EasyRAG: 面向 AIOps 的高效 RAG 框架

冯张驰 计算机学院 北京航空航天大学
匡栋栋 计算机学院 北京航空航天大学
王中元 计算机学院 北京航空航天大学
聂志捷 计算机学院 北京航空航天大学
张日崇 计算机学院 北京航空航天大学

摘要

本文提出了 EasyRAG，一个面向 AIOps 网络运维的简洁、轻量、高效的检索增强问答框架。我们主要的贡献为：(1) **问答准确**，我们设计了一套简洁的基于两路稀疏检索粗排—LLM Reranker 重排—LLM 答案生成与优化的 RAG 方案以及配套的数据处理流程，在初赛中取得了 GLM4 赛道第 1 的成绩，在复赛中取得了 GLM4 赛道第 2 的成绩。(2) **部署简单**，我们的方法主要由 bm25 检索和 bge-reranker 重排组成，无需微调任何模型，占用显存少，部署容易，可扩展性强；我们提供了一套灵活的代码库，内置了多种搜索和生成策略，方便自定义流程实现。(3) **推理高效**，我们设计了一套粗排-重排-生成全流程的高效推理加速方案，能够大幅降低 RAG 的推理延迟且较好地维持了准确度水平；每个加速方案都可以即插即用到任意的 RAG 流程的相关组件中，一致地提升 RAG 系统的效率。

1 EasyRAG 框架介绍

我们的复赛方案可以用图1概括。包含数据处理流程 (1.1) 和 RAG 流程 (1.2)。

1.1 数据处理流程

1.1.1 zedx 文件处理

由于我们发现原来的处理脚本会漏掉一些文件，我们使用如下的步骤对 zedx 进行了重新处理：

1. **zedx 解压**：解压官方源数据 4 个.zedx 文件，得到 4 份 html 文档包
2. **路径解析**：从文档包的 nodetree.xml 中读取每个 html 文档的**知识路径**以及实际存储的**文件路径**
3. **文档抽取**：使用 BeautifulSoup 抽取每个 html 的文本、图像标题和图像路径
4. **保存**：保存文档的文本为 txt 格式，与 html 文档的相对位置保持一致。同时保存知识路径、文件路径与图像路径信息。

1.1.2 文本分块

分块设置 我们使用了 SentenceSplitter 对文档进行分块，即先利用中文分隔符分割为句子，再按照设置的文本块大小合并，使用的块大小 (chunk-size) 为 1024，块重叠大小 (chunk-overlap) 为 200。

消除分块中的路径影响 在实践中，我们发现 llama-idnex 的原实现中含有对路径信息的简单但不稳定的使用方式，即将文本长度减去文件路径长度得到实际使用的文本长度，这样会让不同的数据路径产生不同的分块结果，在初赛中我们发现同样的 chunk-size 和 chunk-overlap，更换路径最多可以为最终评测结果带来 3 个点的波动，这显然在实践中无法接受。针对此问题，我们实现了自定义的分块类，将路径长度的利用给消除，从而保证可稳定复现。

1.1.3 图像信息抽取

基于多模态大模型的图像内容提取 我们先使用 GLM-4V-9B (GLM et al., 2024) 对所有图像的信息进行了提取，我们实践中发现使用以下的简单提示词就能够达到比较好的效果：

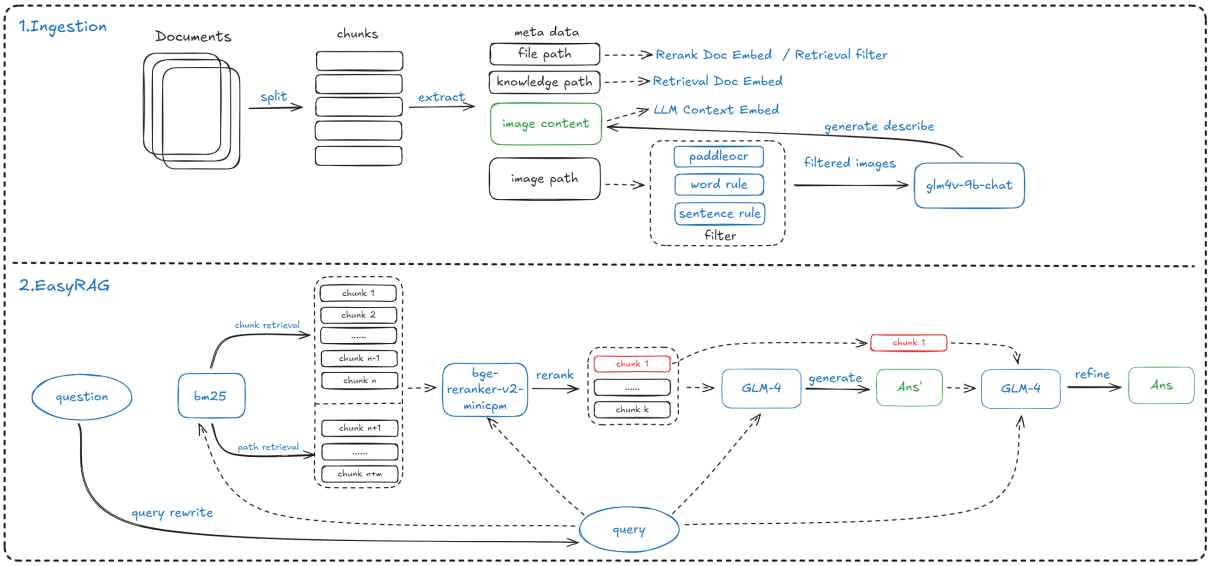

图 1: EasyRAG 框架

简要描述图像

基于多种规则的图像过滤 我们发现少量的图像对于最后的问题回答是有益的，然而并不是所有图像都有利，因此我们设计了灵活的策略对无用的图像使用如下步骤进行过滤：

1. 使用 PP-OCRv4 模型[1]提取图像中的文字内容，过滤不含有中文的图像
2. 过滤标题中含有特别关键词 (组网图，架构等) 的图像
3. 过滤在文中以特定方式被引用的图像 (配置如图 x 所示，文件如图 x 所示等)

用这个过滤步骤，我们将最终的图像数量从原始的 6000 张减少到 200 张以内，值得注意的是，过滤地步骤可以很轻松地进行配置，从而面向实际场景调优。

1.2 RAG 流程

注意，查询改写比赛中实际并未使用，密集检索仅初赛使用，但这两部分技术对于更多场景存在一定意义，因此我们都进行了详细介绍。

[1]https://github.com/PaddlePaddle/PaddleOCR

1.2.1 查询改写

在比赛过程中由于所给的查询都非常简短，并且我们发现部分查询语句存在语义不通顺或者关键词不清晰的问题。例如:“EMSPLuS 告警种类有种?”，“故障来源有哪些?”。我们在将问题 query 输入到 RAG Pipeline 前，通过 LLM（GLM4）进行查询改写，分为查询扩展和假设文档嵌入（HyDE）(Gao et al., 2022) 两种方法。

查询扩展 我们在初赛时对于当前运维场景的查询问题进行了总结，其中查询语句存在以下特点：

- 查询中的技术关键词非常重要
- 查询语句长度短、信息量方差大

在这种情况下，我们尝试了根据初始查询语句和设计的提示词，利用 LLM 模型总结问题中的关键词或者是可能涉及到的一些关键词，即利用 LLM 模型的知识，进行运维、通信领域的关键词联想和查询关键词的总结，我们称之为**关键词扩展**。

人工标注若干条数据中的关键词和可能联想的关键词后，随即利用 LLM 即（GLM4）进行 few-shots 的关键词总结、扩展。我们参考

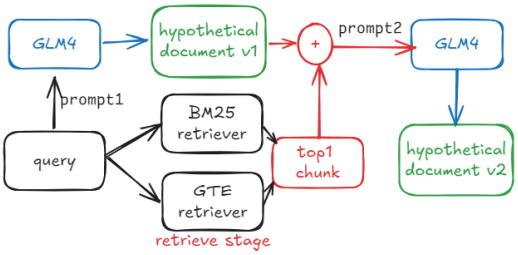

图 2: 虚拟文档生成流程

(Wang et al., 2023), 将扩展得到的关键词基于原始查询通过直接拼接、再次利用大语言模型总结两种方式生成新的查询。

$\mathcal{L}$ 表示的大语言模型 LLM, q, p 分别代表的是初始查询和提示词，p_{exp} 表示扩展查询的提示词，其中包括人工标注的几条数据，p_{sum} 表示的是利用大模型进行语句和扩展关键词的总结拼接的提示词。

HyDE 在面对查询 (query) 缺乏具体性或缺乏易于识别的元素时，例如，不论是密集检索还是稀疏检索方法都很难检索到对应的目标文档, 对此我们参考 (Gao et al., 2022) 中的方法，设计了一套虚拟文档（Hypothetical Document Embeddings）的产生方法。

对于虚构文档的产生方法，我们设计了两种方式，如图2所示，首先仿照论文的做法直接将提示词 p_{hy} 和原始问题 q 输入大语言模型 $\mathcal{L}$，产生得到的虚构文档 q'_0。但是我们在复赛过程中发现这样产生的虚构文档中会有大量因为大模型幻觉产生的无关关键词和冗余的信息极大地影响了检索过程中的效果。于是我们试图通过 BM25 算法和密集检索（利用 GTE-QWEN 编码），找到最相关的 top1 文档，并利用 top1 文档进行上下文提示以减少因大语言模型幻觉导致的初始的虚构文档中 q'_0 的存在无端联想和冗余信息。

对于产生的虚拟文档我们也采用了两种应用方法：1. 利用虚拟文档 q' 和原文档 q 拼接形成新的文档进行**粗排检索**。2. 仅利用虚拟文档 q' 和原文档 q 拼接形成新的文档进行检索结果的**重排序**。

1.2.2 两路稀疏检索粗排

在稀疏检索部分，我们采用 BM25 算法构建检索器, BM25 的核心思想是基于词频 (TF) 和逆文档频率 (IDF) 来, 同时还引入了文档的长度信息来计算文档和查询 q 之间的相关性。具体实现上, BM25 检索器主要由中文分词器、停用词表构成。我们逐一进行介绍

中文分词器 对于中文分词器，我们采用的是常见的 jieba 中文分词器[2]，jieba 分词器的优点在于轻量级，可以开启多线程模式加速分词和词性分析，并且可以根据自己的需要自定义词频或是自定义词典调整分词偏好。对于分词器我们也尝试做了词表的自定义，当前场景是 5G 通讯的运维场景，我们选用了相近领域的清华大学收集的 IT 领域词库[3]，载入分词器的词表中，不过实践中效果一般，因此最终我们仍然使用了原始的 jieba 词表。

停用词表 对于中文的停用词表，我们采用了哈尔滨工业大学搜集的中文常见停用词表[4]作为中文分词过程中过滤无意义词汇的参考词表，通过过滤无意义的词汇和特殊符号以提高有效关键词的命中率, 提高正确文档的召回率。

两路检索 BM25 两路检索粗排为文本块检索和路径检索。

1. 文本块检索。使用 BM25 对分割好的文本块进行搜索，粗排召回得分大于 0 的前 192 个文本块。

2. 路径检索。考虑到部分的问题和我们提取的知识路径相关性很高，比如问题 “VNF 弹性分几类？”，VNF 和弹性都可以直接在相关的知识路径中找到，因此我们设计了路径搜索，使用 BM25 对知识路径进行搜索，粗排召回得分大于 0 的前 6 个文本块。

[2] https://github.com/fxsjy/jieba
[3] http://thuocl.thunlp.org
[4] https://github.com/goto456/stopwords

检索流程 BM25 检索器对于一个给定的查询 q，具体的文档检索流程如下：

1. 文档扩展。对于文本块检索，我们将文件路径和每个文本块拼接起来作为扩展文档用于检索。
2. 文档预处理。先对所有文档 (文本块或路径) 进行停用词过滤，再利用中文分词器进行中文分词，并预先计算文档的的 IDF 分数。
3. 查询处理。对查询 q 进行停用词过滤和中文分词。
4. 相似度召回。统计查询 q 的关键词和计算各个文档的 TF 值，根据 TF、IDF 值计算查询 q 和各个文档中的相关分数，根据分数召回相关文档。
5. 文件路径过滤。对于文本块检索，我们利用赛题中的文件路径来源，对元数据进行比对，过滤掉其他来源的文本块。

1.2.3 密集检索粗排

密集检索部分我们采用了阿里巴巴开发的 gte-Qwen2-7B-instruct[5] (Li et al., 2023)，此模型在 MTEB 上达到了先进效果。

检索流程 密集检索器对于一个给定的查询 q，具体的文档检索流程如下：

1. 文档扩展。我们将文件路径和每个文本块拼接起来作为扩展文档用于检索。
2. 文档编码。将所有文本块输入模型进行编码得到表征，存入 qdrant[6]向量数据库
3. 查询编码。利用查询提示模板将 q 转换为 GTE 的查询输入，利用模型进行编码。
4. 相似度召回。在检索时，使用余弦相似度进行匹配，召回前 288 个文本块。
5. 文件路径过滤。利用赛题中的文件路径来源，使用 qdrant-filter 过滤掉其他来源的文本块。

1.2.4 LLM Reranker 重排

我们采用了 bge-reranker-v2-minicpm-layerwise (Chen et al., 2024)，一个基于 MiniCPM-2B-dpo-bf16 在混合的多个多语言排序数据集上训练的 LLM Reranker，此模型在中英文上具有先进的排序效果，且含有配套的工具代码，可以很方便地根据具体场景进行进一步微调。

重排流程 LLM-Reranker 对于一个给定地查询 q 和 k' 个粗排得到的文本块，具体地文档排序流程如下：

1. 文档扩展。将知识路径和每个文本块拼接起来作为扩展文档用于检索。
2. 文本处理。将 q 和 k' 文本块分别组合成 k' 个查询-文档对，输入分词器得到 LLM 的输入数据。
3. 相似度排序。将输入数据输入 LLM 得到查询和每个文本块的重排分数，并根据此分数进行排序，取最高的 k(一般为 6) 个文本块返回。

1.2.5 多路排序融合

融合算法 由于我们设计了多路检索粗排，那么也需要设计相应的排序融合策略，我们主要使用了简单合并与倒数排序融合 (Reciprocal Rank Fusion RRF) 两种策略。简单合并策略直接将多路得到的文本块进行去重与合并。倒数排序融合则将同一个文档在多个路径的搜索排序的倒数进行求和作为融合的分数进行再次排序。

粗排融合 最直接的排序融合的使用方式是，得到多路粗排结果后直接使用融合算法将多路粗排得到的文本块合并为一个文本块集合，再交给 Reranker 重排。复赛中我们使用了简单合并将两路稀疏检索粗排结果合并。

[5]https://huggingface.co/Alibaba-NLP/gte-Qwen2-7B-instruct

[6]https://qdrant.tech/

重排融合 我们还可以在每一路完成粗排-重排后，再进行融合。初赛中我们融合了文本块稀疏检索和文本块密集检索两路，针对这两路检索，我们设计了三种重排融合方法。(1) 使用 RRF 将多路的粗排-精排后的结果合并。(2) 将多路重排后的文本块输入 LLM 分别得到相应的答案，取答案更长的作为最终答案。(3) 将多路重排后的文本块输入 LLM 分别得到相应的答案，将多路的答案直接拼接。

1.2.6 LLM 回答

此部分我们先将重排得到的 top6 文本块的内容用以下模板拼接得到上下文字符串：

```
### 文档 0: {chunk_i}
...
### 文档 5: {chunk_i}
```

注意，此处输入 GLM4 的文本块是将图像的内容拼接后的，而前文的粗排和重排过程中的文本块都没有拼接图像内容。

之后我们将上下文字符串和问题用如下的问答模板组合成提示词，输入 GLM4 获得答案。

```
上下文信息如下：
---------
{context_str}
---------

请你基于上下文信息而不是自己的知识，回答以下问题，可以分点作答，如果上下文信息没有相关知识，可以回答不确定，不要复述上下文信息：

{query_str}

回答：
```

除此之外，我们还设计了其他格式的问答模板，参考思维链 (Wei et al., 2022) 设计了思维链问答模板 (附录A.2)，参考 COSTAR (Teo, 2023) 设计了 markdown 格式问答模板 (附录A.1)，为了让 LLM 更重视 top1 文档设计了侧重问答模板 (附录A.3)，相关实验结果参见。

1.2.7 LLM 答案优化

由于我们发现 LLM 对于每个文本块都会给予一定的注意力，可能会导致 top1 文本块的有效信息没有得到充分利用，对于这种情况，我们设计了答案整合提示词 (附录B)，将根据 6 个文本块得到的答案利用 top1 文本块进行补充整合，得到最终的答案。

2 准确度

2.1 缩写介绍

为了书写方便，先介绍一些重要组件的编号。

数据 ⓪代表官方处理好的 txt 数据。①代表我们自己处理的 v0 版本 txt 数据，相较官方数据补充了部分缺失的数据。②相较①在每个 txt 开头拼接了对应的知识路径。③代表我们自己处理的 v1 版本 txt 数据，相比 v0 版数据，保留了更多和官方数据一致的 markdown 结构化数据。④相较③在每个 txt 开头拼接了对应的知识路径。

粗排 ⓪ 代表 bge-small-zh-v1.5，①代表 bge-base-zh-v1.5，②代表 bce-embedding-base_v1，③代表 bm25 文本块检索，④代表 gte-Qwen2-7B-instruct。⑤代表 bm25 知识路径检索。

重排 ⓪代表 bge-reranker-v2-m3，①代表 bce-reranker-base_v1，②代表 bge-reranker-v2-minicpm-layerwise 的 40 层，③代表 bge-reranker-v2-minicpm-layerwise 的 28 层。

融合 ⓪代表粗排简单合并，①代表粗排 RRF 融合，②代表重排融合中的生成前融合，③代表生成后选答案更长融合方法，④代表生成后答案拼接融合方法。

序号	数据	分块	粗排	重排	融合	准确度
0	⓪	1024,50	⓪,3	无	无	57.86
1	⓪	1024,50	①,8	无	无	**68.59**
2	⓪	1024,50	③,8	无	无	**69.55**
3	⓪	1024,50	①,192	⓪,8	无	**73.73**
4	⓪	1024,50	①,256	⓪,8	无	70.68
5	⓪	1024,50	②,192	①,8	无	69.25
6	⓪	1024,50	①,288	②,8	无	77.07
7	①	1024,50	①,288	②,8	无	77.51
8	②	1024,50	①,288	②,8	无	77.92
9	②	1024,50	①,256	②,8	无	78.49
10	①	1024,50	③,192	②,6	无	80.90
11	②	1024,50	③,192	②,6	无	81.38
12	②	1024,100	③,192	②,6	无	81.77
13	②	1024,100	③,192	③,6	无	81.88
14	②	1024,200	③,192	②,6	无	82.87
15	②	1024,200	③,192	③,6	无	**82.97**
16	②	1024,200	④,288	③,6	无	**83.02**
17	②	1024,200	④,288 ③,192	③,6	⓪	81.80
18	②	1024,200	④,288 ③,192	③,6	①	82.50
19	②	1024,200	④,288 ③,192	③,6	②	**83.45**
20	②	1024,200	④,288 ③,192	③,6	③	**83.70**
21	②	1024,200	④,288 ③,192	③,6	④	**84.38**

表 1: 初赛实验结果。分块列中两个数字分别代表 chunk_size 和 chunk_overlap。粗排/重排两列用空格分隔多个搜索路径，每个搜索路径逗号隔开的内容分别代表检索/排序方法，topk。

文档扩展 在粗排和重排两列中，①代表文档中拼接了文件路径，②代表文档拼接了知识路径。

2.2 初赛实验

在初赛中，我们将主要的结果在表1中展示，经历了以下 4 个阶段的改进：

1. 单路粗排检索 (0-2)。我们探究了 bge-zh-v1.5 系列 (small,base,large) 模型和 bm25 的单路检索效果。发现 bge-base-zh-v1.5 和 bm25 效果最好，且效果取 top8 时效果最好，过多会导致 LLM 理解不准确，过少会导致缺乏必要信息。

2. 基于 BERT-Reranker 的重排 (3-5)。我们探究了基于 BERT 的 bge 系列重排器和 bce-reranker-base-v1 的效果，发现 bge-reranker-v2-m3 效果最好，粗排大概在 192-288 之间效果取得最佳，重排在 top8 效果最佳。

3. 基于 LLM-Reranker 的重排 (6-16)。这一部分我们有以下探索:(1) 探究了基于 LLM 的 bge 系列重排器效果，发现 bge-reranker-v2-minicpm-layerwise 性能远好

序号	数据	分块	粗排	重排	融合	图像信息	答案整合	准确度
0	②	960,200	③,192	③,6	无	无	无	**91.53**
1	②	960,200	④,288	③,6	无	无	无	88.40
2	②	960,200	④,288 ③,192	③,6	②	无	无	90.00
3	③	1024,200	③,192	③,6	无	无	无	90.26
4	④	1024,200	③,192	③,6	无	无	无	**91.38**
5	③	1024,200	③,192,①	③,6	无	无	无	**92.70**
6	③	1024,200	③,192,①	③,6,①	无	无	无	89.30
7	③	1024,200	③,192,①	③,6,②	无	无	无	87.12
8	③	1024,200	③,192,②	③,6	无	无	无	92.43
9	③	1024,200	③,192,②	③,6,①	无	无	无	**93.11**
10	③	1024,200	③,192,②	③,6,②	无	无	无	90.17
11	③	1024,200	③,192,②	③,6,①	无	OCR 粗筛	无	92.5
12	③	1024,200	③,192,②	③,6,①	无	规则精筛	无	**94.24**
13	③	1024,200	③,192,② ⑤,6	③,6,①	⓪	规则精筛	无	**94.49**
14	③	1024,200	③,192,② ⑤,6	③,6,①	⓪	规则精筛	文档拼接	**96.65**
15	③	1024,200	③,192,② ⑤,6	③,6,①	⓪	规则精筛	整合提示	**95.72**

表 2: 复赛实验结果。分块列中两个数字分别代表 chunk_size 和 chunk_overlap。粗排/重排两列用空格分隔多个搜索路径，每个搜索路径逗号隔开的内容分别代表检索/排序方法，topk，文档扩展类型 (若不扩展则没有)

于基于 BERT 的重排器，带来了 3 个点以上的提升。(2) 探究了基于 LLM 的 Embedding 模型密集检索效果，发现 gte-Qwen2-7B-Instruct 由于上下文长度长，且模型更大，效果高于 bge-zh-v1.5 系列。(3) 完善了数据处理流程，对官方处理流程中遗漏数据做了补充，带来了 1 个点的提升。(4) 对分块参数进行了调优，发现 chunk-overlap 可以保留更完整的语义信息，50 调至 200 带来了 2 个点的提升。

4. 两路排序融合 (17-21)。这一部分我们探究了对稀疏和密集两路的不同排序融合策略对于结果的影响。其中粗排融合效果比两路的结果都要低。重排融合中，三种方法均取得了比两路更高的结果。答案拼接和取更长的操作由于需要每一路单独生成一次答案再融合，效率较低，不稳定，因此一般实践中倾向于使用第一种重排融合策略，即将两路分别重排后的结果再进行 RRF 融合，作为最终输入 LLM 的上下文。

2.3 复赛实验

在初赛中，我们将主要的结果在表2中展示，经历了以下 4 个阶段的改进:

1. 粗排方案探究 (0-4)。我们对初赛的数据处理进行了优化，保留了更多的结构化语义信息，探究了初赛中的一些比较好的方案，发现密集检索效果较差，bm25 单路效果就可以达到较好效果。

2. 排序文档扩展 (5-10)。我们探究了在粗排和精排过程中给每个文本块拼接路径字符

方法	初赛准确度
原始	**82.0**
拼接	78.2
总结	79.4

表 3: 查询改写效果

方法	复赛准确度
原始	**92.7**
粗排 +HyDE	89.2
rerank+HyDE	88.2

表 4: 生成虚假文档效果

串对于效果的影响。发现粗排中添加路径能够带来较大增益，精排中拼接文件路径能带来一定增益，最终选定了粗排插入知识路径，精排插入文件路径的排序文档扩展方案。这一部分提升了 2 个点。

3. 图像信息利用 (11-13)。这一部分我们对图像信息进行了探究，发现利用 OCR 中文粗筛后的图像数量多且杂，在经过规则精筛后，相较之前方法提升了 1 个点。

4. 答案优化 (14-15)。这一部分我们发现，将 top1 文本块和答案拼接能够获得 2 个点的提升，考虑到落地中的有效性，我们设计了答案整合提示词，让 LLM 结合 top1 文本块对答案进行补充优化，这一部分提升了 2 个点左右。

2.4 探索实验

2.4.1 查询改写

对于1.2.1提到的查询扩展和 HyDE 方法，我们分别在初赛和复赛阶段进行了测试，其中结果分别如表3和表4所示。总体来说，由于初赛和复赛的题目的查询词已经写的相对具体，查询改写无法带来增益，这些改写方法可能在用户查询词不完整时发挥一定作用。

2.4.2 提示词

对于1.2.6提到的不同的提示词，我们在复赛阶段进行了测试，结果如表5所示。我们发现

提示词种类	复赛准确度
正常问答模板	**94.49**
思维链问答模板	89.75
Markdown 格式问答模板	92.27
侧重问答模板	93.51

表 5: 不同提示词效果

效果最好的仍然是简单的简单问答提示词，思维链问答模板会导致输出过多的解释，Markdown 问答模板会导致输出一些多余的字符，侧重问答模板与原模版相差不大。同时，通过对更多提示词的大量实验，我们发现，GLM4 的回答效果在提示词较简单时更好，较复杂的结构化提示词反而会带来负作用。

3 资源消耗

我们的 RAG 流程中只有 Reranker 需要消耗显存，由于开启了 bfloat16，模型加载只需要消耗 5G 显存[7]，默认批大小 32 时，推理占用的显存开销总共为 12G。

4 部署难度

RAG 框架封装为一个流程类，方便直接加载和使用，可一键部署，我们提供了 docker 部署脚本，docker 大小仅为 28G 左右。我们也提供了基于 FastAPI[8]的 API 部署脚本和基于 Streamlit[9]的 WebUI，可以很方便使用。

5 推理时延

5.1 常规方案

常规时间延迟 复赛的常规方案中，我们将重排的批大小设置为 32，推理时延为一个问题 26s，其中文档排序花费 6s，调用两次 GLM4 花费 20s。

去除答案整合 去掉答案整合步骤，直接返回 top6 生成的答案，只需要调用 GLM41 次，推

[7]我们也尝试了 **8bit 量化**与**剪枝**等模型压缩算法，但减少一定显存同时效果下降明显，值得未来继续研究。

[8]https://fastapi.tiangolo.com/

[9]https://streamlit.io/

理时延为 16s。

增大重排批大小 将批大小增加到 256，显存开销较大，但推理时延可以降低到 24s。

全流程加速方案 除了简单的优化策略，我们还设计了全流程的加速方案，在接下来的三个小节进行介绍，旨在让每个步骤都能够降低时间开销，由于 GLM4 生成的不稳定性，本部分所有的实验在第一次生成答案就终止，没有进行最后的答案整合步骤，从而能够更严谨地比较各个加速方法对于性能的影响。

5.2 BM25 加速

由于我们在检索阶段主要依赖 BM25 进行关键词匹配，因此我们引入 bm25s (Lù, 2024) 库优化 BM25 检索速度

实现方式	时间 (s)	准确度
BM25Okapi	17	94.49
BM25s	0.05	94.24

表 6: 测试集上的 BM25 加速效果。其中时间为 103 个问题和 BM25 相关的总搜索时间，准确度为最终生成答案的评测分数

5.3 Reranker 加速

我们使用了智源研究院开发的 bge-reranker-v2-minicpm-layerwise (Chen et al., 2024) 模型作为 LLM Reranker，此模型支持自定义推理的层数，可以从 8-40 层中根据自己的需求和资源限制选择合适的层数从而降低显存开销。在我们的初赛实验中，发现 **28 层**效果**略好于 40 层**效果，相差在 0.2 分左右，这也和原仓库给出的实证研究结论一致，因此我们初赛和复赛的准确度实验都使用了 28 层。

然而，由于实际推理中 Reranker 比较耗时，我们就思考能否使用更少的层进行推理从而提升速度，BERT 里经典的模型早退工作 FastBERT (Liu et al., 2020) 和 DeeBERT (Xin et al., 2020) 将信息熵是否大于阈值作为早退条件，这种方式计算量较大且效果不稳定，因此我们设计了一种基于最大相似度选择的模型早退算法，即对于每个 query，查看第一个批内第 12 层输出的 softmax 相似度中是否含有大于某个阈值的数值，如果有则此 query 直接使用 12 层进行推理，否则使用 28 层。我们做了一个实验，使用 A100 40G GPU，探究在批大小为 32 时，不同层和不同模型早退方法的推理时间、显存占用和准确度。我们随机选取了 10 个查询，对每个查询选取了 192 个文本块，其中包含完整 RAG 中使用 28 层排序得到的 6 个 ground truth 文本块，和随机选取的其他 186 个文本块。我们用各种方法预测了 ground truth 文本块的相对于所有文本块的 softmax 分数之和。之后我们用预测的比例除以 28 层得到的比例评估相似度准确率，用预测的 ground truth 排序和 28 层排序比较得到排序准确率，得到了如表 7 中的结果。可以发现，我们提出的模型早退方法在降低 33% 推理时间的同时，基本能够维持排序结果与直接使用 28 层一致，超越了熵选择方法。

方法	时间 (s)	相似度 (%)	排序
8 层	**1.67**	73	2.5
12 层	2.20	**88**	3.2
20 层	3.58	86	**4.0**
28 层	5.25	100	6.0
40 层	7.71	100	5.4
Maximum(0.1)	**2.59**	90	3.7
Maximum(0.2)	3.55	96	4.5
Maximum(0.4)	4.57	**97**	**5.4**
熵选择 (0.2)	2.74	89	3.4
熵选择 (0.4)	3.37	91	3.6
熵选择 (0.6)	4.01	91	4.0

表 7: Reranker 加速实验

5.4 上下文压缩

我们设计了一种基于 BM25 语义相似度的上下文压缩方法，我们称之为 BM25-

Extract。对于每个 chunk，先切割成句子，再利用 BM25 计算 query 和每个 chunk 的相似度，最后按照相似度从高到低加入句子列表，直到达到设定的压缩率，最后按照句子原始相对位置拼接在一起。我们将 BM25-Extract 和先进的上下文压缩方法 LLMLingua (Jiang et al., 2023a) 和 LongLLMLingua (Jiang et al., 2023b) 进行了比较，如表 8 所示，我们的方法存在无显存占用，速度更快，准确度更高的优势，在特别需要降低成本的情况下，我们提出的上下文压缩方法对运维任务显然更有效。

压缩算法	压缩率（%）	节省 token	准确度	时间（s）
原始上下文	100	0	94.49	9.30
LLMLingua(0.5)	62.80	143k	83.44	10.47
LongLLMLingua(0.5)	62.80	143k	80.86	10.52
BM25-Extract(0.5)	**55.92**	**160k**	**86.48**	**7.70**
BM25-Extract(0.8)	**83.84**	**59k**	**89.00**	**8.12**

表 8: 测试集上的上下文压缩效果。压缩率指压缩后的 prompt 和原 prompt 字符串长度比例；节省 token 指压缩前后减少的上下文字符串长度除以经验值 1.6 估算的 token 数量；时间指平均每个问题从获取到文档到获取到答案的时间，包含了上下文压缩和 GLM4 生成的时间

6 可扩展性

文档可扩展性 我们的方案主要基于 bm25 检索与 Reranker 重排，只需要对最新的文档进行处理，之后重新进行分块和 idf 值计算，整个过程时间开销较小，可以在 5 分钟内完成全部的处理流程。

用户可扩展性 我们的方案显存占用较小，且在各个环节都设计了相应的推理加速方法，可以根据用户的具体规模决定使用相应的优化策略。即使是使用完全无加速的方案，一张 80G 的显卡也可以支撑至少 6 个 RAG 进程，在半分钟内返回答案给用户。

7 结论

本文提出了 EasyRAG，一个面向 AIOps 网络运维的准确、轻量、高效、灵活且可扩展的检索增强问答框架。

参考文献

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2022. Precise zero-shot dense retrieval without relevance labels. *arXiv preprint arXiv:2212.10496*.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu, Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023a. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376, Singapore. Association for Computational Linguistics.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023b. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. *arXiv preprint arXiv:2310.06839*.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.

Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 2020. FastBERT: a self-distilling BERT with adaptive inference time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6035–6044, Online. Association for Computational Linguistics.

Xing Han Lù. 2024. Bm25s: Orders of magnitude faster lexical search via eager sparse scoring.

Sheila Teo. 2023. How i won singapore' s gpt-4 prompt engineering competition.

Liang Wang, Nan Yang, and Furu Wei. 2023. Query2doc: Query expansion with large language models. *arXiv preprint arXiv:2303.07678*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2246–2251, Online. Association for Computational Linguistics.

A 问答提示词模板

A.1 Markdown 格式问答模板

```
## 目标

请你结合上下文中 k 个 5G 运维私域文档的信息，回答给定的问题

## 要求

1. 可以分点作答，尽量详细且具体
2. 不要简单复述上下文信息
3. 不要使用自己的知识，只能根据上下文文档中的内容作答

## 上下文

{context_str}

## 问题

{query_str}

## 回答
```

A.2 思维链问答模板

```
上下文信息如下：
---------
{context_str}
---------

请你基于上下文信息而不是自己的知识，回答以下问题，请一步步思考，先给出分析过程，再生成答案：

{query_str}

回答：
```

A.3 侧重问答模板

```
上下文信息如下：
---------
{context_str}
---------

请你基于上下文信息而不是自己的知识，回答以下问题，可以分点作答，0号文档的内容比较重要，可以重点参考，如果上下文信息没有相关知识，可以回答不确定，不要复述上下文信息：

{query_str}

回答：
```

B 答案整合模板

上下文:

{top1_content_str}

你将看到一个问题，和这个问题对应的参考答案

请基于上下文知识而不是自己的知识补充参考答案，让其更完整地回答问题

请注意，严格保留参考答案的每个字符，并将补充的内容和参考答案合理地合并，输出更长更完整的包含更多术语和分点的新答案

请注意，严格保留参考答案的每个字符，并将补充的内容和参考答案合理地合并，输出更长更完整的包含更多术语和分点的新答案

请注意，严格保留参考答案的每个字符，并将补充的内容和参考答案合理地合并，输出更长更完整的包含更多术语和分点的新答案

问题:
{query_str}

参考答案:
{answer_str}

新答案: