1. 论文名称：the chronicles of rag: the retriever, the chunk and the generator
   RAG编年史：检索器，文本块和生成器

Abstract

背景：

- RAG是一种使大型语言模型（LLMs）访问外部数据的流行范式。
- RAG的实施面临诸多挑战，包括检索模型的有效整合、表示学习、数据多样性、计算效率优化等。

目的：

- 提出对巴西葡萄牙语实施、优化和评估RAG的良好实践。
- 通过简单的流水线进行推理和实验，回答有关哈利·波特书籍的问题。

方法：

- 使用OpenAI的gpt-4、gpt-4-1106-preview、gpt-3.5-turbo-1106和Google的Gemini Pro生成答案。
- 重点提升检索器的质量。

结果：

- 方法使得MRR@10相比基线提高了35.4%。
- 经过优化，性能进一步提升了2.4%。

结论：

- RAG架构经过建议的增强后，相对得分从57.88%提高到了98.61%的最大值。

1 Introduction

背景：

- LLMs在AI应用中取得显著成绩，尤其在翻译、总结等任务中。
- 需要基于最新信息和外部数据提供答案的问题上存在挑战。

挑战：

- 实现RAG技术面临新挑战，包括发展可靠检索器和确保检索文本相关性。

RAG领域发展：

- RAG研究快速扩张，新论文介绍了不同的实现方式和技术改进。
- 这些发展给AI实践者评估性能和适用性带来挑战。

研究内容：

- 针对巴西葡萄牙语的RAG实施的全面实验。
- 评估不同检索技术，探讨分块策略优化检索信息的整合。

- 分析文档位置对生成内容质量的影响。
- 比较不同LLMs在整合检索信息产生回应的能力。

主要贡献：

1. 数据集准备方法，量化RAG系统不同步骤的质量。
2. 最大相对分数度量，量化方法与理想RAG系统间的差距。
3. 开发RAG系统时的最佳实践和优化讨论。

2 数据预处理

1. 数据集选择：选用的数据集是巴西葡萄牙语版本的第一本《哈利·波特》书籍。这个选择的原因是该书籍广为人知，且Gemini Pro和OpenAI模型都能就此主题回答一般性问题。
2. 数据处理：使用标准的ChatGPT分词器c110k_base，发现总共大约有140,000个词标。这允许创建包含整本书的提示语。
3. 数据集构建：随后，开发了一个包含问题和相应答案的数据集，问题和答案都是由gpt-4模型生成，并基于一个参考文本块。

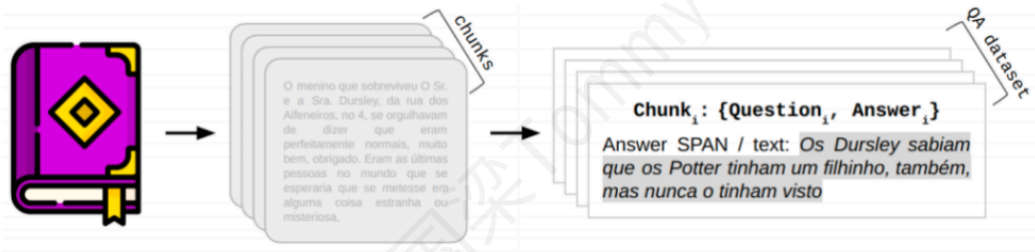

图1：从一个大文档（书籍）中创建了块，并且对于每个块，使用GPT-4生成了问题和答案，答案包含在块中。

3 如何评估

1. 问题：

- 传统评价指标如BLEU和ROUGE可能无法准确反映文本之间的上下文相似性。

2. 方法：

- 使用gpt-4提供基于特定提示的评分。
- 开发了一个五级评分系统来比较文本。

3. 评分标准：

- 1分：答案与参考无关。
- 3分：答案略有相关性，但不一致。
- 5分：答案中度相关，但不准确。
- 7分：答案基本一致，但有遗漏。

- 10分：答案完全准确，完美对齐。

4. 评估提示：

- 评估使用附录A中的提示语。
- 评估采用一次性技术进行每个类别评分。

3.1 相对最大得分

1.目的：

- 评估不同LLM在RAG系统中的性能变化。

2.方法：

- 创建相对最大分数指标，基于模型评估问题和块的正确组合时给出的分数。

3.结果：

- 相对最大分数约为7.4分，表明即便有完美检索器，RAG系统也未能在此数据集上获得完美分数。

4.实验设计：

- 实验结果基于相对最大分数和降级分数进行评估。

5.降级分数定义：

$$\text{降级分数} = 1 - \frac{\text{实验分数}}{\text{相对最大分数}}$$

6.目标：

- 通过降级分数明确检索器系统的主要差距。

Table 1: Relative maximum score in 140 questions from the created QA Harry Potter dataset.

Model	Relative Maximum
gpt-4	7.55
gpt-4-1106-preview	7.32
gpt-3.5-turbo-1106	7.35
Gemini Pro	7.52

4 初步实验

1. 目的：
    - 建立基线以用于评估第3节中定义的指标。
    - 比较使用较低复杂度技术的结果与基线的差异。
2. 注意：
    - 没有探索提示工程技术，尽管已知提示工程直接影响性能。

4.1 基线实验：无上下文

- LLMs在大量数据集上训练，涵盖整个网络，结合哈利·波特宇宙的流行，为测试孤立问题提供了强假设。
- 对于基本问题，如"哈利·波特是谁？"、"谁杀了邓布利多？"和"哈利·波特的主要朋友是谁？"，ChatGPT能够精确回答。
- 对于更详细的问题，性能只是合理的。
- 问题和答案示例：
  - 查询：哈利为了阻止奎里尔专注于镜子并试图发现魔法石的位置采取了什么策略?
    - 答案：哈利想到的是让奎里尔继续谈话以阻止他专注于镜子。
  - 查询：哈利·波特收到了哪型的扫帚，是谁提到了给弗立维教授的特殊情况?
    - 答案：哈利·波特收到的扫帚型号是光轮2000，提到特殊情况的是弗立维教授。
- 结果：
  - 表2展示了使用某些已知LLM对140个问题（如第2节中所建立）得到的基线结果，这些问题没有使用检索到的上下文，仅使用了问题本身进行评估。

Table 2: Performance of the External Knowledge experiment.

Model	Average Score	Degradation
gpt-4	5.35	-29.1%
gpt-4-1106-preview	5.06	-30.9%
gpt-3.5-turbo-1106	4.91	-32.8%
Gemini Pro	3.81	-50.8%

4.2 Long context

1. 对比：

- 与GPT-1和GPT-2模型相比，gpt-4-1106-preview模型能处理高达128k个输入词标，比前者的1024个输入词标大幅提升。

2. 架构和训练：

- gpt-4的具体架构未公开，但据信没有在128k词标输入上下文中进行预训练。
- 可能通过后训练技术扩展输入词标数量。
- 重要的是，使用这样的技术可能在达到扩展极限时导致性能下降。

3. 实验：

- 为评估gpt-4-1106-preview在全上下文容量下的性能，进行了类似于数据集中的"中间丢失"分析。

- 分析了模型输出，同时改变答案在提示中的位置。
- 实验通过改变包含问题答案的块的深度（以总词标数量的10%为增量），在y轴上展示了11种答案深度变化。

4. 结果：

- 提高输入长度时，分数有显著下降。
- 位于（40%，80%）区间的答案表现最差，如"中间丢失"文章中所记录。

1. 图表：
    - 图2展示了实验分数，颜色越绿表示分数越好。
    - 图3显示了答案位置与性能下降之间的关系。

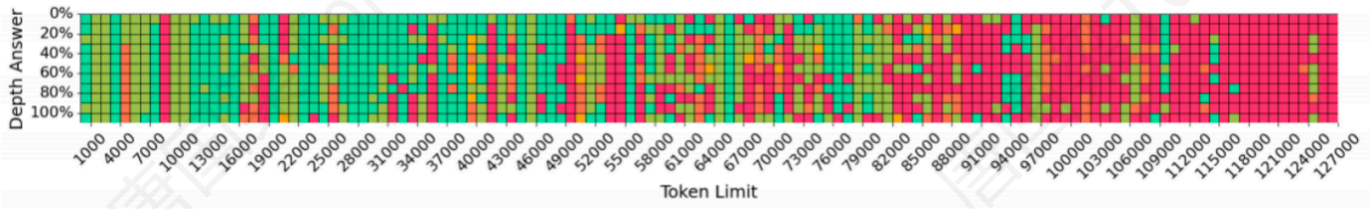

图2：GPT-4-1106-preview在哈利波特数据集上的性能，x轴：在文档中输入的每个1,000个标记之间间隔，y轴：代表答案在文档中的深度。越绿越好。基于Gregory库的图像。

图2展示了实验分数，颜色越绿表示分数越好

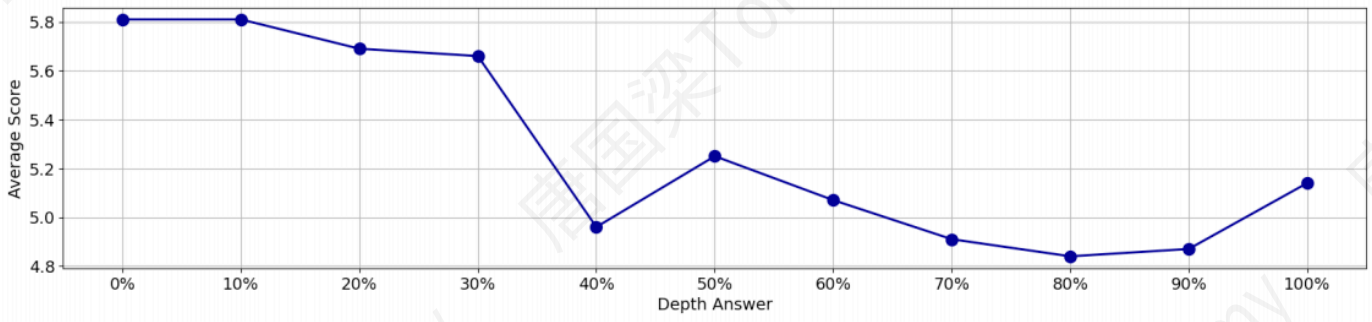

Figure 3: Average performance analysis of `gpt-4-1006-preview` using 128k tokens context per answer depth.

图3显示了答案位置与性能下降之间的关系

4.3 RAG Naive

1. 方法：

- 使用llama-index进行直接的RAG实验，采用所有默认的超参数。
- 使用余弦相似性和ADA-002嵌入来检索块。
- 图4描述了问题处理的基本流程。

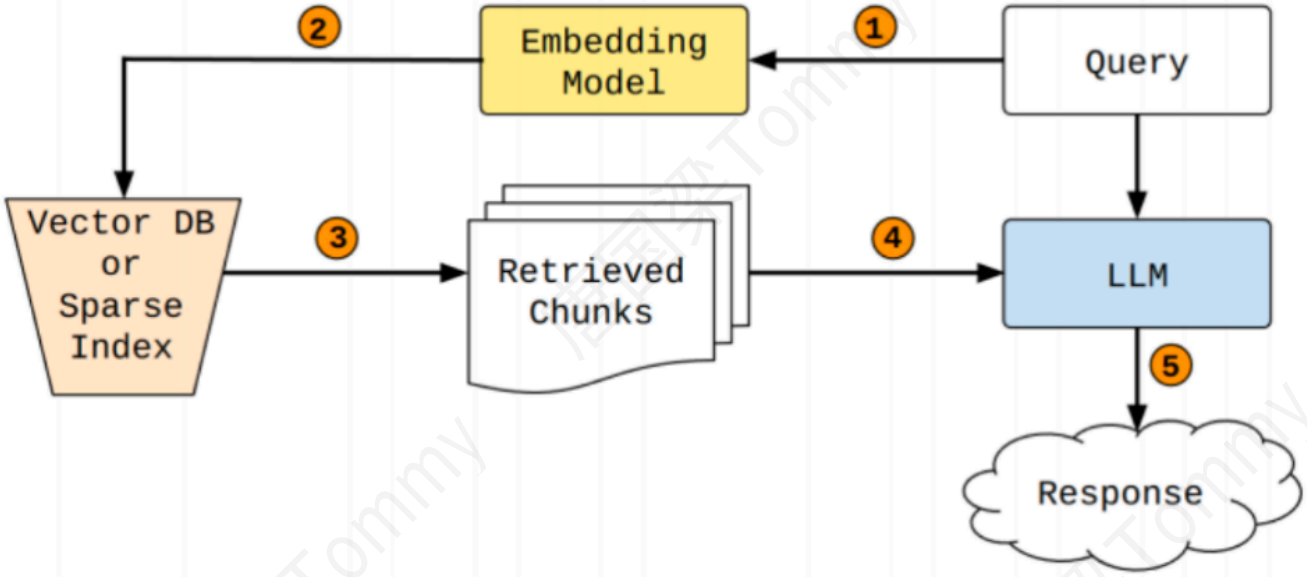

图4：1. 将查询传递给嵌入模型，将其语义表示为嵌入查询向量；2. 将嵌入查询向量传递给向量数据库或稀疏索引（BM25）；3. 根据检索器算法获取前k个最相关的块；4. 将查询文本和检索到的块转发给大型语言模型（LLM）；5. 使用LLM基于检索到的内容填充提示生成响应。

Table 3: Performance of the RAG naive.

Model	Average Score	Degradation
gpt-4	6.04	-20%
gpt-4-1106-preview	5.74	-21.6%
gpt-3.5-turbo-1106	5.80	-21.0%

5 高级实验

1. 背景：
   - 第4节的研究和实验显示了不满意的性能，至少比峰值性能降低了20%。
   - 本节探索了RAG的各种检索方法，认识到检索器的质量是提高这种问题性能的关键因素。
2. 实验设计：
   - 对稀疏和密集搜索进行评估，使用混合方法，甚至采用了多阶段架构使用重排序器。
3. 系统选择：
   - 为了代码调试的灵活性和每个阶段更容易的定制化，没有使用RAG框架（如LangChain或Llama-Index）。

5.1 检索器 Retriever

1. 检索器评估：

- 评估策略围绕检索器在检索相关信息上的表现，使用召回率作为评价指标。
- 引入了倒数排名（RR）到分析中，这个指标考虑了相关块出现的具体排名位置。
- 评估使用召回率和倒数排名在特定截止点进行，作为给定检索方法效果的综合度量。

2. 检索方法：

- 稀疏检索器强调BM25技术，一个基于统计权重的技术。
- 双编码器设计的双向检索器，可以独立编码查询和文档，创建单独的向量表示。
- 混合搜索技术试图结合稀疏和密集搜索技术的优势。

3. 多阶段搜索架构：

- 基于检索和重排管道，第一阶段使用好的检索器来选择初始列表的文档，然后在第二阶段进行高计算复杂度的重排。

5.1.1 BM25

1.BM25的优势：

- BM25因其用户友好性质在RAG评估中常被采用。
- 一个相关的研究表明，BM25能够建立一个强大的基线，并因数据与SQuAD数据集的相似性而提高效果。

2.BM25排名函数：

- k_1 和 b 是影响词项饱和度和文档长度规范化的参数。
- BM25公式整合了这些参数来评分文档对于一个查询的相关性，提供了在不同检索场景下提高有效性的灵活性。

1. 实验结果（表4）：
    - 展示了使用 $k_1 = 0.82$ 和 $b = 0.68$ 时，不同BM25实现的比较。
    - 对比了rank-bm25和Pyserini BM25两种实现，以及Pyserini相较于rank-bm25的性能增益。

Table 4: Comparison between BM25 packages using k1=0.82 and b=0.68.

Recall@k	rank-bm25	Pyserini BM25	Pyserini Gain (%)
3	0.735	0.914	24.3
5	0.814	0.971	19.2
7	0.857	0.985	14.9
9	0.878	0.985	12.1

2. 实施影响：
    - 选择的BM25实现会影响效果。
    - Pyserini的BM25实现包括预处理步骤，如词干提取和特定语言的停用词去除。
    - 为了比较，包括了使用rank-bm25的结果，这是一种没有预处理的基础实现，并且在Python中广泛使用，集成到像LangChain和Llama-index这样的库中。
3. Pyserini BM25的应用：
    - 在所有实验中都使用了Pyserini BM25实现，考虑 $k_1 = 0.82$ 和 $b = 0.68$ 。

5.1.2 ADA-002

1. ADA-002架构：
    - OpenAI没有公开ADA-002架构的详细信息。
    - 尽管如此，该模型在检索过程中被用作所展示的双编码器设计。
2. 双编码器设计：
    - 为所有可用的块构建向量表示。
    - 每个输入查询在搜索时计算其嵌入。
    - 随后，使用余弦相似性评估查询和块之间的相似度。
3. 图5说明：
    - 展示了双编码器架构，其中包括输入1的查询词标和输入2的块词标。
    - 两个输入在多层架构中各自编码，然后通过余弦相似度来评估它们之间的相似性。

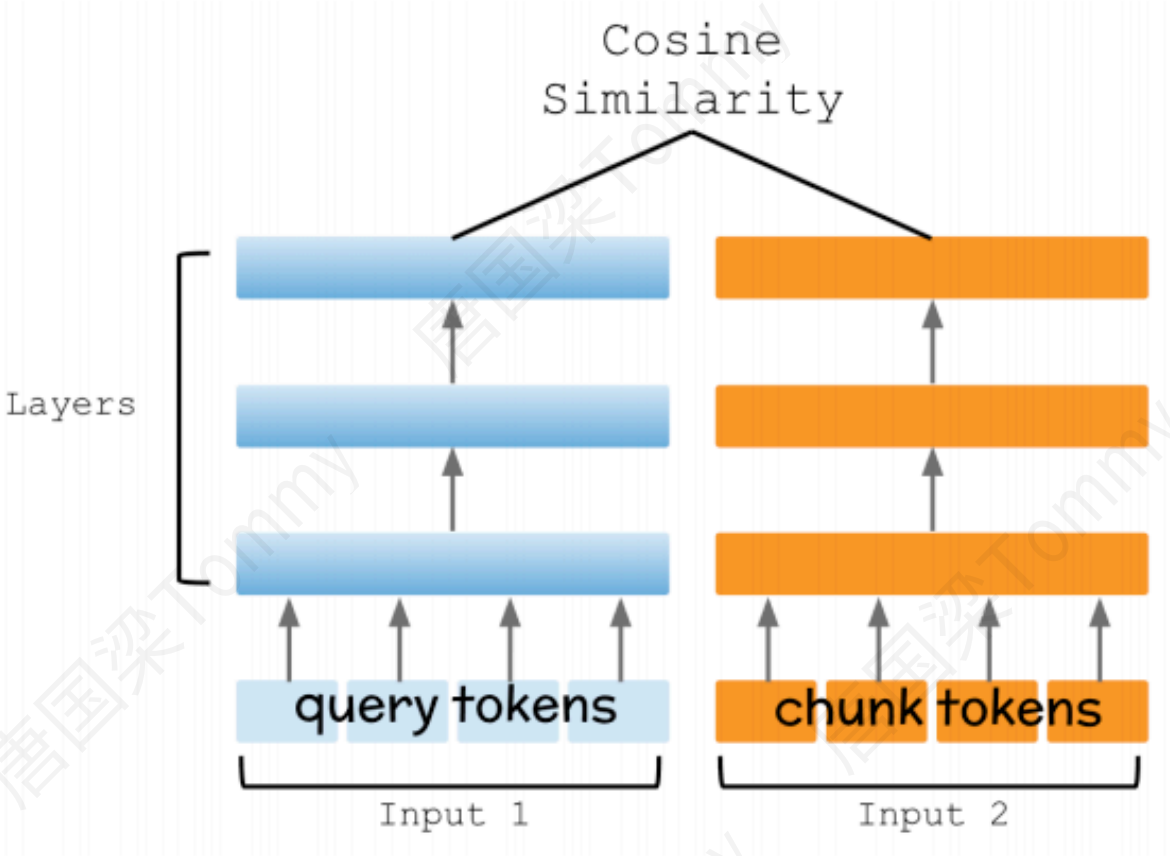

Figure 5: Bi-Encoder Architecture

4. 由于没有关于ADA-002的更多细节，文本中提到将这种方法仅称为密集检索器。

5.1.3 Custom ADA-002

1. 自定义ADA-002：
    - 采用了自定义ADA-002方法，在第5.1.2节中介绍的密集检索器配置中。
    - 嵌入自定义在提高整体表征中扮演了关键角色。
2. 嵌入自定义技术：
    - 不仅限于OpenAI的嵌入，适用于其他同类嵌入。
    - 有多种方法可以优化矩阵，其中之一是在未来工作中探索的多负例排名损失（Multiple Negative Ranking Loss）。
3. 微调阶段：
    - 需要两种类型的样本：正样本（标签为1）和负样本（标签为-1）。
    - 在数据集中，通常只有正样本，但可以通过简单的随机洗牌生成负样本。
    - 最终数据集包含约400个例子，保持1：3的正负样本比例。
4. 影响性能的超参数：
    - 学习率、批量大小和投影矩阵的维度。
    - ADA-002模型有1536个序列长度，投影矩阵的大小是1536 x N，其中N可以是1024、2048、4096。

- 实验中观察到2048维度的投影矩阵取得了最佳精度。

5. 微调要求：
   - 需要较低的GPU资源，训练时间大约5分钟，使用A100 GPU。
   - 模型结构简单，包含一个带有dropout的矩阵和双曲正切激活函数，这在训练集中提供了额外的准确性增益。
6. 分析正负类别的余弦相似性：
   - 分析了正类和负类之间的余弦相似性，观察到他们共享的"阴影"。
   - 理想情况下，希望类别之间是分离的，以确保空间的清晰定义。

5.1.4 混合搜索

1. 混合搜索：
   - 当需要结合两种或两种以上检索方法的结果时应用混合搜索。
   - 用于解决这类问题的一个广泛使用的算法是倒数排名融合（Reciprocal Rank Fusion, RRF）。
2. RRF算法：
   - 对于文档集 D 和不同方法 r 的搜索结果，可以计算 $RRF_{score}(d \in D)$。
   - RRF_{score} 的计算包括了倒数排名，其中 $r(d)$ 表示文档 d 被检索方法 r 检索到的位置，k 用于帮助控制离群系统。
3. 图8和公式（2）说明：

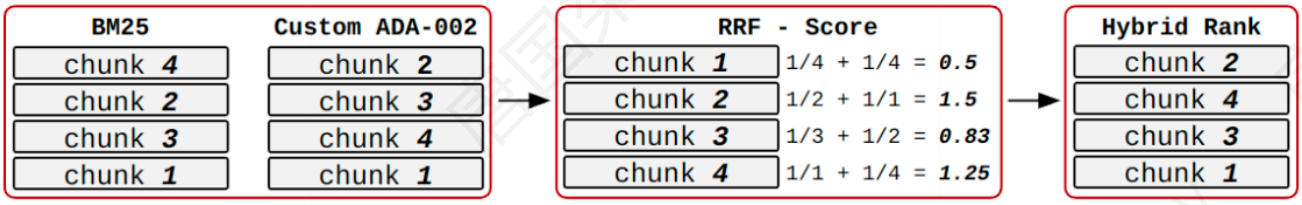

Figure 8: Hybrid Search schema with k=1.

$$RRF_{score}(d \in D) = \sum_{r \in R} \frac{1}{k + r(d)}, \tag{2}$$

   - 展示了如何计算 RRF_{score}，在这个例子中 $k = 1$。
   - 有四个块按照两种搜索方法（BM25和自定义ADA-002）检索到的不同顺序。
   - 计算每个块的倒数排名分数，然后将这些值求和，创建一个新的分数。
   - 最终的混合列表是使用这个新分数的块排序。
4. 实验结果（表5）：

Table 5: Retriever comparison. Where MRR is the Mean Recriprocal Rank metric and R@k is the Recall.

Metric	Hybrid-BM25-ADA-002	Hybrid-BM25-Custom ADA-002
MRR@10	0.758	0.850
R@3	0.829	0.921
R@5	0.879	0.943
R@7	0.921	0.964
R@9	0.957	0.979

- 展示了混合BM25-ADA-002方法的检索器比较。
- 其中包括平均倒数排名（MRR）和召回率（R@k）。
- 实验中，只有Pyserini的BM25被测试为稀疏检索器，而ADA-002和自定义ADA-002被测试为密集检索器。
- 提供了最佳结果的混合组合是使用BM25和自定义ADA-002。

5.1.5 Reranker 重排

1. 重排器的基本概念：
   - 多阶段排名的基础是将文档排名分为一系列阶段，每个后续阶段重新评估并排名前一阶段传递来的候选集。
2. 多阶段排名：
   - 图9展示了一个多阶段管道，其中Pyserini BM25负责第一阶段，然后候选块由重排器重新评估。

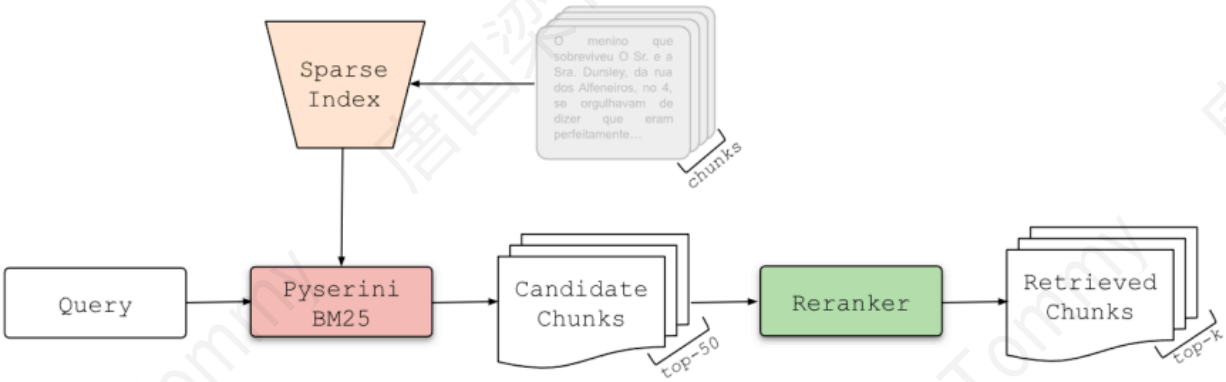

Figure 9: Reranker Pipeline

   - 重排后的列表称为检索到的块，由 k 个块组成最终结果。
3. 重排器的使用：
   - 通常使用基于Transformer的模型作为重排器，利用它们捕捉文档和查询中复杂关系和上下文信息的能力来提高信息检索系统的效果。
   - 最初在多阶段排名框架内使用Transformer的研究提出了monoBERT，将排名过程转换为相关性分类问题。
4. monoBERT与monoT5：
   - monoBERT通过计算 $P(\text{Relevant} = 1|d_i; q)$ 来对文本进行排序，其中 q 是查询，d_i 表示文档。

- 另一方面，monoT5是一种序列到序列的重排器，使用T5模型生成查询和文档之间的分数，这要求对文本到文本任务进行一些调整。

5. 实验中的应用：
   - 在实验中，首先使用Pyserini BM25作为第一阶段检索，检索50个文档进行下一阶段的重排。
   - 第二阶段利用了unicamp-dl/mt5-base-en-pt-msmarco-v2模型，这是一个基于序列到序列的重排器，它在英语和葡萄牙语的查询和文档对上进行了训练。
6. 重排器的推理过程：
   - 图11展示了monoT5的推理过程，其中输入格式是 `'Query: {query} Document: {document} Relevant:'`，在训练时使用标签yes如果文档与查询相关，否则为no。
   - 在推理时，通过计算是（yes）和否（no）标记的softmax值来获取分数。

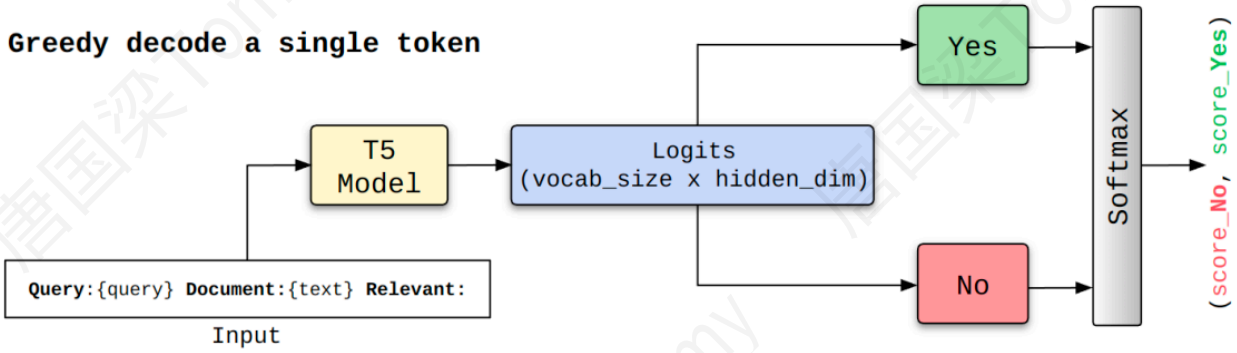

Figure 11: monoT5's inference process

5.2 检索结果

Table 6: Retriever comparison.

Metric	ADA-002	Custom ADA-002	Hybrid-BM25-ADA-002	Hybrid-BM25-Custom ADA-002	BM25	BM25 + Reranker
MRR@10	0.565	0.665	0.758	0.850	0.879	0.919
R@3	0.628	0.735	0.829	0.921	0.914	0.971
R@5	0.692	0.835	0.879	0.943	0.971	0.985
R@7	0.750	0.871	0.921	0.964	0.985	0.992
R@9	0.814	0.921	0.957	0.979	0.985	1

1. 结果概述：
   - 展示了各种检索器的结果，如表6所示。
2. 检索器比较（表6）：
   - 对比了ADA-002、自定义ADA-002、混合BM25-ADA-002、BM25以及BM25加重排器的性能。
   - 使用了两个指标：平均倒数排名（MRR）@10和召回率（R）@k。
3. 性能表现：
   - BM25加重排器在MRR@10和R@k上取得了最佳结果。
   - 其他检索器配置在不同召回率阈值下的表现也有所提升。

这表明，通过结合BM25检索器和后续重排过程，可以显著提高检索任务的性能。

6 结论

1. RAG系统的挑战：
    - RAG系统的实施面临着有效整合检索模型、高效的表示学习、数据多样性、计算效率优化、评估和文本生成质量等挑战。
2. 最佳实践和简化流程：
    - 文章提出了在巴西葡萄牙语数据集上实施、优化和评估RAG的最佳实践，重点是简化推理和实验的流程。
3. 性能改进：
    - 讨论了检索器质量和性能改进之间的关系，提到采用的方法使MRR@10相比基线提高了35.4%。
4. 输入大小对性能的影响：
    - 通过输入大小的优化，可以进一步提升信息检索策略的性能，观察到性能提高了2.4%。
5. RAG架构的完整评估：
    - 最后，展示了完整的RAG架构，并提出了优化建议。
    - 最终，该方法的准确率达到了98.61%，相比基线显示了40.73%的降级分数改进。

6.1 检索分数与性能

1. 性能变化：
    - 检索器的性能变化范围在MRR@10指标上为0.565到0.919，如表6所详细描述。

Table 6: Retriever comparison.

Metric	ADA-002	Custom ADA-002	Hybrid-BM25-ADA-002	Hybrid-BM25-Custom ADA-002	BM25	BM25 + Reranker
MRR@10	0.565	0.665	0.758	0.850	0.879	0.919
R@3	0.628	0.735	0.829	0.921	0.914	0.971
R@5	0.692	0.835	0.879	0.943	0.971	0.985
R@7	0.750	0.871	0.921	0.964	0.985	0.992
R@9	0.814	0.921	0.957	0.979	0.985	1

2. 性能与检索器质量的关系：
    - RAG的性能直接受到检索器质量的影响，这一点非常重要。
3. 图12解释：
    - 展示了不同检索方法的有效性与RAG性能之间的关系。
    - 横轴是MRR@10指标，纵轴是降级分数（0代表理想情况）。

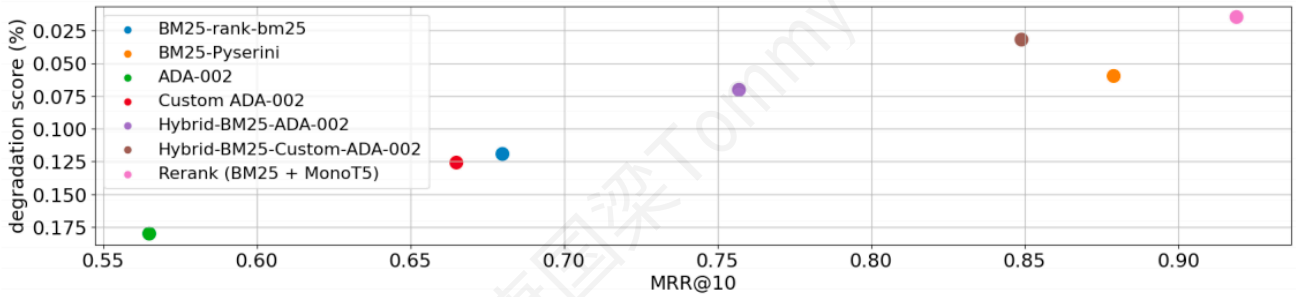

Figure 12: Retriever effectiveness vs RAG Performance. x axis is the MRR@10 metric and y axis is the degradation (where 0 is the perfect scenario) score.

图12说明了不同检索方法在保持较低降级分数的同时，如何在MRR@10上实现不同的性能水平，其中包括BM25、Pyserini BM25、ADA-002、自定义ADA-002、混合方法和重排方法。这些数据可用于评估和选择适合特定RAG应用的最佳检索器。

6.2 输入大小与性能

1. 性能观察：
   - 使用检索-重排策略检索3个块时，获得了最佳性能，如表7所示。

Table 7: Performance of number of chunks retrieved with gpt-4.

# Retrieved Chunks	ADA-002	Custom ADA-002	BM25	Hybrid	BM25 + Reranker
3	6.19	6.41	7.10	7.31	**7.44**
5	6.29	6.61	7.32	7.37	7.43
7	6.42	6.82	7.17	7.20	7.32
9	6.57	6.88	7.22	7.34	7.37

   - 重排器的使用（如图9所示）在测试中改善了信息检索的结果。

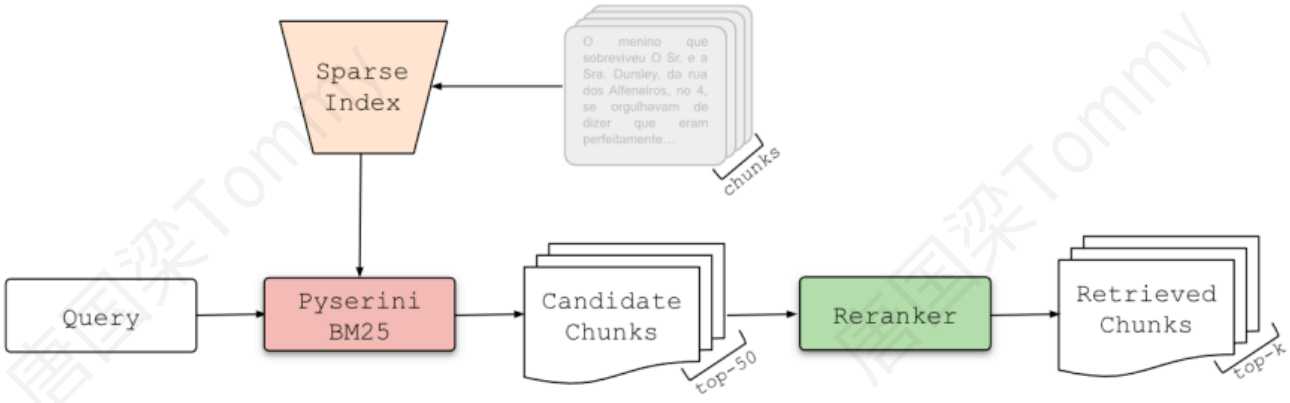

Figure 9: Reranker Pipeline

   - 在相同配置下，Gemini Pro的性能与gpt-4相似，如表8所示。

Table 8: Performance of best retriever RAG.

Model	Retriever Method	# Retrieved Chunks	Degradation
gpt-4	ADA-002	9	-13.0%
gpt-4	ADA-002 Custom	9	-8.8%
gpt-4	BM25	5	-3%
gpt-4	Hybrid	5	-2.3%
gpt-4	**BM25 + Reranker**	**3**	**-1.4%**
Gemini Pro	BM25 + Reranker	3	-2.2%

2. 输入大小的影响：
    - 尽管使用9000个词标的输入（表6所示）实现了9块的完美召回率，但这并未导致最佳性能。

Table 6: Retriever comparison.

Metric	ADA-002	Custom ADA-002	Hybrid-BM25-ADA-002	Hybrid-BM25-Custom ADA-002	BM25	BM25 + Reranker
MRR@10	0.565	0.665	0.758	0.850	0.879	0.919
R@3	0.628	0.735	0.829	0.921	0.914	0.971
R@5	0.692	0.835	0.879	0.943	0.971	0.985
R@7	0.750	0.871	0.921	0.964	0.985	0.992
R@9	0.814	0.921	0.957	0.979	0.985	1

    - 如第4.2节所讨论的，RAG的最终结果与输入大小及答案所在的位置有直接关系。
3. 成本考虑：
    - 从成本角度考虑，避免过载LLM是至关重要的，因为成本也基于输入文本的数量。
4. 研究结果的通用性：
    - 本研究中获得的结果不能视为对其他数据集的泛化。
    - 探索性数据分析（EDA）的使用和良好检索器实践总是实现良好结果的可靠路径。

表7和表8提供了检索器性能的具体数字，表明检索块数量和重排器使用对于提高RAG系统性能的重要性。

Table 7: Performance of number of chunks retrieved with `gpt-4`.

# Retrieved Chunks	ADA-002	Custom ADA-002	BM25	Hybrid	BM25 + Reranker
3	6.19	6.41	7.10	7.31	**7.44**
5	6.29	6.61	7.32	7.37	7.43
7	6.42	6.82	7.17	7.20	7.32
9	6.57	6.88	7.22	7.34	7.37

Table 8: Performance of best retriever RAG.

Model	Retriever Method	# Retrieved Chunks	Degradation
gpt-4	ADA-002	9	-13.0%
gpt-4	ADA-002 Custom	9	-8.8%
gpt-4	BM25	5	-3%
gpt-4	Hybrid	5	-2.3%
gpt-4	**BM25 + Reranker**	**3**	**-1.4%**
Gemini Pro	BM25 + Reranker	3	-2.2%