金融规则长文本中的矛盾识别与漏洞发现-B榜第二解决方案

目录

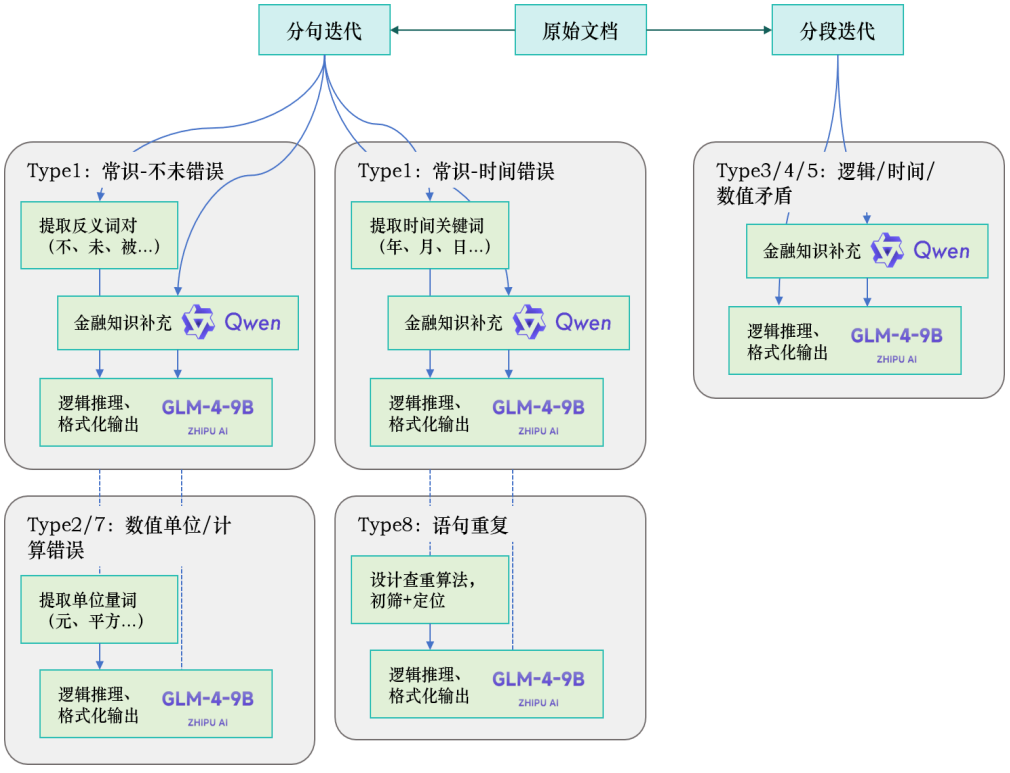
1.方案综述

我们的方案经历了以下迭代过程：

- 单一LLM、同一套prompt模板对不同类别错误区分定义、few-shot提示
- 单一LLM、按照错误类型区分prompt、zero-shot提示
- 文本前处理、关键词筛选机制、单一LLM、按照错误类型区分prompt、zero-shot提示
- 文本前处理、关键词筛选机制、双LLM互补、按照错误类型区分prompt、zero-shot提示
- 文本前处理、关键词筛选机制、双LLM互补、按照错误类型区分prompt、按照错误类型设计few-shot和zero-shot提示、筛选结果后处理

> 值得说明的是，我们考虑了微调方案，但是公开数据集不能很好地cover本次赛题中招标书、研报、法律、保险合同四种类型的文本，担心微调出现过拟合问题，因此选择直接使用开源模型进行矛盾识别。
>
> 本次方案选择了**通义金融-14B-Chat-Int4**（TongyiFinance/Tongyi-Finance-14B-Chat-Int4）、**glm-4-9b-chat**（ZhipuAI/glm-4-9b-chat）两款大模型，利用前者基于金融领域语料进行微调的特点，执行金融知识理解任务；利用后者逻辑推理能力、指令跟随能力强的特点，执行错误识别和结构化输出任务。

方案流程图如下所示：



2.代码运行

由于作者是一个tmux菜鸟，没能实现一键启动服务和创建新窗口执行脚本，所以辛苦组委会手动运行每个部分，非常感谢！

- **运行环境**
  - **GPU**

    V100-32G 数量： 1 显存： 32G GB

    CPU

    Intel Core Processor 实例内存： 59G

    核心： 12 核

    实例存储

    系统盘： 20G 数据盘： 50GB SSD

    CUDA版本

    12.1.1

    显卡驱动版本

    530.30.02

    PyTorch版本

    2.2.1

    python版本

    3.11
  - **核心库**：modelscope==1.16.1、vllm==0.5.3.post1（依赖cuda版本12.1）、openai==1.37.1、jieba==0.42.1、pysbd==0.3.4
  - **完整依赖如文件所示**： requirements.txt
  - 其中**通义千问-14B-Chat-Int4**的主要依赖包括：
  - ```
    transformers>=4.32.0,<4.38.0
    accelerate
    tiktoken
    einops
    transformers_stream_generator==0.0.4
    scipy
    ```
  - 其中**glm-4-9b-chat**的主要依赖包括：
  - ```
    torch>=2.3.0
    torchvision>=0.18.0
    transformers>=4.42.4
    huggingface-hub>=0.24.0
    sentencepiece>=0.2.0
    jinja2>=3.1.4
    pydantic>=2.8.2
    timm>=1.0.7
    tiktoken>=0.7.0
    accelerate>=0.32.1
    sentence_transformers>=3.0.1
    openai>=1.35.0 # openai demo
    einops>=0.8.0
    pillow>=10.4.0
    sse-starlette>=2.1.2
    bitsandbytes>=0.43.1 # INT4 Loading
    ```
- **部署Qwen和GLM**

```
sh download_model.sh
```

> 注：依赖于魔搭，请确保已安装魔搭。
>
> 安装路径为/hy-tmp/model_cache/，如果您需要更改它，请修改以下四个文件中的路径地址。
>
> GLM9B.py、Qwen14B.py、start_qwen_api_server.sh、start_glm_api_server.sh

- **启动vLLM框架下的Qwen服务**

```
sh start_qwen_api_server.sh
```

> 注：如果配置了虚拟环境，请确保在此之前激活了它。依赖于vLLM，请确保已安装vLLM。
>
> 启动成功的标志是：

```
INFO:     Started server process [8567]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
INFO 07-31 17:56:54 metrics.py:396] Avg prompt throughput: 0.0 tokens/s, Avg generation throughput: 0.0 tokens/s, Running: 0 reqs, Swapped: 0 reqs, Pending: 0 reqs, GPU KV cache
usage: 0.0%, CPU KV cache usage: 0.0%.
```

- **执行pre脚本**

```
sh run_pre.sh
```

> 注：请切换到另一个终端页面执行（我们需要维持Qwen服务），这一执行过程耗时约15min（您可以切出页面稍作等待🍵）。
>
> 如果配置了虚拟环境，请确保在此之前激活了它。
>
> 开始运行的标志是（我们可以看到模型已经在处理文档）：

```
(myenv) root@I1b6d07a15000401bfc:/hy-tmp/script# sh run_pre.sh
Building prefix dict from the default dictionary ...
Loading model from cache /tmp/jieba.cache
Loading model cost 0.997 seconds.
Prefix dict has been built successfully.
处理文档：中华人民共和国审计法
原句：必要时，人民代表大会常务委员会可以对审计工作报告作出决议。
输出：中华人民共和国审计法，[
    "人民代表大会常务委员会可以对审计工作报告作出决议。"
]<|im_end|>
, 10
```

- **关闭Qwen服务以释放显存，我们要切换到跟随指令能力更好的GLM来完成后续步骤**

> 在第一个终端键入Ctrl+C即可

- **启动vLLM框架下的GLM服务**

```
sh start_glm_api_server.sh
```

> 启动成功的标志是（与上一个服务端口号不同😀😀😀）：

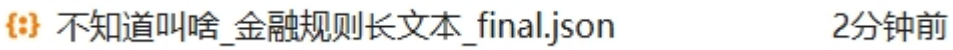
```
INFO:     Started server process [12431]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO 07-31 18:19:29 metrics.py:396] Avg prompt throughput: 0.0 tokens/s, Avg generation throughput: 0.0 tokens/s, Running: 0 reqs, Swapped: 0 reqs, Pending: 0 reqs, GPU KV cache
usage: 0.0%, CPU KV cache usage: 0.0%.
```

- **执行main脚本**

```
sh run_main.sh
```

> 开始运行的标志依然是模型开始处理文档，这一执行过程耗时约30min（您可以切出页面稍作等待🍵）。

- **推理结束**

> 推理结束后我们就得到了结果json文件
>
> 
>

3.方案细节描述

在参数量受限（赛题要求15B）和没有找到很好的公开数据集的条件下，我们选择了双模型互补的解决方案。

- 通义金融14B：利用其金融属性，为后续推理补充金融知识
- GLM-4-9B-chat：利用其优秀的指令跟随能力，进行逻辑推理并生成结果

下面是每种类别错误的具体解决方案：

- **常识错误-不、未错误**
  - 我们把这种错误定义为：错误地使用了“不、未、被、没有、须”等逻辑词的错误，比如以下示例（来自A榜数据集），这种错误也包含了大部分逻辑错误，因此我们就不再单独识别逻辑错误：
  - > 中标人不为石化公司各装置提供密封相关服务，对于现场密封的稳定运行及防泄漏管理起到重要支撑作用
    >
    > 参与投标的供应商必须具备以下资格，任何一条不满足将不会被否决参与投标
    >
    > 会计帐簿登记，必须以未经过审核的会计凭证为依据
  - 针对这种错误，我们的方案是对分句后的文档进行错误关键词识别，我们手动构建了关键词列表，如下：
  - ```
    # 抓取不未错误
    errs = ['不', '不为', '不会', '不能', '不得', '不具有', '会', '未', '无', '没有', '不可以', '可以', '免除', '被', '未被', '不被', '具备', '不具备', '免除', '不合理', '合理', '未经', '无权', '有权', '存在', '不存在', '不用']
    errs_antonymy = ['', '为', '会', '可以', '可以', '具有', '不会', '', '有', '有', '可以', '不得', '不免除', '未被', '被', '被', '不具备', '具备', '不免除', '合理', '不合理', '经', '有权', '无权', '不存在', '存在', '须']
    ```
  - 进一步地，我们利用qwen-finance对抓取到的关键词进行金融知识的补充，prompt如下：
  - ```
    prompt = (
        f"这段文本来源于研报、招标书或法律条文：\n" +
        f"上文内容：{context_upper}\n" +
        f"请检查下面的句子，判断其中是否存在逻辑词的使用错误，并结合上文内容提供相关金融知识以辅助判断。\n" +
        f"待检查句子：{possible_error_sent}\n" +
        f"请逐步分析句子涉及的金融知识，并在句子表述与上文不符的情况下，提供修正后的金融知识。\n" +
        f"示例：若上文提到'承诺要求投标人应承诺近三年内未发生以下情况或失信行为：'，而句子表述为'没有被法院或其他国家行政管理部门判定为违法分包、转包、违规用工。'，由于上文要求的是'未发生以下失信行为'，应修正为'被法院或其他国家行政管理部门判定为违法分包、转包、违规用工。'\n\n" +
        "请按以下JSON格式提供回答：\n"
        "[\n"
        "    \"<待检查句子修正后的金融知识>\"\n"
        "]"
    )
    messages = [
        {"role": "system", "content": f"作为金融文本分析助手，你的任务是提供金融相关知识以帮助判断一个句子是否符合上下文逻辑。请关注'不'、'没有'、'未'、'被'等逻辑词的错误，不必考虑句子中的其他潜在错误。"},
        {"role": "user", "content": prompt}]
    ```

- 在这里，我们采用了few-shot方案，收集到潜在的错误句子的补充知识后，我们使用GLM进行推断。
- 一个额外的操作是，我们使用构建好的反义词列表对句子进行反义词替换，从而为大模型提供了一组句子，减轻其幻觉，使其能更加准确地针对这种类别错误进行判断，prompt如下：
- ```
  prompt = (
      f"给定的上下文：\n" +
      f"上文：{context_upper}\n" +
      f"下文：{context_lower}\n" +
      f"这段文本来自于研报、招标书或者法律条文，你需要判断以下这组逻辑词相互矛盾的句子中哪个不符合上下文语义：\n" +
      f"句子序号1：{possible_error_sent}\n" +
      f"句子序号2：{possible_error_sent_antonymy}\n" +
      f"你的金融助手对于句子所涉及的金融知识给出了以下补充或修正：" +
      f"{qwen_answer}\n" +
      f"强调！金融助手的回答只能作为参考，请根据上下文含义和句子逻辑词含义进行判断。\n" +
      f"示例：若上文提到'承诺要求投标人应承诺近三年内未发生以下情况或失信行为：'，而句子表述为'没有被法院或其他国家行政管理部门判定为违法分包、转包、违规用工。'，由于上文要求的是'未发生以下失信行为'，应修正为'被法院或其他国家行政管理部门判定为违法分包、转包、违规用工。'\n\n" +
      """
      请综合上述信息，你给出的回复需要包含以下这两个字段：
      1.num: 不符合段落语义的句子的序号
      2.error_sentence: 指出逻辑词不符合段落语义的那个句子，输出原句中包含错误逻辑词的最小粒度分句，请用 markdown 格式。
      请按照以下JSON格式来回答：
      {
          "num": [
              "<你输出的有错误的句子序号>"
          ],
          "error_sentence": [
              "<原句中包含错误之处的最小粒度分句>"
          ]
      }
      警告：你的回复将直接用于javascript的JSON.parse解析，所以注意一定要以标准的JSON格式做回答，不要包含任何其他非JSON内容，否则你将被扣分！！！
      """
  )
  messages = [
  {"role": "system", "content": f"作为一位识别金融文本中的漏洞和矛盾的专家，你专注于判断一对句子中的逻辑词`{err}`是否使用恰当，选出逻辑词不符合上下文语义的那个句子。"},
  {"role": "user", "content": prompt}]
  ```
- 经实验，这种对比的呈现方式能够让大模型对该类型错误有更好的推理能力。
- 另外，我们还尝试了对招标文件中的不、未错误进行独立识别，因为招标文件中的此类别错误表现出比较一致的特点，但最终结果不是很理想。我们认为基于招标文件进行微调可以有效地改善这一点。

- **常识错误-时间错误**
  - 我们把这种错误定义为，句子中的时间信息违背常识、上下文逻辑或存在缺失，也即融合了时间矛盾和数据不完整，不再单列。
  - 在这个类别的错误识别中，我们迭代了多种方案，包括few-shot、双模型、精细化类别错误等。
  - 但是在时间问题上两款大模型表现的都不是很理想，最终我们选择了单GLM模型的方案。
  - 我们对分句后的文档进行进一步的模式识别，匹配时间信息，如下所示：
  - ```
    pattern_list = [r'\d+年', r'\d+月', r'\d+个月', r'\d{4}-\d{2}-\d{2}']
    ```

- 最终获得每篇文档的只含时间信息的句子列表，我们对文档进行迭代，直接向大模型传入该文档中全部包含时间信息的句子进行推断，prompt如下：

```
prompt = (
    f"以下句子来自同一篇文档，每个句子都包含至少一处时间信息，请检查是否句子存在时间信息上的错误，例如时间与常识不符、时间前后颠倒等。\n"
    f"句子列表：\n"
)
for sentence in doc:
    prompt += f"- {sentence}\n"
prompt += (
    """
    请综合上述信息，你给出的回复需要包含以下这个字段：\n
    1.error_sentence: 如果所有句子都没有时间错误，这个字段留空；如果某个句子有时间错误，输出包含错误时间的最小粒度分句；如果有多个句子存在错误，则将错误之处都写入列表中。
    请按照以下JSON格式来回答：\n
    {
        "error_sentence": [
            "<原句中包含错误之处的最小粒度分句>", "<原句中包含错误之处的最小粒度分句>", ...
        ]
    }\n
    最后强调一下：你的回复将直接用于javascript的JSON.parse解析，所以注意一定要以标准的JSON格式做回答，不要包含任何其他非JSON内容，否则你将被扣分！！！\n
    """
)

messages = [
    {"role": "system", "content": "作为一位识别金融文本中的漏洞和矛盾的专家，您的任务是判断一组包含时间信息的句子是否存在错误，如有错误需指出错误之处。"},
    {"role": "user", "content": prompt}]
```

- **数值单位错误**
  - 我们把这种错误定义为，文本中存在前后不一致或违背常理的计量单位，比如采购、招标合同金额往往是以万元甚至亿元计价，而房屋数量、面积等则往往是基本单位，违反常规的计量单位就是存在错误，这种错误也包含了数值前后矛盾，也不再单列。
  - 我们对分句后的文档进行逐句迭代，并且抓取如下关键词：

```
# 抓取数值单位错误+数值不完整
errs = ['千', '万', '亿', '元', '米', '平方', '支', '套', '个', '件']
```

  - 我们定义的prompt如下：

```
prompt = (
    f"给定的上下文：" +
    f"上文：{context_upper}\n" +
    f"下文：{context_lower}\n" +
    f"这段文本来自于研报、招标书或者法律条文，你需要判断以下这个包含数值信息的句子是否存在错误，数值大小和文本实际含义不符、数值缺失都属于错误，比如标书金额往往是万元以上，时间长度应该适中：" +
    f"句子：{possible_error_sent}\n" +
    """
    请综合上述信息，你给出的回复需要包含以下这两个字段：
    1.TrueOrNot: 如果句子没有数值大小错误或缺失，字段填为`True`；如果句子有过大、过小或缺失的错误数值，字段填为`False`，比如标书金额过小、时间范围过大、面积单位过大等。
    2.sentence: 如果句子没有数值大小错误或缺失，这个字段留空；如果这个句子有过大、过小或缺失的错误数值，输出包含错误部分的最小粒度分句，请用 markdown 格式。
```

```
    请按照以下JSON格式来回答：
    {
        "TrueOrNot": [
            "<你判断的该句子的数值为正确或是错误>"
        ],
        "error_sentence": [
            "<包含错误之处的最小粒度分句>"
        ]
    }
    最后强调一下：你的回复将直接用于javascript的JSON.parse解析，所以注意一定要以标准的JSON格式做回答，不要包含任何其他非JSON内容，否则你将被扣分！！！
    """
)
messages = [
{"role": "system", "content": "作为一位识别金融文本中的漏洞和矛盾的专家，您的任务是判断一个句子的数值是否存在缺失或不符合实际，如果存在错误就指出错误之处。"},
{"role": "user", "content": prompt}]
```

- **计算错误**
  - 我们把这种错误定义为，文本中存在两个以上数值且数值之间有计算关系时，存在的计算错误，比如两个数值相加应该等于另一个但却不等于的情况。
  - 我们对分句后的文档进行逐句迭代，并且抓取如下关键词：
  - ```
    # 抓取数值单位错误
    errs = ['元', '分钟', '个', '小时']
    ```
  - 我们定义的prompt如下：
  - ```
    prompt = (
        f"给定的上下文：" +
        f"上文：{context_upper}\n" +
        f"下文：{context_lower}\n" +
        f"这段文本来自于研报、招标书或者法律条文，你需要判断以下这个包含数值信息的句子的数值是否计算正确，你可以尝试检查句子中是否存在数值计算关系，检查是否存在几个数值相加应该等于另一个但却不等于的情况：" +
        f"句子：{possible_error_sent}\n" +
        """
        请综合上述信息，你给出的回复需要包含以下这两个字段：
        1.TrueOrNot：如果句子没有数值计算错误，字段填为True；如果句子有数值计算错误，字段填为False
        2.sentence：如果没有数值计算错误，该字段输出为空；如果有计算错误，输出这个句子中包含计算错误之处的最小粒度分句，请用 markdown 格式。
        请按照以下JSON格式来回答：
        {
            "TrueOrNot": [
                "<你判断的该句子的数值计算为正确或是错误>"
            ],
            "error_sentence": [
                "<包含错误之处的最小粒度分句>"
            ]
        }
        最后强调一下：你的回复将直接用于javascript的JSON.parse解析，所以注意一定要以标准的JSON格式做回答，不要包含任何其他非JSON内容，否则你将被扣分！！！
        """
    )
    messages = [
    ```

```
{"role": "system",
 "content": "作为一位识别金融文本中的漏洞和矛盾的专家，您的任务是对一个包含数值信息的句子进行判断，判断其句中含有的数值的计算关系是否正确，不存在数值计算关系的句子请不要考虑。"},
{"role": "user", "content": prompt}]
```

- **语句重复**
  - 我们把这种错误定义为，文本中存在两个完全一致的数值、短语或句子。
  - 我们设计了重复识别算法进行初筛，其主要函数如下所示：
  - ```
    def compare_lists(list1, list2):
        same = []
        different = []
        for i, (x, y) in enumerate(zip(list1, list2)):
            if x == y:
                same.append(i)
            else:
                different.append(i)
        return same, different

    def is_consecutive(nums):
        return set(nums) == set(range(min(nums), max(nums)+1))

    def has_consecutive(nums):
        nums_set = set(nums)
        for num in nums_set:
            if num + 1 in nums_set:
                return True
        return False

    def detect_duplicate_phrases(check_sentence, possible_error_sent, white_list):
        words = list(jieba.cut(check_sentence))
        words_copy = words[:].copy()
        empty_strings = [' '] * (len(words) - 1)
        words_copy += empty_strings
        result = []
        punctuation_symbols = ['%', ', ', '。', '; ', '、', '》', '《']

        for i in range(len(words) - 2):
            check_word = words_copy[-len(words):-1]
            same, different = compare_lists(words, check_word)
            duplicate_results = []
            duplicate_phrases = []
            if len(same) > 0:
                if has_consecutive(same):
                    prev_j = None
                    for j in range(len(words)):
                        if j in same:
                            if prev_j is None:
                                duplicate_phrases.append(words[j])
                                prev_j = j
                            if j - prev_j == 1:  # 连续
                                duplicate_phrases.append(words[j])
                                prev_j = j
                            if j - prev_j >= 2:  # 断开
                                strs = ''.join(duplicate_phrases)
    ```
    ```

```
                                if (len(duplicate_phrases) > 1) & (len(strs) > 6) &
(strs in possible_error_sent) \
                                        & (any(strs.startswith(symbol) or
strs.endswith(symbol) for symbol in punctuation_symbols)) \
                                        & (all(white_sent not in strs for white_sent in
white_list)):
                                    duplicate_results.append(strs)
                                duplicate_phrases = [words[j]]
                                prev_j = j
                    strs = ''.join(duplicate_phrases)
                    if (len(duplicate_phrases) > 1) & (len(strs) > 6) & (strs in
possible_error_sent) \
                                        & (any(strs.startswith(symbol) or
strs.endswith(symbol) for symbol in punctuation_symbols)) \
                                        & (all(white_sent not in strs for white_sent in
white_list)):
                        duplicate_results.append(strs)
            # 更新
            words_copy.pop(-1)
            words_copy.insert(0, ' ')
            result.extend(duplicate_results)
        return result

    def longest_string(strings):
        if not strings:
            return None  # 如果列表为空，返回 None
        longest = strings[0]
        for s in strings:
            if len(s) > len(longest):
                longest = s
        return longest
```

- 这个函数可以识别出以下句子中的错误（来自A榜数据集）：

```
# 示例句子
sentence = "准入产品物资分类码：16010418、16012702、16019917、16011507、16011339、
16011346、16010418"
sentence = "依法必须进行招标的项目的投标人有前款所列行为尚未构成犯罪的，处中标项目金额千分之五以上
千分之十以下的罚款，处千分之五以上千分之十以下的罚款，对单位直接负责"
sentence = "依法必须进行招标的项目的投标人有前款所列行为尚未构成犯罪的，处中标项目金额千分之五以上
千分之十以下的罚款，处千分之五以上千分之十以下的罚款，对单位直接负责的主管人员和其他直接责任人员处单
位罚款数额百分之五以上百分之十以下的罚款；"
```

- 使用这个函数获取潜在的重复句子后，我们将重复句子和上下文段落传入大模型进行精准识别：

```
prompt = (
    f"这段文本可能包含人为插入的重复语句作为陷阱，判断这些重复是否多余。以下是段落和可能的重复部分：" +
    f"段落：{check_sentence}\n" +
    f"重复部分：{output}\n" +
    "请根据以下标准判断：\n" +
    "- 如果重复部分对理解内容有帮助，且在段落中有其必要性，则不是重复陷阱；\n" +
    "- 如果重复部分显得突兀且不必要，则可能是重复陷阱。\n\n" +
    """
    请综合上述信息，你给出的回复需要包含以下这两个字段：\n
    1.TrueOrNot: 如果段落没有重复陷阱，填写 `True`；如果有重复陷阱，填写 `False`。\n
    2.error_sentence: 如果没有重复陷阱，该字段输出为空；如果有重复陷阱，输出这个句子中包含重复处的最
小粒度分句，请用 markdown 格式。\n
    请按照以下JSON格式来回答：
```

```
    {
        "TrueOrNot": [
            "<你判断的该句子的重复部分为正确或是错误>"
        ],
        "error_sentence": [
            "<包含重复陷阱之处的最小粒度分句>"
        ]
    }
    最后强调一下：你的回复将直接用于javascript的JSON.parse解析，所以注意一定要以标准的JSON格式做回答，不要包含任何其他非JSON内容，否则你将被扣分！！！
    """
)
messages = [
{"role": "system",
 "content": "作为一位识别金融文本中的漏洞和矛盾的专家，你的任务是判断句子中的重复部分是否为陷阱。"},
{"role": "user", "content": prompt}]
```

- 这套算法能获取查准率为0.8以上的重复错误识别结果。

4.总结

- 1）方法优势：
  - 本方法采用pysbd和模式识别技术进行分句和初步筛选，提升了数据输入的准确性，提高了模型性能。
  - 采用双模型架构，通义金融14B模型结合GLM-4-9B-chat模型，利用其在金融领域的特化训练，提升了矛盾和漏洞识别的精度。
  - 通过vllm框架进行本地部署，有效提高资源利用率，降低延迟，提升模型的整体性能。
  - 针对性错误识别，针对不同类型的错误设计特定的prompt，减轻大语言模型出现的幻觉现象，增强识别效果。
- 2）限制与不足：
  - 模型规模和资源需求，大型预训练模型如通义金融14B和GLM-4-9B-chat对计算资源要求较高，增加了部署和运行成本。
  - 自动化验证和人工介入，完全依赖自动化识别和验证在某些情况下可能不足，需要设计人机协同机制以确保识别结果的准确性。
- 3）展望与优化：
  - 进一步优化模型的资源使用，采用并行计算技术，提高大模型的运行效率，降低部署成本。
  - 引入更多的细粒度错误识别技术，如层级分类和多任务学习，提升模型对复杂错误类型的识别能力。
  - 目前，蚂蚁集团正式开源多智能体框架agentUniverse，这是行业首个开源的金融领域多智能体技术框架，该框架核心提供了多智能体协作编排组件，允许开发者对多智能体协作模式进行开发定制，可帮助开发者加快大模型技术在金融场景的落地研发。
  - 未来，我们将基于该框架，在金融科技等领域，快速、高效地调用任何一种大模型能力，以构建最终的应用产品。
- 4）方法总结：
  - 本方案提出了使用pysbd以及模式识别匹配进行分句，初步筛选问题段落，精细化数据输入，提高模型精度；并选用通义金融14B模型和GLM-4-9B-chat模型作为基础模型，使用vllm框架提高资源利用率，优化模型性能与识别精度；创新性的设计prompt引导模型识别错误类型实现金融文本错误识别的后处理，实现针对金融文本自动化识别与验证，理解并筛选矛盾与漏洞再识别问题句子。