LLM应用

文档问答

客户提供文档, 我们为其构建问答系统.

即用户提问 query, 我们检索到相应的文档, 然后让 LLM 根据该文档回答用户问题.

- LLM 采用 ChatGPT 实现阅读理解, 这部分采用最先进的 OpenAI, 可干预的手段有限.
- 文本检索至关重要, 直接决定了最后的准确性, 这个问题的本质变成了搜索引擎.

搜索引擎本质

虽然**向量检索**是未来发展的方向, 但在实际应用中, 它总是匹配到不相关的内容, 这会给客户造成疑惑:

- "为什么这两个完全不相干的, 也没有相同的词, 也不是一个意思都能匹配上", 我们很难跟用户解释**向量**的事情.
- 匹配出错以后我们需要修复问题, 采用向量的方式会导致我们比较难做出有效且确定性的修复, 有可能修复以后类似的问题又出现了, 客户会说 "你们到底有没有修复".

最终导致的结果是**客户不信任**我们. 在这种情况下即使向量检索的准确率已经超过了传统方法, 它在获客方面仍然存在许多风险, 除非客户非常专业的用测试集来系统测试, 但即使这样, 客户的客户还会质疑.

我们的建议是回归传统的 Elasticsearch 方法. 即 **回归搜索引擎本质**, 把它当成一个系统的独立任务来做.

在这个文档问答的任务中, 客户提供一些说明手册等非结构化数据给我们:

1. 它就像是建立一个小型的只有几十个, 几百个网页的搜索引擎. **这个我们直接建立 Elasticsearch 索引即可**.
2. 由于 LLM 的上下文有限, 我们必须确保正确答案在有限的上下文里. **这个问题, 不同的原始数据对应了不同的分割策略**.
3. Elasticsearch 检索的问题是 "有哪些适合旅游的城市" 这样的 query 无法匹配到 "北京的风景很好", "上海的风景很好" 这样的文档. 推荐的解决思路是 LLM **生成新的相关文档**, **给文档增加元信息标签**.

文档分割策略

(1)text_chunk, 就是按一定的长度限制将文档划分成片段, 召回时会召回到 k 个片段, 全部提供给 LLM 作参考.

缺点:

- 可能会把不相关的上下文提供给 LLM, 而 LLM 又不知道这段文字的主题, 会默认是与当前 query 相关的信息, 而**回答错误**.
- 有些 query 所涉及的参考上下文比较长, 这种方式很难确保所有相关的片段都被召回, 会导致很多**回答部分正确**的情况.

优点:

- 这种方式的理念是召回所有相关的内容, 让 LLM 做总结.
- 比如两个片段 "姚明的女儿是姚沁蕾", "姚明的老婆是叶莉", LLM就可能推断出 "叶莉的女儿是姚沁蕾". (但实际应用中可能是 LLM 做自动知识挖掘, 构建知识库, 来实现这个功能).

我的观点: 需要根据几个较远距离的上下文做总结的任务比较难, 不是基本需求.

(2)一个文档对应一个主题, 并且其文档总长度小于**最大上下文长度**.

方案:

1. Elasticsearch 检索, 找到最相关的 k 个文档.
2. LLM 基于这 k 个文档的摘要信息选择最相关的一篇文档, 或者 LLM 对其排序, 再根据上下文允许的长度选择前几个文档.
3. LLM 回答问题.

缺点:

- 只实现了最基本的阅读理解功能, 无法跨越多个文档片段.

优点:

- 在 "一个文档对应一个主题" 的假设下, 正确率大幅提高. 解决了 **回答部分正确** 的问题.
- 解决问题的思路更简明 (当 query 无法正确回答时, 我们只需要优化检索准确率或增加相关文档).
- 我们可以让 LLM 根据已有文档生成 Question-Answer, 即推理出新的问答对来丰富知识库.

我的观点:

这种方法是 **文档问答** 的最优解决思路, 即每一个问题 query 都对应一个文档, LLM 只根据该文档回答问题.

- 当 query 无法从 Elasticsearch 检索到对应文档时, 我们可以添加关键词来干预.
- 对于 "首都", "城市", "北京" 等上位词关系的 query, 如果 Elasticsearch 无法正确匹配, 我们还可以让 LLM 自动创建同义词.

可控是非常主要的特性.

(3)一个文档包含多个主题, 比如上市公司财报, 产品说明书, 很多的内容被包含在一个文件中发布的文档, 其长度较大.

方案:

1. 按第二种 **(2)一个文档对应一个主题** 的思路, 正确分割文档, 并给文档符加关键词 (辅助检索).

我的观点: 比如 "产品说明书", 则可能需要专门建立相匹配的 Elasticsearch 索引结构.

智能客服

整体思路是 ReAct 和 MRKL.

- ReAct 强调思考, 行动, 观察的思维过程.
- MRKL 强调的是在何时调用正确的工具的能力.

本质上 LLM 在这里充当的是多轮会话管理的能力.

假设我们的是电商客户.

Tool 包含: `商品推荐`, `订单查询`, `物流查询`, `注册`,`付款`,`保障`, `优惠活动查询`, 等等.

- "商品推荐", "订单查询", "物流查询": 客户提供 API, 我们将其包装成 tool.
- "注册,付款,保障": 是用户提供 FAQ 或者相应的文档, 即 **文档问答**. (文档问答最终是做为一个工具存在, tool).