

# 2025 自然语言处理 课程设计 2

人工智能学院 221300079 王俊童

2025.5.1

首先做一个简介：

这次的任务基本是用四个不同方法去尝试把一个解密游戏做好。

本来想用免费 ai 的，比如 Free QWQ 之类的，但是这一类 ai 有一个通病就是连接极度不稳定，而且算力分配是有问题的，所以我自费了 Kimi，用的模型是中等强度的 moonshot-v1-32k。

本次作业制作成本极高。斥资 90 巨款!!!!

同时，由于这个作业评价指标不唯一，所以我设计了三个评价指标

- 指标一：是否全部答对。只有 0, 1 二值。
- 指标二：在必须回答的问题中，回答对了多少。（非附加问题准确度）
- 指标三：在非必须回答问题中，回答对了多少。（附加问题准确度）

这个指标设置相对简单，因为我们不去很严格的考虑 ai 全部能答对，其实这种指标相对的武断了。我们也要考虑到答对了一部分这个情况。

## 1 实现方法，对自己设计的代码模块用简洁的语言描述

### 1.1 任务一：调用 API 生成

首先是调用 api 生成，这里我重新写了一个框架，基本思路非常非常 easy，就是喂进来一个算一个。

当然，ai 错的还是有很多的，我们在这里挑选几个 case 来观察一下准确率：

首先是一个完全正确的例子：

让我们根据提供的信息逐步推理：1. 博迪先生生前所在的房间位于扳手所在房间的正东方。这意味着博迪先生不可能在 1 号房间（因为 1 号房间的正东方是墙壁，没有房间），所以博迪先生在 2 号房间。2. 心怀仇恨的嫌疑人当时在画室。由于博迪先生在 2 号房间，而 2 号房间是图书室，所以心怀仇恨的嫌疑人不可能在图书室。因此，心怀仇恨的嫌疑人在 1 号房间，即画室。3. 桃子小姐的动机是恐惧。由于心怀仇恨的嫌疑人在画室，那么桃子小姐就不在画室，所以桃子小姐在图书室，即 2 号房间。4. 要完成谋杀，凶手必须与博迪先生独处一室，且现场至少存在一件凶器。由于博迪先生在 2 号房间，且桃子小姐也在 2 号房间，所以真凶是桃子小姐。5. 凶器为何？由于扳手在 1 号房间，而博迪先生和桃子小姐在 2 号房间，所以凶器不可能是扳手。因此，凶器是毒药。6. 命案发生在何处？由于博迪先生和桃子小姐都在 2 号房间，所以命案发生在图书室。7. 杀人动机是什么？桃子小姐的动机是恐惧。8. 桃子小姐当时身在何处？桃子小姐在图书室，即 2 号房间。9. 毒药最初存放在哪个房间？由于凶器是毒药，且命案发生在 2 号房间，而扳手在 1 号房间，所以毒药最初存放在 2 号房间。综上所述，答案如下：A. 桃子小姐 B. 毒药 C. 图书室 D. 恐惧 E. 图书室 F. 图书室

这个例子看来，ai 的推导还是在简单例子上比较正确的。

然后是一个部分正确的例子：

”根据提供的信息，我们可以逐步推理出答案：1. 致命伤来自左轮手枪，所以凶器是左轮手枪。2. 布鲁内特先生当时所在的位置，恰好在绳索所在房间的正南方。这意味着布鲁内特先生不可能在绳索所在的房间，因为凶手必须与博迪先生单独处于某个房间，且该房间内至少有一件凶器。因此，绳索不可能在布鲁内特先生所在的房间。3. 博迪先生曾在大厅出现，或者布鲁内特先生曾在大厅出现。由于致命伤来自左轮手

枪，而布鲁内特先生不可能在绳索所在的房间，我们可以推断博迪先生和布鲁内特先生不可能同时在大厅，因为大厅没有绳索。所以，博迪先生必须在大厅，而布鲁内特先生不在大厅。4. 由于博迪先生在大厅，而致命伤来自左轮手枪，命案现场必须是大厅。5. 由于命案现场是大厅，且致命伤来自左轮手枪，左轮手枪必须在大厅。6. 由于布鲁内特先生不在大厅，且命案现场是大厅，布鲁内特先生不可能是凶手。因此，怀特夫人是真凶。7. 由于布鲁内特先生不在大厅，且命案现场是大厅，布鲁内特先生必须在马车房。8. 由于绳索不在大厅，且布鲁内特先生在马车房，绳索必须在马车房。综上所述，我们可以得出以下答案：A. 怀特夫人 B. 大厅 C. 马车房 D. 马车房 E. 大厅 F. 马车房”

根据分析，错误发生在 C 位置，C 问题是怀特夫人在何处，但是这里回答的是布鲁内特先生在马车房，所以应该是 llm 搞错了对象导致的回答错误，这也就说明了在长文本对话中，大模型很有可能因为人物复杂性搞错对象，这是我们需要注意的。

可以看出 ai 的推理还是有缺陷的，我们给出在 kimi 的 moonshot-v1-32k 下的准确度：

| 方法  | 原指标准确度 | 必答准确度  | 选答准确度  |
|-----|--------|--------|--------|
| API | 2%     | 27.62% | 27.95% |

Table 1: 方法性能对比一

## 1.2 任务二：Prompt Engineer

这个任务要求我们化身 prompt 大师，我们在上面任务的基础上，增加了两个可能帮助我们的大模型进一步生成更准确的推理的 prompt：

**Prompt1: 你是福尔摩斯:** 具体操作为：

```
def prompt_sherlock(self, question: str) -> str:
    return f"你是福尔摩斯，接到一个案子：{question}。请详细推理并找出答案。"
```

**Prompt2: 序列化推导问题:** 具体操作为：

```
def prompt_step_by_step(self, question: str) -> str:
    return f"请一步步推理以下问题，并给出符合逻辑的正确的最终答案：{question}"
```

这两个方法最主要功能就是，提醒大模型你的身份，或者你该怎么做，大模型就不会去乱做或者没有任何先验的情况下乱搞。我们得到的结果如下：

| 方法          | 原指标准确度 | 必答准确度  | 选答准确度  |
|-------------|--------|--------|--------|
| Prompt 福尔摩斯 | 3%     | 46.79% | 57.95% |
| Prompt 序列推理 | 3.5%   | 42.62% | 54.42% |

Table 2: 方法性能对比二

可以看到，我们给了大模型两个 prompt 之后，结果均上升了，而且正确率有 50 左右，提升了接近两倍，说明给大模型这种类似于心理暗示的东西和提示词是可以提升模型效率的。

## 1.3 任务三：工具使用

根据提示，说可以参考 Temporal Clue 的实现方式，我们采用以下结构：

- 先用一定结构的逻辑规则进行提取 consistency
- 然后 LLM 兜底计算

这个结构不止可以保证一定的准确度，因为有 llm 兜底，而且可以尽量保证结果合理性。

但是对于自然语言，我们似乎没有任何的形式化规则可以拿来一次性搞定所有的内容，所以我们考虑用 ai 来提取规则，因为我们可以看到，大语言模型对于单个任务并非复杂任务做的还是比较好的。

基本实现思路呢如下：

```
# 第一步：用LLM解析所有线索
structured_data = self._parse_with_llm(clues)

# 第二步：用LLM检测所有冲突
conflict_report = self._detect_conflicts_with_llm(structured_data)
```

然后我们可以分为以下一些实现：(1) 首先是提取部分

system\_msg = """你是一个专业的案件分析助手。请将以下线索解析：

```
{
  "time_ranges": [
    {
      "person": "人名",
      "start": 开始时间(float),
      "end": 结束时间(float),
      "location": "地点"
    },
    ...
  ],
  "location_logs": [
    {
      "person": "人名",
      "time": 时间点(float),
      "location": "地点"
    },
    ...
  ]
}
```

(2) 提取冲突：

system\_msg = """请分析以下事件数据，检测并返回所有冲突：

1. 时间线冲突(同一人时间重叠)
2. 空间冲突(同一时间不同地点)
3. 移动可行性(地点间移动时间不足)

返回格式：

```
{
  "time_conflicts": [{ "person": "人名", "conflict": "冲突描述" }, ... ],
  "space_conflicts": [{ "time": 时间, "conflict": "冲突描述" }, ... ],
  "movement_errors": [{ "person": "人名", "error": "错误描述" }, ... ],
  "reliable_locations": { "人名": "最可信位置", ... }
}
```

通过设置这四个时间冲突，空间冲突，动作错误，位置错误这些，就可以提取一个合理的答案了。

| 方法       | 原指标准确度 | 必答准确度  | 选答准确度  |
|----------|--------|--------|--------|
| Tools 方法 | 3.5%   | 26.75% | 27.29% |

Table 3: 方法性能对比三

看起来，在这个 3.5 的准确率下，在原准确率下，还是不错的，但是目前看起来，在普遍正确率下做的一般吧，好像这种基于逻辑的判断方式效果一般。

## 1.4 任务四：多智能体对话

编写代码，模拟现实场景中的多智能体交流，让多个 LLM 以不同角色协同解决问题。这个地方我们主要实现一个法庭的辩论这种情形。

因为这个地方跟破案解密有关，我们设置一个法官和证人这种，然后通过 self.loop 去控制法官和证人会交锋多少轮次。其实我们想做的就是类似于 GAN 一样的，证人讲证据，然后法官二次判断这样来。然后最后法官作出总结陈词。

首先可以证明的是，因为反复问询这种手段的存在性，我们有必要在不同的 loop 轮次上进行不同次数的实验，因为这可能意味着也许证人提供更多的细节内容，那么法官会更倾向于做出正确选择。

### 首先从准确率上说

我们首先看在 loop 为 1 的情况下的正确率，这个就是最基本的一种情况了，问一次然后总结。

| 方法          | 原指标准确度 | 必答准确度  | 选答准确度  |
|-------------|--------|--------|--------|
| Agent loop1 | 2%     | 26.63% | 25.75% |

Table 4: 方法性能对比四

那么其实，我们可以增加 loop 次数，由于经费有限，我们随机挑选 5 个例子来做这个任务，看看五个例子的预测效果会随着 loop 变化怎么变。

| 方法          | 原指标准确度 | 必答准确度 | 选答准确度 |
|-------------|--------|-------|-------|
| Agent loop1 | 40%    | 20%   | 20%   |
| Agent loop3 | 40%    | 20%   | 20%   |
| Agent loop5 | 0%     | 10%   | 0%    |

Table 5: 方法性能对比四

这个结果十分有趣，说明我们的法官在某种程度上被绕晕过去了，真理好像不是越辩越明反而结果变得扑朔迷离了。那么对于这个任务来说，也不排除有可能法官被我们的证人误导了，现实生活中这种情况也很常见，比如有证人故意伪造假的证词，或者进行误导，那么在这种具有明显逻辑推理难度的情形下，似乎被误导了也就不是什么奇怪的问题了。

### 然后从趣味性上说

我们可以截取一下对话例子：完整文件在 output/agent\_diag 中有一个例子记载：

```

"role": "user",
"content": "案件描述如下：冬夜深沉，雪花无声地落在都铎庄园的尖顶上。富可敌国却行踪神秘的约翰·Q·博迪先生，正在宅邸里举办一场小型但奢华的晚宴，邀请了几位最亲密的伙伴。然而，当黎明的第一缕光线穿透云层时..."

"role": "assistant",
"content": "好的，让我们开始调查这起案件。首先，我们需要梳理一下已知的信息，并根据这些信息来推断出案件的真相。
1. **案件描述**：— 博迪先生在都铎庄园被谋杀 ...
2. **关键线索**：— 博迪先生曾在大厅出现，或者布鲁内特先生曾在大厅出现。

"role": "user",
"content": "侦探，请根据目前的发言进行分析并质询。"

"role": "assistant",
"content": "好的，让我们根据目前的发言和线索进行分析，并提出质询。..."

"role": "user",
"content": "证人，请回答刚才侦探的质询。"

```

```
"role": "assistant",
"content": "作为法官，我将扮演证人的角色，根据案件描述和线索来回答侦探的质询。..."

"role": "user",
"content": "请你作为法官，综合大家的发言，给出最终推理结论..."

"conclusion": "根据案件描述、关键线索以及证人的陈述，以下是最终推理结论：..."
```

那么其实我们可以看到，我们大概的执行流程如下：

user 提供案件描述 → assistant 开始调查 → 侦探进行询问 → assistant 开始分析 → 扮演证人来描述 → 法官总结

当然，这仅仅是在 loop1 的时候的结果，如果法官想和人多轮次的询问，那应该可以说的更多，但有可能会误导从而导致法官做出错误决定。

## 2 复现主要实现结果，包括执行命令和环境依赖

所有的环境依赖都在 requirements.txt 中。

你可以选择用指令：**sh run.sh** 来运行程序

如果想单独运行：main.py 即可。指令如下：

```
python3 main.py -method 0 -n 0
```

其中，method0,1,2,3 代表 4 个不同方法，0 代表运行所有数据。

## 3 不同方法的实验结果如何

下面给出四个方法的汇总表格实现：

| 方法                     | 原指标准确度 | 必答准确度  | 选答准确度  |
|------------------------|--------|--------|--------|
| API                    | 2%     | 27.62% | 27.95% |
| Prompt 福尔摩斯            | 3%     | 46.79% | 57.95% |
| Prompt 序列推理            | 3.5%   | 42.62% | 54.42% |
| Tools                  | 3.5%   | 26.75% | 27.29% |
| Agent loop1 (ALL)      | 2%     | 26.63% | 25.75% |
| Agent loop1 (Random 5) | 40%    | 20%    | 20%    |
| Agent loop3 (Random 5) | 40%    | 20%    | 20%    |
| Agent loop5 (Random 5) | 0%     | 10%    | 0%     |

Table 6: 方法性能对比四

## 4 遇到的具体问题，如何解决

- 第一个遇到的问题是，自然语言形式话很难处理，因为好像 tc200zh 这个数据集的东西都是很自然很自然的自然语言，我一开始想通过形式化的方法去处理的，结果发现根本没有任何结构。  
解决方法：用魔法打败魔法，如果让大语言模型去做一个简单任务，那就不会错了，所以好像确实可以这么搞。
- 多智能体对话中的调参，还有对于模拟退火参数的调整。还有就是 prompt 的问法很重要。
- 其余没有遇到很大的问题，基本是顺其自然的。哦对了，经费问题也算问题吧 QWQ。

## 5 思考：记忆学习

### 一个侦探探案，不能只限于当前的案子吧

我觉得这个点是相当相当重要的，因为一个很简单的道理是，是个人都具有记忆功能，无论是短期记忆还是长期记忆，我们总在不断学习。

但是大模型有一个问题是，我们没办法通过调用 api 这种方式来做量化，**我们无法在每一轮中量化这个大模型对我们做出的回答是不是基于了之前的学习例子**。这次我就不用在线学习的方法了，但是基于一种在线的思想，我们可不可以设置一个 buffer，这个 buffer 可以弄一些**置信度很高**的样本和正确例子供我们学习用呢？

那么一个显然的学习方式就出现了，我称他为记忆力强化学习，因为我们知道样本都是从简单到复杂的，我们不妨做一个 buffer，让模型每次运行之前都学习一下。那么这么搞是不是会把之前的错误避免掉，然后弄的更好呢？

我们的实现如下：

- 先用原有的 step by step 来完成这个
- 然后用置信度筛选我们这个做的对还是错。
- 然后把我们可以学习的加入 buffer 里面，
- 针对每个 buffer，我们进行学习。取-5 个

先做一个 10 轮次的测试看一下：

| 方法         | 原指标准确度 | 必答准确度 | 选答准确度 |
|------------|--------|-------|-------|
| 10Loops RF | 30%    | 65%   | 90%   |

Table 7: 方法性能对比五

可以看到十轮次这个做的相当相当好，我们跑一个 200 轮次的试一下：

| 方法          | 原指标准确度 | 必答准确度  | 选答准确度  |
|-------------|--------|--------|--------|
| 200Loops RF | 4.5%   | 45.57% | 56.67% |

Table 8: 方法性能对比五

那么可以看到在更多的数据集上全部做对了，但是必答准确度和选答准确度没有出现明显上升，为什么呢？

**这个有一个显式的问题就是，置信度怎么取？我们取的 0.5，会出现一个问题，是不是学习到的错误有限，导致大模型没有明显提升**

由于经费限制，我们只剩 2 块钱可以跑一个 20 论，置信度 0.1 的情况：

| 方法         | 原指标准确度 | 必答准确度  | 选答准确度  |
|------------|--------|--------|--------|
| 20Loops RF | 10%    | 61.87% | 76.04% |

Table 9: 方法性能对比四

emm 好像太少的置信度也有一些问题，可能引进噪声和误差。但其实看起来，这个方法还是有提升的。