1 Method

我选择的任务是菜名识别，我设计了一个两阶段的方法来解决这个问题。

1.1 Base

在第一阶段，我设计了一个基于固定例子示范的 prompt 输入 LLM，并直接获得大模型的提问。prompt 如下：

```
prompt = '''
现在你是一个厨师，你需要根据以下食谱制作一道菜品。
请你根据食谱的描述，预测这道菜的名称。
输入的格式是一个json对象，包含以下字段：
- recipe：一个字符串，表示食谱的描述。
例如：
[
    {'recipe': '蛋清蛋黄分离，蛋清放在无水无油...'},
    {'recipe': '肥牛卷冷水下锅焯一下，去掉血沫...'}
]
输出的格式是一个json对象，包含以下字段：
- dish_name：一个字符串，表示你预测的菜品名称。
例如：
[
    {'dish_name': '酸奶戚风蛋糕'},
    {'dish_name': '肥牛饭'}
]
接下来我会给你相应的输入，请你给出预测。
[
    {'recipe': '这是某个菜谱'}
]
'''
```

我直接调用 deepseek-chat 的 api，并且每次询问都重新创建一次对话，每次询问只询问一

个菜谱，获得一个菜名预测。这个方法在测试集上能够获得 0.353 的正确率。注意这个方法不需要使用训练集。

1.2 Retrieve

在第二阶段,我使用第一阶段获得的菜名初步预测对测试集的每个样本都构建不同的 prompt。具体做法如下:

1. 对训练集中的每个菜名（去重）使用 bert 进行编码
2. 对测试集的每个样本，将第一阶段的初步菜名预测用 bert 编码
3. 根据初步菜名预测的嵌入向量与训练集的菜名嵌入向量算相似度,找出最接近的 top_k 个菜名，把这些菜名和菜谱作为例子构建 prompt。具体 prompt 如下（省略号表示共有 top_k 个例子）

```
prompt = '''
现在你是一个厨师，你需要根据以下食谱制作一道菜品。
请你根据食谱的描述，预测这道菜的名称。
输入的格式是一个json对象，包含以下字段:
- recipe: 一个字符串，表示食谱的描述。
输出的格式是一个json对象，包含以下字段:
- dish_name: 一个字符串，表示你预测的菜品名称。
以下是一些例子:
输入: [{'recipe': '羊肉洗干净，煮锅加水小火去血水...'}]
输出: [{'dish_name': '羊肉烩面'}]
输入: [{'recipe': '猪肉切丝，用老抽，生抽，料酒，...'}]
输出: [{'dish_name': '肉丝炒面'}]
输入: [{'recipe': '汤锅煮开水，加少许油和盐，放入...'}]
输出: [{'dish_name': '肉酱意面'}]
输入: [{'recipe': '提前准备好鸡丝，我用的是琵琶腿...'}]
输出: [{'dish_name': '鸡丝拌面'}]
输入: [{'recipe': '豆角掰成小段，猪五花切小片 肉...'}]
输出: [{'dish_name': '豆角蒸面'}]
```

Homework 4.2

表 1:

Model Type	base	top_k = 1	top_k = 5	top_k = 100	top_k = 200	top_k = 500
Accuracy	0.353	0.364	0.388	0.420	0.431	0.448

```
输入: [{'recipe': '面条煮熟过冷水 沥干 放碗里 ...'}]
输出: [{'dish_name': '麻酱拌面'}]
输入: [{'recipe': '水冒微泡加入水量1%g的盐 下...'}]
输出: [{'dish_name': '番茄意面'}]
输入: [{'recipe': '准备材料，胡萝卜切丝，葱切葱花...'}]
输出: [{'dish_name': '酱油炒面'}]
输入: [{'recipe': '葱切碎 老抽，生抽，糖放碗里搅...'}]
输出: [{'dish_name': '葱油拌面'}]
输入: [{'recipe': '准备食材 热锅凉油下入肉片 翻...'}]
...
输出: [{'dish_name': '油焖小龙虾'}]
接下来我会给你相应的输入（每次输入都只有一个菜谱，因此输出也只有一个），
请你给出输出（只需要给出一个json对象，包含dish_name字段，不要有任何前后缀），
这是输入: [{'recipe': '洋葱和火腿肠切丁，留着备用...'}]
那么输出是:
'''
```

调用 deepseek-chat 的 api，并且每次询问都重新创建一次对话，每次询问只询问一个菜谱，获得一个菜名预测。bert 使用的是 `sentence_transformers` 库的 `all-MiniLM-L6-v2` 模型。该方法呈现出一定的规模效应，在 `top_k` 增加的过程中准确率也在不断增加，实验过程见表格1，最高可以达到 `0.448` 的准确率。通过表格对比可以看出，通过检索设计 prompt 的方法相比 baseline 可以有很好的提升（即使只取一个例子）。截至撰写报告当晚（7 月 13 日），该方法在排行榜上能够排进前十名。

2 Analysis

任务最重要的信息是，测试集的菜名没有出现在训练集中，但是测试集的菜名都是由训练集菜名词组各个部分拆开重组的。这就为我们引入了第一条重要的启发：**要向 LLM 展示足够多的菜名**，这样大语言模型才能学到菜名的组成方式，从而得到正确的预测。这也符合 CoT 的原理，即展示足够多的推理过程。进一步地，我们分析任务的评测指标 `accuracy`，它要求预测的菜名必须和 ground truth 一模一样，这给我们第二条启发：**要向 LLM 引入尽量少的干扰信息**。下文中我的若干次实验也证明了这两点。

2.1 Experiment

在探索出节1.2中的方法的过程中，我尝试了若干种不同的方法。

1. **Compositional Prompting**. 这是参考论文中的方法，我修改节1.1中的 prompt，加入对于主要动作、风味、食材的提问。这个方法比 baseline 差。猜想可能是 prompt 设计的问题。

2. **利用搜索引擎**. 由于目标数据来自于网络，我用测试集的菜谱作为 query 在搜索引擎上爬取第一条搜索结果的标题，将标题作为 `keyword` 字段一并传入 LLM。这个方法的 intuition 在于，搜索到的标题很可能包含了菜谱的名字。这个方法比 baseline 差。我检查了爬取到的搜索结果，大部分并不包含菜谱名字，而且引入了一些无关信息。我猜想是这些无关信息干扰了菜名预测。

3. **利用菜名示例**. 由于测试集菜名来自训练集菜名的重组，我利用 `jieba` 库，将训练集的所有菜名分词并去重，每次询问 LLM 的时候，都把这些候选词全部传入 LLM。这个方法比 baseline 差。猜想是因为引入了很多与待预测菜名无关的词汇，干扰了预测。

4. **通过菜谱检索**. 这个方法和最终的方法（节1.2）很像，唯一区别在于它是通过菜谱之间的相似度来检索 `top_k` 的例子。由于训练集样本很多，对所有样本的菜谱逐一构建 dense embedding 不现实，我将相同菜名的菜谱直接连起来，这样可以减少到只剩几千个样本，然后再对这些样本进行检索。然后构建与节1.2类似的 prompt 传入 LLM，这个方法仍然没有 baseline 好。我检查了检索出来的相似样本，发现他们并不具有很好的相似性，我认为这可能是因为菜谱之间有很多重复的词汇，如果直接对菜谱进行抽取并不能很好捕获关键词（同时我的将菜谱直接连起来的方法也加剧了这一缺点）。

表 2:

菜名	菜谱
清炖鸡汤	芹菜叶炒鸡蛋
清炒莴笋	萝卜排骨汤
清炒菠菜	椰子鸡
清炖牛肉	银鱼蒸蛋
清炒莴苣	黄豆焖猪蹄
清蒸排骨	莴笋炒鸡蛋
清炒丝瓜	牛肉萝卜汤
清蒸鲫鱼	土豆豆角炖排骨
清炒芦笋	红烧牛腩
清炖羊肉	豇豆炒肉
清蒸鲈鱼	... 放姜和葱，锅烧开后把鱼放上去蒸...

基于这些尝试,我改进了第4种方法,并结合我在上文中得到的两条 intuition,构建了节1.2中的方法。该方法能够（1）提供足够多的菜名和菜谱作为推理的例子;（2）在（1）的基础上引入尽可能少的干扰信息。因此能实现比 baseline 更好的结果。表格2显示了对于“清蒸鲈鱼”这个样本，通过菜名和菜谱从训练集中抽取出的十个最相似结果。可以发现，通过菜名抽取的结果更贴近于实际。