Lab2 Report

范禹尧 522070910015

2025 年 5 月 8 日

1 系统实现

本系统基于 RAG 技术构建，核心流程包括**上下文分块**、**混合检索**、**生成模型调用**和**答案评估**四个阶段。系统无需显式查询分解 (Query Decomposition)，直接通过原问题检索相关文档，结合 LLM 生成答案，满足多跳问答系统的设计要求。

1.1 基础框架和评估功能

参考`example.py` 文件中的框架，对 `main.py` 文件中的框架进行概述：

- **数据加载：** `load_dataset` 函数可以加载`hotpotqa_longbench.json` 数据集。
- **模型交互：** `query_chat_model`函数演示了如何调用模型接口。
- **评估函数：**提供了用于清洗答案的`normalize_answer`函数与计算分数的`compute_em`和`compute_f1`函数。
- **主流程：** `run_evaluation`函数串联了数据加载、模型调用、评估和结果保存的完整流程。

1.2 RAG 实现核心步骤

- **上下文分块：**使用`semantic_chunking`函数实现，目的是将长文本分割为语义连贯的短块，便于后续检索，参数设置为`chunk_size=300; overlap=100`，即每块约 300 词，滑动窗口叠 100 词，减少关键信息截断风险。
- **混合检索：** `hybrid_retrieval`通过使用 BM25 和词重叠统计策略，结合关键词匹配与语义相似性，提升多跳问题相关块的召回率。
  - **BM25:** 基于词频和逆文档频率的经典算法，擅长精确匹配关键词。
  - **词重叠统计:** 补充召回与问题语义相关但关键词不匹配块。
- **生成模型交互：**将检索到的上下文与问题拼接，引导模型生成精确答案。

  初始提示词：

```
query_to_model = (
    f"Context: {context_str}"
    f"Question: {item['question']}"
    "Instruction: Based *only* on the provided context, answer the question concisely."
```

```
        "For Yes/No questions, answer only 'Yes' or 'No'. "
        "For other questions, provide only the exact answer phrase."
    )
```

强调答案必须基于上下文，且限制答案格式为短短语，避免冗余解释。

最终主函数中 RAG 检索代码如下：

```
        context_chunks = semantic_chunking([item["context"]])
        retrieved_chunks = hybrid_retrieval(item["question"], context_chunks, top_k=5)
        context_str = "\n".join(retrieved_chunks)
```

2 评估结果

评估结果描述：

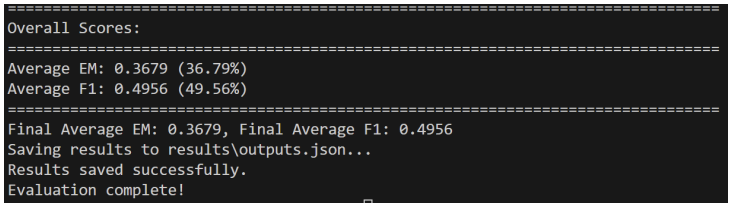

图 1: main.py 运行结果

- **性能指标概览：**
  - EM(Exact Match):36.79%
  - F1 Score:49.56%
- 系统成功满足最低要求 ($EM > 0.3, F1 > 0.4$)，且在两项指标上均超出基准。
- 结果表明，RAG 系统能够有效结合检索与生成能力，部分解决多跳问题的复杂推理需求。

系统表现良好的核心原因分析：

模块	作用
分块策略	保留多跳问题所需的关联段落
混合检索	提高多实体问题的召回率
生成约束	减少模型自由发挥，提升答案准确性
评估标准化	降低格式差异对评分的影响，公平反映语义匹配

表 1: 原因分析

3 用例分析及策略改进

3.1 用例分析

尽管代码实现了基于 RAG 的多跳问答系统并达到了基本的性能要求，但在处理某些特定类型的问题时仍存在不足。以下是具体案例分析：

错误案例 1：Sample 189

问题：what is the group called that Dianne Morgan and Joe Wilkinson a part of in the BBC comedy "Two Episodes of Mash"

预测答案：Two Episodes of Mash **正确答案**：the deadpan sketch group

问题原因：语义理解不足：模型未能正确理解问题的语义，将问题中的 "Two Episodes of Mash" 误认为是答案。

检索结果不准确：检索到的上下文可能没有直接包含 "the deadpan sketch group" 这一关键信息，导致模型无法找到正确答案。

错误案例 2：Sample 185

问题：Which is a beauty magazine, Allure or Claudia?

预测答案：Yes **正确答案**：Allure

问题原因：问题格式误解：模型可能将问题误解为一个一般性问题，而不是一个二选一的具体选择问题。

错误案例 3：Sample 183

问题：Who did the actor that plays Sean Tully defeat in a dancing contest?

预测答案：Antony Cotton **正确答案**：Jodie Prenger

问题原因：多跳推理不足：问题需要先识别 Sean Tully 的扮演者，然后再从其经历中找到舞蹈比赛的对手。模型未能完成这一多跳推理过程。

3.2 策略改进方案

为了解决上述问题，可以尝试以下改进方法：

改进方法 1：引入多轮对话和验证机制

在提示词中添加"Check the answer you provide before you finally submit, make sure the answer you provide is the best answer." "If you do not find the appropriate answer to the problem, try to read the context again and sketch the possible answer." 允许模型在多轮对话中逐步检索和整合信息，增强多跳问题推理能力；随着对话的进行，动态更新上下文，使模型能基于最新信息进行推理。

改进方法 2：提示词优化，明确任务类型

注意到在解决多选一问题时，模型会错误将其判断为 Yes/No 判断问题，因此可以在提示词中添加相应约束，明确该类型任务解决方法。添加提示词如下："If the question is one of the two (e.g. A or B), answer the option name directly, not Yes/No."

改进方法 3：使用推理能力更强的模型

在原始实验中我们使用 Llama3:8B Instruct 对问题进行解答，而错误案例表明，该模型在语义理解和结果检测中仍存在不足，因此考虑使用更复杂的模型进行推理，这里我们使用付费模型 Qwen plus 进行替代。该模型具有更强的理解与生成能力，支持复杂逻辑推理和多轮对话等任务，优化后的架构计算资源消耗低，响应速度快，可以很好的解决原始模型的问题。我们将上述改进后

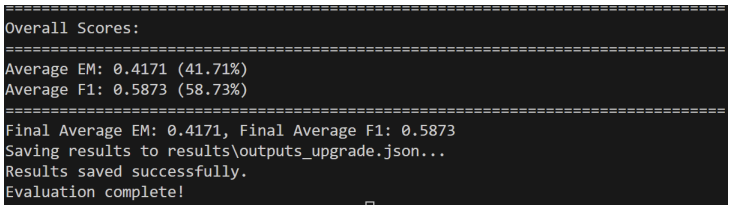

图 2: $main_improved.py$ 评估结果

的代码单独存放在`main_improved.py`文件里，运行该文件，最终得到评估结果。

性能指标概览：

- **EM**:36.79% → 41.71%
- **F1 Score**:49.56% → 58.73%

从评估结果来看，该改进方法对最终实验结果有所提升，即新策略是有效策略。