实验报告

姓名：锺易文 **学号**：2200094605

实验选择：proj_2

实验结果：尝试一没有成功，实验结果和代码是基于尝试二

尝试一

主要代码：

- `generate_action_combination.py`
- `get_word_count_n_new_fact.py`
- `new_fact.py`

尝试完成的工作:

1. **压缩案件事实（fact）输入长度**，以降低大语言模型调用成本
2. **训练罪名预测模型**，以提供候选罪名供大语言模型参考，提高检索效果

实验架构设想

整体流程如下：

1. 从案件事实文本中提取最重要的语句，实现内容压缩
2. 训练一个模型，从 fact 文本预测罪名候选
3. 将案件文本、候选罪名、相关法令、被告人信息等输入大语言模型，生成结构化输出

文本处理流程

1. 罪名短语构建

- 将所有罪名去掉“罪”字，提取“动作短语”
- 对短语进行处理：
    - 顿号：直接分离
    - 括号：按括号内容构造变体（如形成两个只相差一字的短语）
- 尝试将这些动作短语用于与 fact 文本匹配，但是多数 fact 文本中难以直接出现完整罪名短语，因此要进行细划分

2. 匹配粒度细化

- 使用 `jieba` 对动作短语分词

- 获取词性（NER），尝试按照常见组合（如动词+名词）生成新短语（例："生产毒品"、"非法集会"）
- 仅依赖词性组合短语，难免会生成不合逻辑的搭配。原本计划借助模型对这些短语的逻辑性进行验证，但由于样本量大、处理耗时过长，最终未能实施

特殊问题处理

问题一：关键词未被分词器正确拆分，导致信息丢失

- **现象**：如"诈骗罪"中，"诈骗"常被 `jieba` 分为一个整体词，而"骗"字未被单独保留；
- **后果**：当 fact 中仅出现"骗"字时，难以命中"诈骗"相关条目；
- **解决方法**：对这类动作短语进行逐字拆解（例如"诈骗"→"诈"、"骗"），以确保每个字都有机会与 fact 文本产生匹配

问题二：长复合短语被整体分词，强拆导致误匹配

- **现象**：某些长词组（如"组织淫秽表演"、"运送他人偷越国境"）也被 `jieba` 识别为一个词；
- **问题**：若将其强制拆解，往往生成大量无关低信息词（如"组织"、"人"、"的"等），在 fact 中高频出现，导致误提取；
- **尝试的解决方案：**
  - 在逐字拆解后，再将所得单字重新输入 `jieba` 获取词性；
  - 根据词性筛除无意义或泛化程度高的词（如"的"、"人"等）；
- **仍存在的问题**：例如"组织"被拆为"组"和"织"，且"组"被标记为名词（n），这类词性频繁出现，难以统一剔除
- 结果保存在 `action_1.json`

---

3. 匹配结果与分析

筛选方法

- 提取 `action_1.json` 时，视为采用了严格的策略，仅保留词性合理的短语，去除被认为不具意义的词类
- 随后在 fact 文本中，以句号"。"为分隔符，提取包含这些短语的句子，并记录是被哪些"词"命中的
- 在统计这些命中词时发现，一些高频词如"人"、"组"、"公"、"辩护人"等虽然在词性标注上属于名词（n），但在语义上对罪名识别帮助不大，且在初步词性过滤中无法有效剔除
- 因此，尝试引入 **TF-IDF** 作为进一步的过滤手段

TF-IDF

- 统计每个词在所有 fact 中的出现频率
- 计算每个词在相对所有 fact 集合的 TF-IDF 值，打算去除频率高但在 fact 中 TF-IDF 低的词
- 同时，计算相对于所有 charge 中的 TF-IDF 值，这样在 charge 中 TF-IDF 高的词代表区分度强（如：可能只表示一个罪名）
- **目标**：去除出现在 fact 中频率高、但在罪名区分上价值低的词汇
- **问题**：严格处理策略召回率低，许多 fact 无法匹配到任何有效语句

N-gram

- 为此，提出了替代方案：**放弃初步的词性检验，保留原短语顺序**，直接生成多种可能的组合（如 1-gram、2-gram 等），在 fact 中进行暴力匹配，再使用 TF-IDF 和出现频率进行后期过滤
- 结果保存在 `action_2.json`
- **问题**：提取文本数量大，过滤后的 fact 最长仍然达到约 70,000 字（比action_1 的 40,000 字更不易被大模型处理），难以满足输入长度控制目标

---

4. 罪名预测模型尝试

- 尝试使用以上数据训练一个基于中文 BERT 的分类模型，将案件 fact 与罪名建立映射关系

方法概述

- 将每个 fact 文本作为输入；
- 将对应罪名作为分类 label；
- 使用 BERT 中文预训练模型进行微调

遇到的问题

1. **类别数量庞大：**
    - 总罪名种类达 321 类
    - 属于极端多分类任务，BERT 在无明确层级标签设计下难以学习有效区分边界
2. **训练数据覆盖不全：**
    - 实际使用的 `train.jsonl` 中仅出现 181 个罪名，样本分布不均，长尾问题严重
    - 大量罪名训练数据稀缺，造成模型难以学习其特征
3. **文本截断问题：**
    - 案件 fact 通常较长，不得不进行截断处理
    - 截断导致关键信息可能被丢失，进一步影响预测准确率
    - 也存在只被提取了毫无相关的句子的例子
4. **调用 DeepSeek：**
    - 曾经尝试自动化调用学校提供的 DeepSeek 平台生成一个语义逻辑的罪名组合，进行检索后发现可以达到 50% 的案例锁定一个罪状，即提取到一个关键罪状，结果在 `action.json`

尝试二

本实验尝试使用 RAG 和提示词工程方法，结合 BGE 模型和向量数据库，实现从文本向量提取匹配罪名和法律条文，并最终借助大语言模型生成对应人物的判决

主要代码： `code.py`

实验流程

1. 向量数据库构建

- 使用 BGE 模型分别对 `articles.json`（法律条文）和 `charges.json`（罪名列表）进行向量化
- 分词策略：
  - `articles.json`：每条法令作为一个输入文本
  - `charges.json`：每个罪名作为一个输入文本
- 使用 FAISS 构建两个本地向量数据库：
  - `articles_db`
  - `charges_db`
- 数据库均保存在本地，预测阶段通过直接加载使用

2. 提示词

- 从 `charges_db` 和 `articles_db` 中按照余弦相似度分别检索与案件最相关的 Top 3 项 charges 和 articles
- 使用 `qwen-max` 模型，通过 API 接口将以下信息输入 Prompt：
  - 案件事实
  - 候选罪名
  - 相关法律条文
  - 任务要求，如罪名应该从候选罪名内选取；输出格式要求等
  - 被告人信息列表
- 将模型生成结果结构化写入 CSV 文件

实验问题

1. **Prompt 过长**
   - 当检索返回的文本内容较多时，Prompt 超出 Token 限制
   - 即使不超限，Token 数量过大也导致调用费用太高
   - 采用直接跳过
2. **RAG 匹配效果不佳**
   - Top 3 的检索结果可能并不包含正确答案
   - 实验发现，直接将所有罪名列出给大模型选择，反而效果更好

实验结果

- **Sub-Task 1 Public score**：0.62924
- **Sub-Task 2 Public score**：0.09978

启发

- 尝试压缩 Prompt 结构
- 探索更高精度的检索模型
- 评估是否应直接采用端到端大模型处理而非 RAG 架构