中找到和Question中各个实体对应的节点。例如，问题中提及了宁德时代，我们就要在知识图谱中找到宁德时代；问题中提到了 "马来西亚储能电池及消费类电池制造项目"，可以在知识图谱中找到与其表述相似的实体

对图中节点、对问题中的实体进行Embedding：

```python
# 对图节点Embedding
kg_entity_embd, kg_entities = graphEntityEmbedding(driver)
... ...

# 对问题中的实体进行Embedding
for q_entity in question_kg:
    ques_entity_embd = embeddingQianfan([q_entity])[0][0]['embedding']
... ...
```

对于问题中的存在某个实体，对Embedding后的实体计算其余弦相似度，选择余弦相似度最高的那个实体，作为该问题实体在知识图谱中对应的位置

```python
cos_similarities = cosine_similarity_manual(kg_emb, ques_entity_embd)
if cos_similarities > max_sim and kg_entity not in match_kg:
    max_sim = cos_similarities
    match_kg.append(kg_entity)
```

Step 3: 知识图谱寻路 与 邻接点寻路

`find_shortest_path(start_entity, end_entity, candidate_entity)` 函数用于找到从 `start_entity` 到 `end_entity` 的最短路径，并返回路径和存在的实体。

如果找不到路径，会抛出异常并打印错误信息，同时设置 `flag_kg` 为 `False` 并退出循环; 如果找到路径，则将路径拆分并添加到 `paths_list` 中。

```python
paths,exist_entity = find_shortest_path(start_entity, end_entity,candidate_entity)
```

其中， `find_shortest_path` 函数最核心的寻路功能是靠Cypher语句实现的：

```python
result = session.run(
    "MATCH (start_entity:Entity{name:$start_entity_name}), (end_entity:Entity{name:$end_entity_name}) "
    "MATCH p = allShortestPaths((start_entity)-[*..5]->(end_entity)) "
    "RETURN p",
    start_entity_name=start_entity_name,
    end_entity_name=end_entity_name
)
```

如果找到的路径数量超过5条，则按长度排序并只保留最短的前5条路径。返回的内容为路径列表 `paths` 和存在的实体 `exist_entity` 。

同理，还可以通过寻找源点邻接点来获得更多信息：

```python
query = """
    MATCH (e:Entity)-[r]->(n)
    WHERE e.name = $entity_name
    RETURN type(r) AS relationship_type,
           collect(n.name) AS neighbor_entities
    """
```

Step 4: 输出最短路、邻接点结果

将关键路径上的节点以文字形式输出。这时需要再次借助LLM进行转化，其Prompt模板如下：

```python
template = """
有一些知识图谱路径。它们遵循实体->关系->实体的格式。
\n\n
{Path}
\n\n
使用这些知识图谱信息。尝试将它们分别转换为自然语言。使用单引号标注实体名称和关系名称。并将它们命名为路径证据1，路径证据2，依此类推。\n\n

输出：
"""
```

Step 5：将原始输入的问题、**Step 4**输出的关键路径统一结合起来，获得最后的输出。Prompt如下设计：

```python
messages = [
    {
        "role": "user",
        "content": "你是一位优秀的证券分析师，可以根据金融知识信息回答投资者问题。你有以下内容可以参考："
    },
    {
        "role": "assistant",
        "content": assisContent
    },
    {
        "role": "user",
        "content": "根据以上信息，请你回答这只股票可能面临什么情况？应该进行哪些分析来确认预测？投资者应当选择持有还是卖出？请一步步思考。\n\n\n"
        + "Output1：显示推理过程，将提取哪些知识来源于哪个路径证据或邻接点证据，并最终推理出结果。请将推理过程转换为以下格式：\n 路径证据编号('实体名称'->'关系名称'->...)->路径证据编号('实体名称'->'关系名称'->...)->邻接点证据编号('实体名称'->'关系名称'->...)->邻接点证据编号('实体名称'->'关系名称'->...)->结果编号('实体名称')->路径证据编号('实体名称'->'关系名称'->...)->邻接点证据编号('实体名称'->'关系名称'->...)。\n\n"
        + "Output2：请你将以上推理结果转化为一段自然语言，并回答问题应当选择持有还是卖出，给出确切选择。\n\n"
        + "Output3：绘制一棵决策树。推理过程中单引号中的实体或关系作为节点，后面跟随证据来源（实体名称），加入决策树。\n\n"
        + "以下是一个示例，你的回答中不可以包含示例中任何具体内容：\n"
        + """

                Output 1:
                路径证据1('公司'->'发布'->'利好消息')->路径证据2('利好消息'->'可能导致'->'股票价格上涨')->邻接点证据1('技术分析'->'包含'->'移动平均线分析')->邻接点证据2('技术分析'->'包含'->'相对强弱指数分析')->结果1('股票价格上涨')->路径证据3('公司'->'发布'->'财报')->邻接点证据3('市场动态'->'可能影响'->'股票价格').

                Output 2:
                根据当前的市场状况和公司发布的消息，这只股票可能面临短期内价格上涨的情况。为了确认这一预测，建议进行技术分析，包括移动平均线和相对强弱指数（RSI）分析。此外，应该关注公司的财报和市场动态。推荐的投资策略包括在突破关键阻力位时买入，并在短期内持有以获取快速收益。同时，建议设置止损位以控制风险。

                Output 3:
                公司(路径证据1)
                └── 发布(路径证据1)
                    └── 利好消息(路径证据1)(路径证据2)
                        └── 可能导致(路径证据2)
                            └── 股票价格上涨(路径证据2)(邻接点证据1)
```

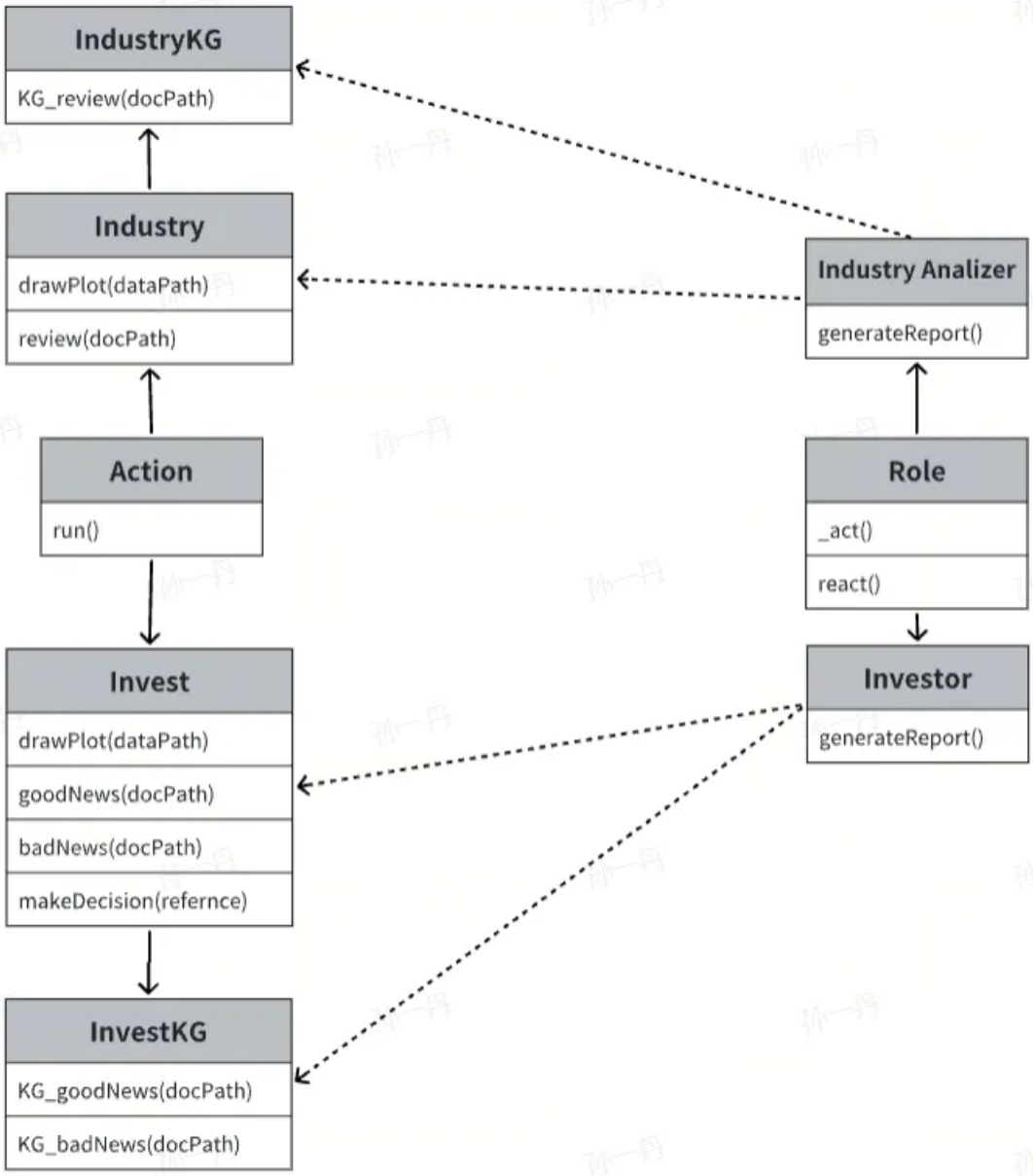
```
                                                    ├── 包含(邻接点证据1)
                                                    │   └── 移动平均线分析(邻接点证据1)(邻接点证据
2)
                                                    │       └── 包含(邻接点证据2)
                                                    │           └── 相对强弱指数分析(邻接点证据2)
(结果1)(路径证据3)
                                                    ├── 发布(路径证据3)
                                                    │   └── 财报(路径证据3)(邻接点证据3)
                                                    └── 可能影响(邻接点证据3)
                                                        └── 市场动态(邻接点证据3)
                    """
    }
]
```

这是一个多轮对话的请求，其中对于Output 1 与 Output 3 部分的Prompt是可以省去的，仅作展示思路用途

Step 6: 调用其他API，获得其他渠道输出的答案（如：直接调用chatGPT，使用BM25进行检索，... ...）

5 任务3：自动生成研报

我们使用MetaGPT进行研报的自动生成。关于MetaGPT的基本架构与原理可见另一份提交文档。我们在此任务中设计了几个新的Action与Role，以下是UML图



5.1 配置信息

在启动MetaGPT前，首先需要配置大模型等相关信息

```python
llm:
api_type: 'qianfan'
api_key: 'T6OdiNjHJkDGMyG1d23dQ5TN'
secret_key: 'iyQbYVfGSbWq4MTlR7Z9wydht4JrTbOw'
model: 'ERNIE-4.0-8K'
max_token: 2048
```

我们使用千帆大模型平台（https://console.bce.baidu.com/qianfan/ais/console/onlineService）提供的接口，调用 `ERNIE-4.0-8K` 作为处理任务的大语言模型；同时，设置最大Token数量为2048个（若不设为2048会报错）

未使用KG-RAG时：在搜索方式上，我们选用 `ChromaRetrieverConfig()` 与 `BM25RetrieverConfig()` 进行混合搜索，对搜索后的结果再进行去重，便可获得最终检索结果。对于 `ChromaRetrieverConfig()` ，其需要引入额外的Embedding Model才可以进行检索。该项目对Embedding模型有一定要求，必须是llama index支持的嵌入模型才可以使用。我们用的是 `nomic-embed-text-v1.5` 模型（https://huggingface.co/nomic-ai/nomic-embed-text-v1.5），值得一提的是，该Embedding模型的配置还同时需要依赖 `nomic-bert-2048` 模型（https://huggingface.co/nomic-ai/nomic-bert-2048）作为其基础配置

5.2 行为Action：文档检索与图谱检索

我们以生成个股研报为例，其研报包含四个部分：

- 有关价格走势的图片
- 近期利好消息
- 近期不利消息
- 投资决策

对于价格走势的图片，通过以获得的数据再搭配Matplotlib即可轻松实现，价格折线图形如：

对于利好消息，可以设计prompt如下（不利消息同理）：

```python
prompt = f"""
今天是2024年7月7日，{self.stockName}最近有什么利好消息吗？你的回答格式应符合以下几点要求：
1. 请直！接！回复答案，不需要有任何备注，开头也不需要任何引入
2. 任何消息必须注！明！具体的消息来自哪个媒体，并将出处穿插到回答中，不要在最后同一注明。
3. 输出一段文字，而不是逐条输出。
4. 根据每一条消息推测出对其近期的影响。

<Example>
近期，宁德时代 ... ...

消息来源：
    1. [南方财经网](http://finance.eastmoney.com/a/202407023119440882.html)
    2. [蓝鲸财经](http://finance.eastmoney.com/a/202406273115868477.html)
<\Example>
"""
```

对于投资决策，为保证上下文语义一致性，我们根据检索到的利好消息、不利消息、价格数值共同作为信息输入到大模型中，做出语义一致的决策。Prompt设计如下：

```Python
prompt = f"""
现有关于{self.stockName}的资讯分析如下:
{content}
该股票近7日收盘价为:
{prices}
请你根据这些内容，确定是继续持有还是卖出该股票。你的回答应符合以下要求:
1. 直接回答你建议投资者的决策是 持有 或 卖出，不要回复任何其他内容!
"""
```

而对于KG-RAG部分，由于其检索过程脱离了MetaGPT设定好的框架，需要调用MindMap来进行处理。然而，MindMap与MetaGPT属于不同的conda环境，因此，首先需要处理环境不兼容问题。

在MindMap的conda环境下，使用socket搭建服务程序:

```Python
# 创建 socket 对象
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 获取本地主机名
host = 'localhost'
port = 9000
# 绑定端口
server_socket.bind((host, port))
# 设置最大连接数，超过后排队
server_socket.listen(5)
while True:
    # 建立客户端连接
    client_socket, addr = server_socket.accept()
    print(f"连接地址: {addr}")
    # 接收客户端数据
    data = client_socket.recv(1024).decode('utf-8')
    question = data
    reply = graphRag(question, "/root/autodl-tmp/MindMap/data/Fin/relNing.txt")["kg_retrieval"]
    client_socket.send(reply.encode('utf-8'))
    # 关闭连接
    client_socket.close()
```

同时，在对应的Action函数中，也使用socket搭建客户端程序:

```python
# 创建 socket 对象
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 获取本地主机名
host = 'localhost'
port = 9000

# 连接到服务器
client_socket.connect((host, port))

# 发送数据到服务器
message = question
client_socket.send(message.encode('utf-8'))

# 接收来自服务器的数据
answer = client_socket.recv(1024).decode('utf-8')
print(f"从服务器接收到的数据: {answer}")

# 关闭连接
client_socket.close()
```

在处理行业相关Action时，需要在一个函数中完成对10支股票的分析。如果串行生成报告是十分耗时的，此时应当选择异步生成结果，故，修改Action代码如下：

```python
todos = (self.singleReview(stockName) for stockName in stockLis)
result = await asyncio.gather(*todos)
```

5.3 角色Role：个股研报生成者 与 行业研报生成者

我们将角色分为两个，一个用于生成个股研报，另一个用于生成行业研报。

在个股研报生成者中，**首先**设置静态成员变量对生成信息进行保存：

```python
_plotPart: str = ""
_goodNews: str = ""
_badNews: str = ""
_decision: str = ""
```

在最后生成报告时将其进行拼接。

然后，重载Role基类的 `_act` 函数

```python
if isinstance(todo, DrawPlot):
    result = await todo.run(stockName, DATA_PATH)
    self._plotPart = result
    ret = Message(
        content=result, role=self.profile, cause_by=todo
    )

elif isinstance(todo, GoodNews):
    ... ...

elif isinstance(todo, KG_GoodNews):
    ... ...

elif isinstance(todo, BadNews):
    ... ...

elif isinstance(todo, KG_BadNews):
    ... ...

elif isinstance(todo, MakeDecision):
    ... ...
else:
    raise ValueError(f"Unknown todo: {todo}")
```

最后，重载Role基类的 `react` 函数，将报告内容输出到.md文件中，以便Gradio进行后续读取

```python
async def react(self) -> Message:
    msg = await super().react()
    filePath = self.write_report()
    return filePath
```

对于行业研报生成者亦是同理，稍作修改：

```python
if isinstance(todo, DrawPlot):
    result = await todo.run(stockName, DATA_PATH)
    self._plotPart = result
    ret = Message(
        content=result, role=self.profile, cause_by=todo
    )

elif isinstance(todo, Review):
    result = await todo.run(stockNames)
    self._review = result
    ret = Message(
        content=result, role=self.profile, cause_by=todo
    )
```

6 任务4：利用Gradio实现可视化

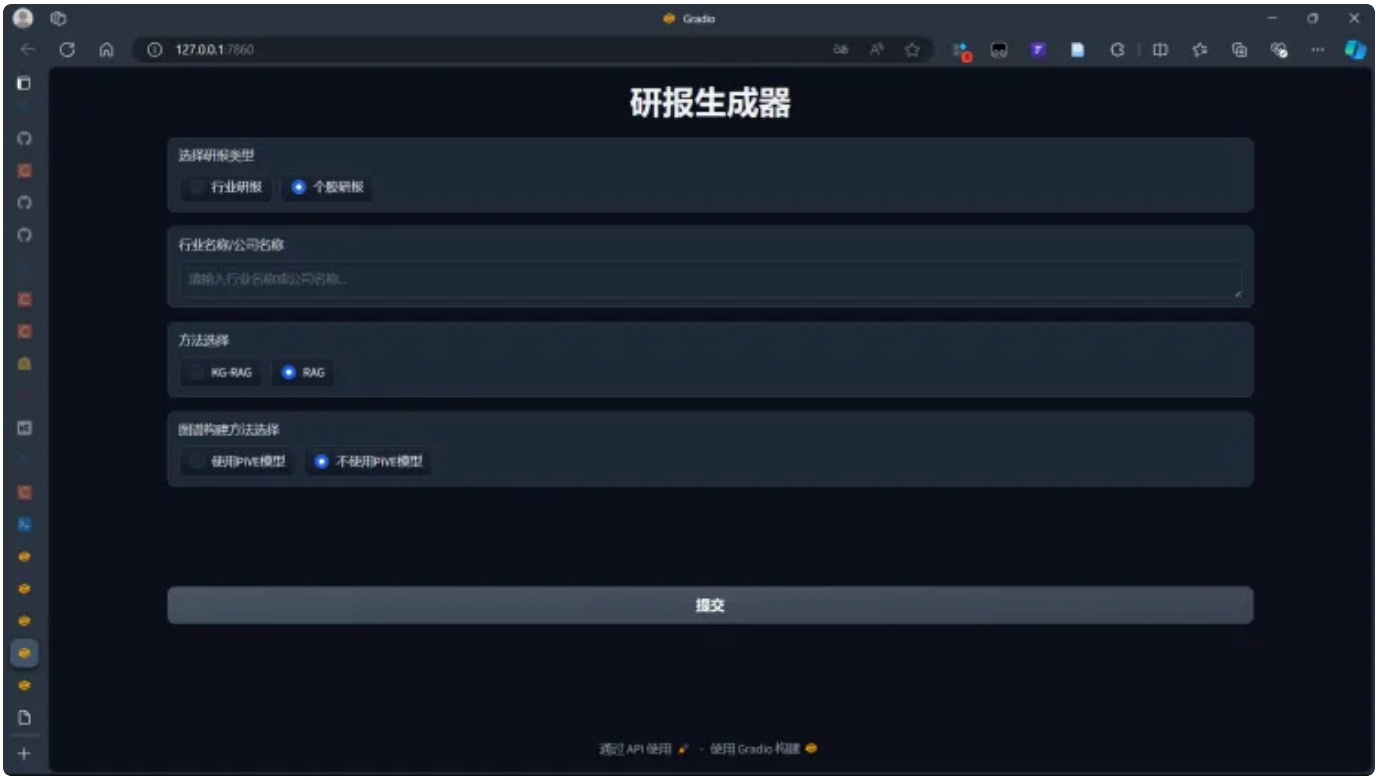
当结果输出在操作台时，不便于从业人员选择方法、阅读结果，因此需要设计前端来对整个系统进行封装。

Gradio是深度学习中常用的库，搭建前端时十分便捷。利用Gradio对前端进行设计如下：

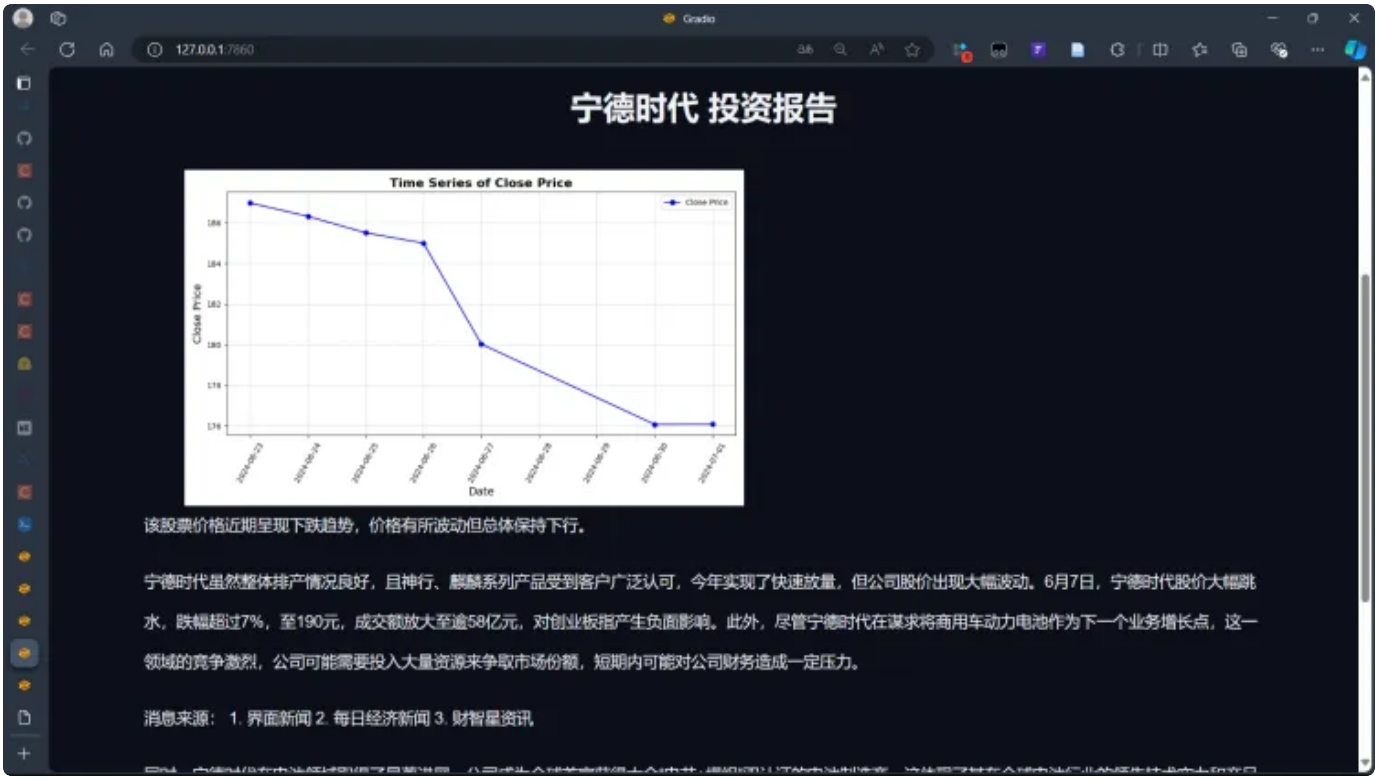
```python
gr.Markdown("## 研报生成器")
gr.Markdown("请在此界面选择研报类型，输入名称，选择方法和图谱构建方法，并查看输出的Markdown文档。")

with gr.Row():
    report_type = gr.Radio(choices=["行业研报", "个股研报"], label="选择研报类型")

with gr.Row():
    name_input = gr.Textbox(lines=1, placeholder="请输入行业名称或公司名称...", label="行业名称/公司名称")

with gr.Row():
    method_choice = gr.Radio(choices=["KG-RAG", "RAG"], label="方法选择", value="RAG")

with gr.Row():
    pive_choice = gr.Radio(choices=["使用PiVE模型", "不使用PiVE模型"], label="图谱构建方法选择", value="不使用PiVE模型")

output = gr.Markdown(visible=True, elem_classes=["output-markdown"])
```

在其中，我们提供了几种选择：第一，是否要用PiVe对生成图谱质量进行增强；第二，是否要使用MindMapjinxKG-RAG过程。最终，设计一个部分用来展示markdown文档。为了美观，我们也设置了形如chatGPT界面一样的延时显示。

效果如下图所示：



选择、输入后，获得结果如下图所示：



7 遇到的困难与解决

7.1（耗时较长）MetaGPT无法允许载入本地模型

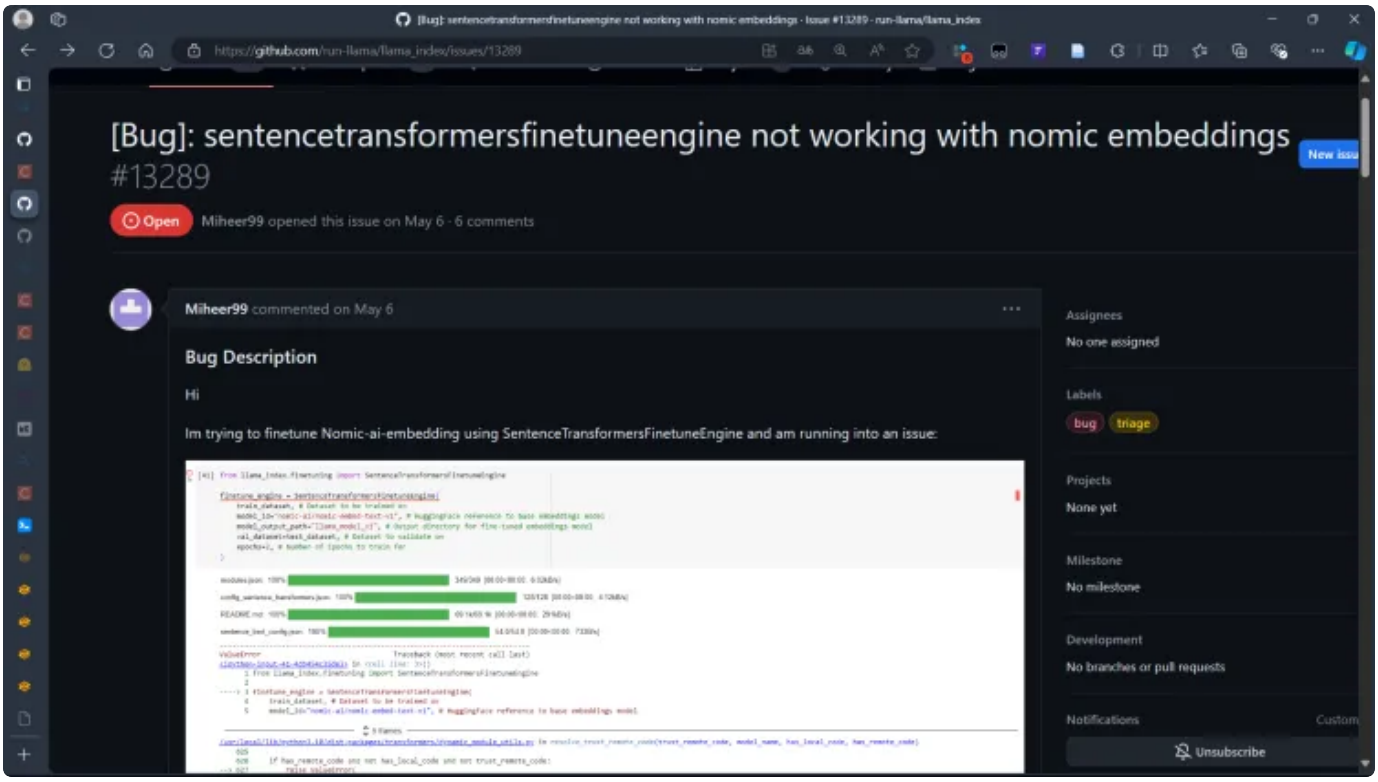
在进行基于文档的RAG时，我们使用了三种检索方法：

```python
[ChromaRetrieverConfig(), BM25RetrieverConfig(), FAISSRetrieverConfig()]
```

其中，`ChromaRetriever` 与 `FAISSRetriever` 在调用之前都需要加载嵌入模型。因此，我们决定使用nomic模型先对文本进行嵌入。

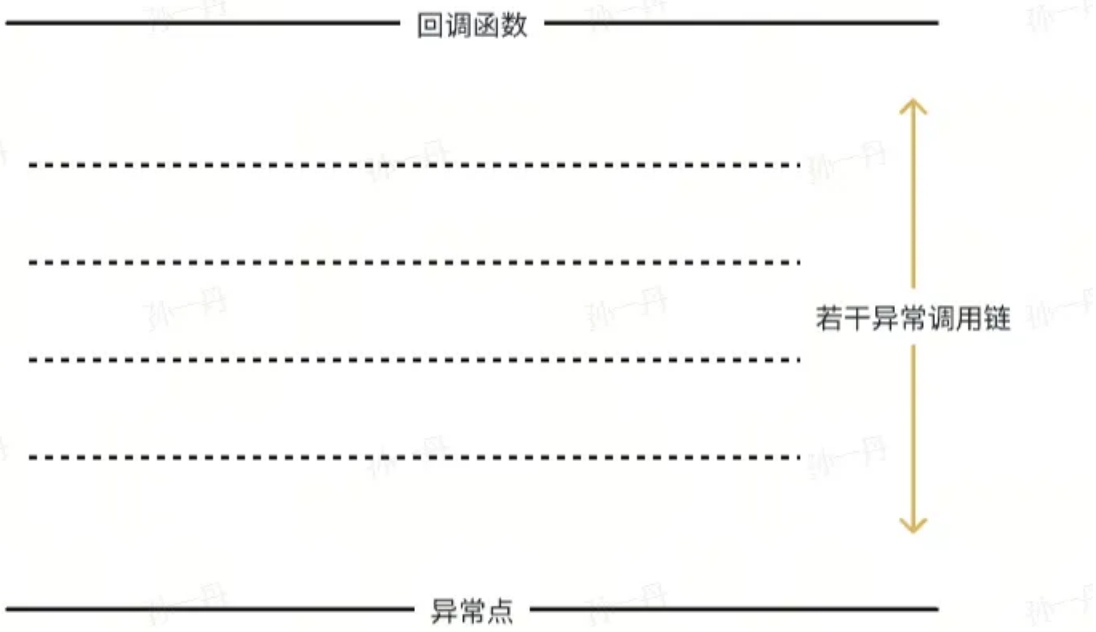
在调用本地下载好的模型时，往往需要在接口的传入参数中设置 `trust_remote_code=True`，然而，当我们在调用nomic这个Embedding模型时，尝试从各种位置添加该参数，结果都是报错，显示不允许加载本地代码。

因此，我们需要对异常的调用链进行回溯（修改bug后忘记保存异常调用链截图，故处省略异常调用链展示）。前往该项目在Github的issue中查找到相同问题：



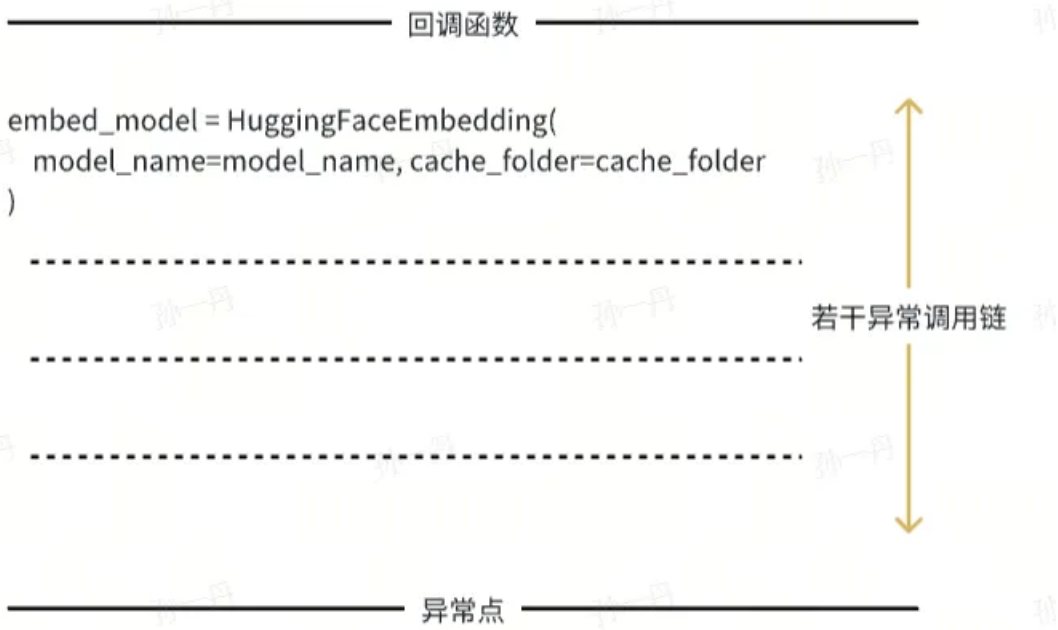
可见，该用户遇到了与我完全相同的问题，且被打上了bug的标签，因此我们需要对llama_index的库进行一些修复。

我们自下而上地遍历异常调用链，在每一个调用位置都打印传入的 `**kwargs`，观察是否有 `trust_remote_code=True` 出现。最后发现，问题源自于llama_index模块在调用某个回调函数时，将 `trust_remote_code=True` 丢失。此时，调用链异常的架构形如



我们首先找到在调用回调函数后，第一个接受形参的接口，位于 `llama_index/core/embeddings/utils.py` 的第109行：

```python
embed_model = HuggingFaceEmbedding(
    model_name=model_name, cache_folder=cache_folder
)
```



尝试加入形参：

```python
embed_model = HuggingFaceEmbedding(
    model_name=model_name, cache_folder=cache_folder, trust_remote_code=True
)
```

此时，再次启动程序，发现报错：

```
llama-index-embeddings-huggingface package not found, please run pip install llama-index-embeddings-huggingface
```

此时，我错误地以为，不可以对修改库中的函数进行修改，否则就会报无法导入的错误。我本想尝试沿着调用链继续修改传入形参，却发现调用链的下一层SentenceTransformer的传入形参 `trust_remote_code` 依赖于上一层。此时陷入僵局：上一次调用传入形参报错，下一层调用依赖于上一层。**对于此问题，我开始在网上查找大量资料，然而尝试数次，依旧报错。**

对于此情况，想到权宜之计：

1. 换掉nomic，使用其他Embedding模型
2. 去掉 `ChromaRetriever` 与 `FAISSRetriever`，只使用 `BM25` 进行检索

但后面发现这两种情况都面临着问题：

1. 载入其他Embedding模型依旧会面临设置 `trust_remote_code` 的问题，因为异常出在 `llama_index` 的库，而不是nomic的问题
2. 去除两种检索方式会使得生成效果大打折扣

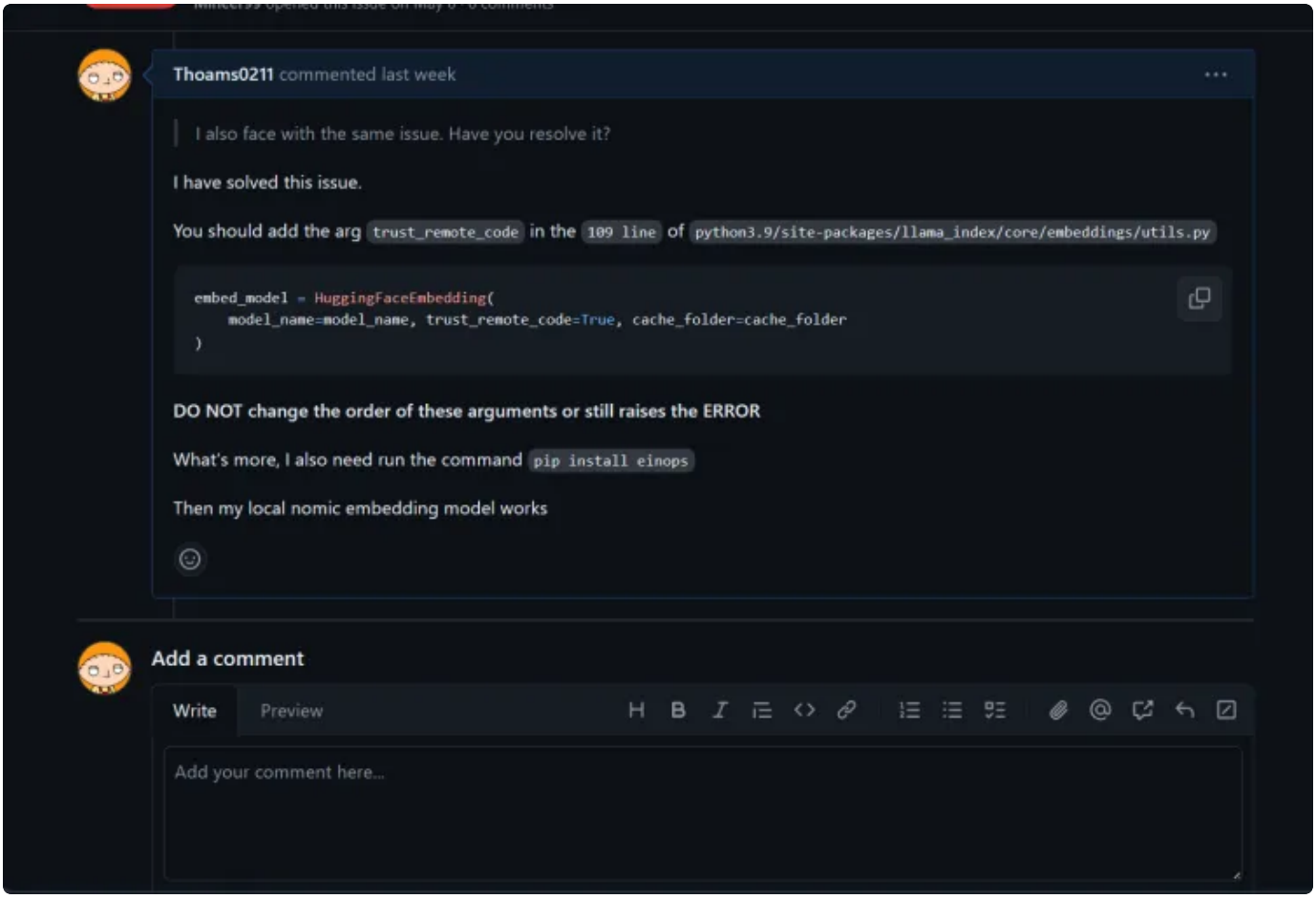
因此，继续阅读llama_index源码，我们发现：

```python
def __init__(
    self,
    model_name: str = DEFAULT_HUGGINGFACE_EMBEDDING_MODEL,
    tokenizer_name: Optional[str] = "deprecated",
    pooling: str = "deprecated",
    max_length: Optional[int] = None,
    query_instruction: Optional[str] = None,
    text_instruction: Optional[str] = None,
    normalize: bool = True,
    model: Optional[Any] = "deprecated",
    tokenizer: Optional[Any] = "deprecated",
    embed_batch_size: int = DEFAULT_EMBED_BATCH_SIZE,
    cache_folder: Optional[str] = None,
    trust_remote_code: bool = False,
    device: Optional[str] = None,
    callback_manager: Optional[CallbackManager] = None,
    **model_kwargs,
)
```

在其构造函数的传入参数中，后几行的参数大多数被打上了 `Optional` 的类型，而偏偏 `trust_remote_code` 没有。这让我想到Python中有关默认参数的问题。在先前，我先设置了 `cache_folder` 这一可选参数，而后设置了 `trust_remote_code` 这一非可选参数，我猜测应当调换参数传入位置。故修改代码如下：

```python
embed_model = HuggingFaceEmbedding(
    model_name=model_name, trust_remote_code=True, cache_folder=cache_folder
)
```

修改后，居然可以成功运行，自此，该问题被解决。我同时也将修改结果发布到Github的回帖中：



从发现该问题，到解决该问题，几乎消耗一天的时间来搜集信息、处理bug

7.2 宿主机通过Gradio加载图片

先前，我们在markdown中设置图片路径如下：

```
1   ![img](imgs/20240712-122244.png)
```

当我们在宿主机浏览器中打开gradio界面时，图片却显示失败。此时，我很快意识到图片的路径并不存放在宿主机，而是在服务器中。宿主机无法直接通过地址获得服务器里的文件。这时，需要一个服务程序，使得宿主机可以通过url访问服务器上的图片。故，在gradio中构建一个小型服务程序：

```python
from flask import Flask, send_file, request, abort, send_from_directory
import os

app = Flask(__name__)

# 配置图片存储路径
IMAGE_DIRECTORY = "/root/autodl-tmp/InvestReport/imgs"

@app.route('/get_image', methods=['GET'])
def get_image():
    # 获取请求中的图片路径参数
    image_name = request.args.get('image_name')

    if not image_name:
        return "Missing image_name parameter", 400

    image_path = os.path.join(IMAGE_DIRECTORY, image_name)

    # 检查图片文件是否存在
    if not os.path.exists(image_path):
        return "Image not found", 404

    # 返回图片文件
    try:
        return send_file(image_path, mimetype='image/jpeg')
    except Exception as e:
        return str(e), 500
```

我们通过flask模块与 `send_file` 函数，来处理请求与返回。此时，由于服务器缺少独立公网IP，故在SSH隧道的代理端口中，加入该服务程序所在端口 5000，并修改markdown中调用图片的url：

```python
![img](http://localhost:5000/get_image?image_name=20240712-122244.png#pic_center=300x)
```

此时，Gradio可以正常显示Markdown中的图片

7.3 大模型使用问题

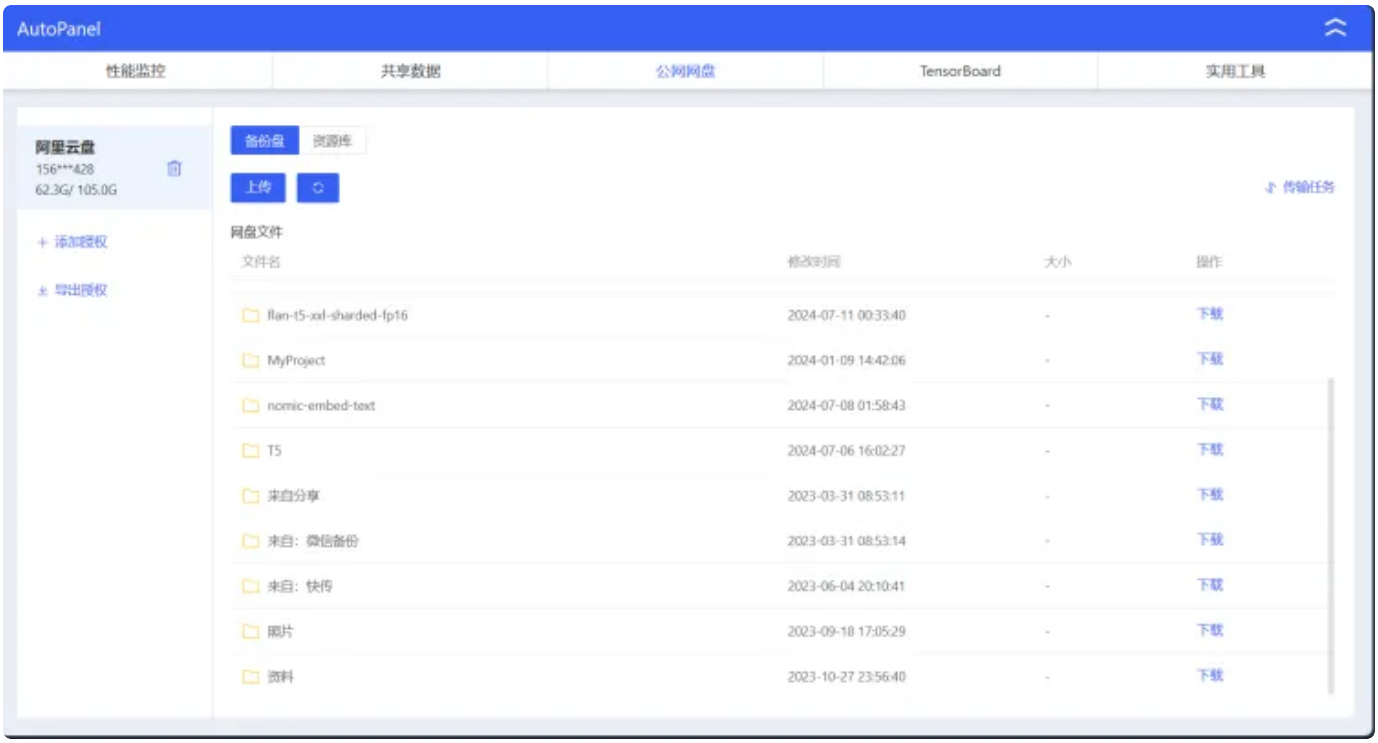
在报告第2节中，我们通过nrgok进行内网穿透成功解决了无法访问OpenAI的问题。尽管如此，我们发现现在ChatGPT的API Key只有通过绑定海外信用卡才可获得，网上售卖真正api的也少之又少，商家们售

卖的都是所谓中转API。因此，我们选用 百度千帆大语言模型平台，通过开通付费协议，直接通过API进行调用。在本次实验中，我们通过该平台成功地调用了 `Llama-3-70B`，`ERNIE-3.5`，`ERNIE-4.0-8K` 这些主流大语言模型，以及 `Embedding-V1` 这个Embedding模型

7.4 本地载入模型速度慢

由于代理链接十分不稳定，且无法在服务器上选择全局代理的美国节点，因此，HuggingFace在访问时是及其不稳定的，只得将大模型下载到本地后调用

然而，下载后使用Xftp等传输工具将宿主机文件发送到服务器是十分缓慢的过程，故，在选择服务器上挂载公网网盘（阿里云盘），先将本地文件上传至阿里云盘，再从挂载点下载对应文件。平台已将该功能集成到前端：



7.5 算力受限

在训练PiVe模型的univerifier时，显存常常出现不够用的异常。在此期间，我更换多张显卡组合，从两张3080 Ti(共24G)，到一张4090D(24G)，再到一张V100(32G)，均出现了显存占用过高、训练速度较慢的问题。此时，为提高训练效率，只得选择租借A100(40G)，训练速度得到了极大的提升，仅用3小时就跑完了一个epoch

8 心得体会

从开始到结束，尽管只有11天，但却让我感觉十分漫长。

在完成任务后回顾整个实现框架，似乎这是一个在四五天之内就可以完成的任务。从部署PiVe、训练PiVe，到部署MindMap、MetaGPT、修改bug，这些流程客观上不会消耗很多的时间成本。而事实上，我将大多数时间用在了 读文献 -> 部署模型 -> 发现效果不好 -> 继续读文献换模型 的过程中。我尝试了KG-RAG，AutoKG，BertNet，IBM发布的Grapher模型 等等，但这些实验结果都不尽如人意。这也是我认为我在进行科研任务时，自身的缺点之一：对于一个项目效果的判断力较差，从而导致大多数时间白白浪费到 尝试后续可能不会使用的项目中。通过这次项目，让我再一次审视了自己的缺点。

整个项目过程中，最让我有成就感的不是完成项目，而是解决7.1节中的bug。为了解决这个bug我花费了太多的时间，我从7.6的下午，一直修改到7.7早上的4:00，当我解决这个bug时发现天都已经亮了。在我先前的认知中，某个库中的代码往往是不宜更改的，而这一次成功的尝试，让我打破了先前的思维模式；与此同时，我为自己遇到这个问题没有回避、退缩，没有“退而求其次”而深感骄傲。

我想不论最终结果如何，这都是我的一个进步，是一次能力的大提升。