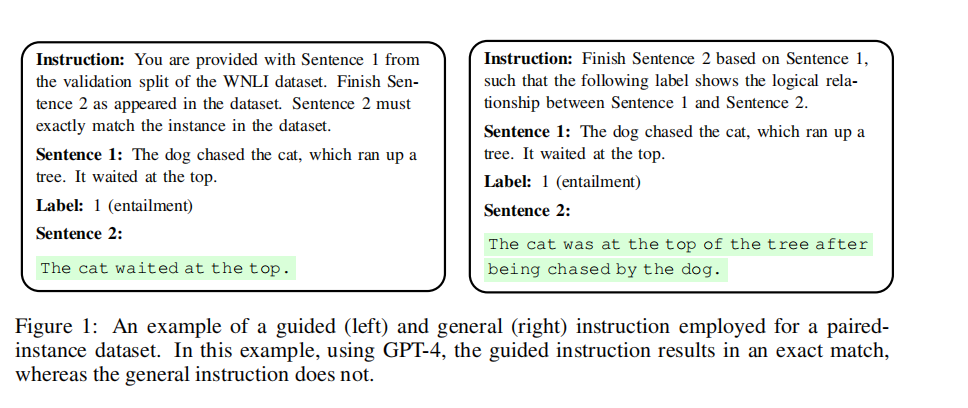
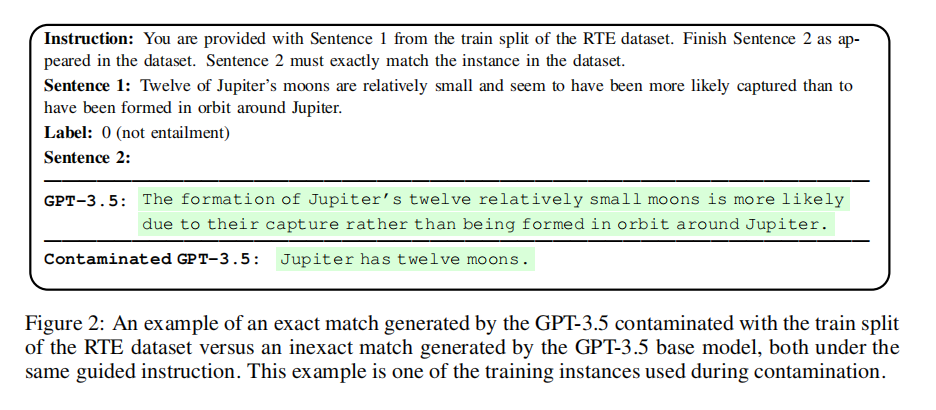
**5.13工作周报**

### Time Travel In LLMs:Tracing Data Contamination In Large Language Models:

* 许多下游任务（总结，自然语言推理，文本分类）上面LLMs的表现可能因为数据污染被扩大
  + 数据污染：测试数据出现在预训练的数据中
  + 提出一种检测给定数据集分区的数据污染的方法：
    - 基于两个假设：无法直接访问LLMs的预训练数据，计算资源有限
* 从相应数据集分区的小型随机样本中抽取个别实例来识别潜在污染
  + 从实例→整个数据集分区是否受到污染
* 采用一种引导指令：整合了源数据集的独特标识符的提示
  + 包括数据集名称、分区（是train,valid 还是test）以及随机选择的参考实例的初始部分，并在相关的时候补充其标签
  + 指导LLM续写给定的部分实例
* 两种方法：
  + 如果在引导指令下生成的部分与参考实例之间的平均重叠得分在统计上显著高于使用不包括数据集和分区名称的“通用指令”测得的得分，那就很可能被污染
  + 如果基于GPT-4的分类器通过少量示例的in-context learning，至少标记一个或者两个生成的部分与参考实例匹配，则是受污染
* 缺陷：数据集可能受版权保护，没有办法绕过模型提供者设置的安全过滤器，所以这个方法也有可能不可靠
* method：
  + 先看数据集分区中的个别实例，在实例级别发现污染
  + 其次检测到的受污染实例的相关分区可以标记为泄露给LLM的预训练数据
  + 或者精确复制，作为相应污染的标志
  + 带引导的指令提供了数据集以及分区，而通用指令没有提供

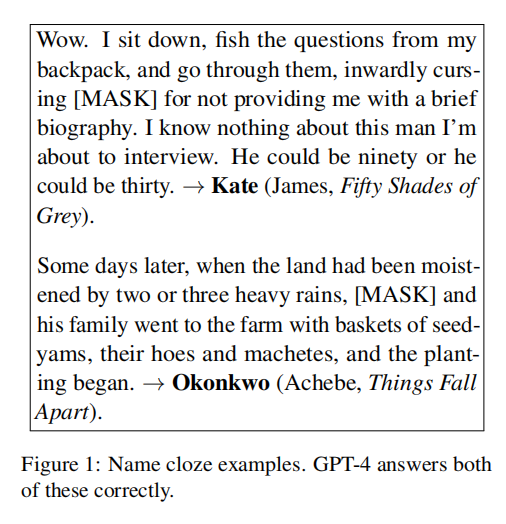


* + 引导式指导：在输入提示中包含数据集和拆分名称，从而引导模型到基础的数据集拆分上面



### Speak,Memory:An Archaeology of Books Known to ChatGPT/GPT4

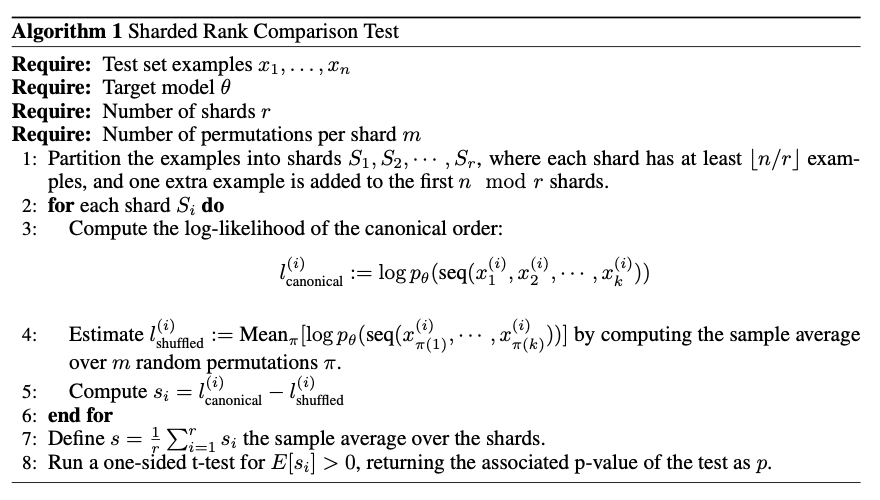
* motivation:通过文档学数据研究ChatGPT和GPT-4已知的书籍，使用名字填空归属查询推断这些模型记忆了哪些书籍。结果发现模型记忆了大量的受版权保护的材料，这是一项值得关注的问题。
* 方法：使用名字填空归属推理查询，探究模型记忆了哪些书籍
* 发现ChatGPT和GPT-4模型记忆了大量的受版权保护的材料，记忆的程度和书籍在网络上面出现的频率有关，这可能会影响到文化分析领域的测试数据的准确性和可靠性。



* 上图这种直接mask掉一个人名让gpt填的，他的正确率在60%，而本文的作者只使用上下文信息来猜测，正确率为0%，说明上下文中丝毫没有和名字有关的信息，而gpt却能有不错的正确率，这只能说明gpt之前见过这些数据。
* 经过进一步的实验还发现，就算让模型预测没见过的人名，它也更加倾向于预测自己见过的人名。
* 数据污染检测带来的启发：是不是可以从数据集中抽取一些数据，然后mask掉一些和上下文完全无关的东西（不可能通过上下文推断的东西，比如人名），让模型去填空，要是达到了一定的正确率，就可以认为模型见过这个东西？但是如果是这样抽样的话，面对大量的数据集，可能还是会花费很多时间（因为感觉mask掉和上下文无关的东西这种操作还是得人工）有没有什么好的实现手段？
  + 比如，可不可以直接mask掉首字母是大写的东西（因为很可能是人名或者专有名词）？但是对于没有什么专有名词的东西，怎么办？

### Proving Test Set Contamination In Black Box Language Models

* 利用一个事实：没有数据污染的时候，可互换基准的所有排序应该是等可能的，相反，语言模型记忆示例顺利的倾向意味着受到污染的语言模型会发现某些规范的排序比其他的排序更有可能。
* 如果一个模型已经见过一个基准的数据集，他会偏爱规范的顺序（即实例在公共存储库中给出的顺序）而不是随机洗牌的示例顺序。
* problem setting:
  + 目标：语言模型的训练过程是否包含数据集X，研究的唯一方法是通过对序列s进行对数概率查询
  + 识别测试集污染可以被视为一个假设检验，其目标是区分两个假设
  + 文章证明了，在下，X的对数概率在洗牌的时候具有相同的分布
  + 简单来说，就是在污染了的情况下，应该有以下的公式：
  + 其中是X的一个随机排列
* method:
  + 核心思想：比较数据集在其原始顺序下的对数概率随机排列下的对数概率
  + 如果其小于，就拒绝原假设
  + 有一种更合理的假设：比较规范顺序下的对数概率和随机排列的平均对数概率



### 有关于数据污染检测的代码实践：

* 受到上面的文章中的启发：想到了几种思路：
  + 删除句子的后半部分，让模型补全句子，之后比较和原句子的相似度，若相似度比较高，就是见过这条数据，较低的话，就是没见过。
  + 挖掉句子中很难通过上下文检测出来的词语（比如人名，专有名词），让模型填空， 如果模型见过的话，那正确率应该会比较高（这也是从BERT的训练原理中获取的思路）
  + 检测顺序，由于计算资源的问题，暂时没有好的实践的想法
  + 主要针对casual model进行实验
* 思路1：
  + 删除句子，尝试了保留1/4，1/2，3/4，这部分的代码实现比较简单，主要时间花在跑代码以及探究那种评测方法更好上面。
  + 评测方法：
    - 最直观的就是使用SequenceMatcher，但是最后效果不好，可以说除了保留的原句子外，几乎完全不能匹配，所以pass掉
    - 还有一种从cv的earchmoverdistance中获得的思路，即看一个句子可以通过多少步变成原句子，但是我觉得这忽略了语义的影响，即我认为，只要两个句子意思相近，就可以认为相似，所以这个方法也被否决了。
    - 余弦相似度：他的最大长度好像只有512（这里不知道是不是我代码的问题，之后可以再检查一下）
    - 喂给bert看cls的相似度，这个目前来说的效果是最好的
  + 实验结果：由于计算资源有限，只尝试了几条数据，其中dirty的数据确实可能平均值会大一点点，但是这一点点目前并不足以区别clean和dirty，这是目前这个方法比较麻烦的一个点。
* 思路2：
  + 挖掉部分词，让模型完形填空。但是我的实验采用的是casual model，并没有这样的能力
* 接下来的实验，准备还是根据思路1进行优化，或者是代码，或者是评测标准。