自研Token级别语义检索引擎分享

汇报人：尹越

2024年12月26日

之江实验室 ZHEJIANG LAB

介绍提纲

需求背景

3类RAG基础检索引擎对比

elasticsearch

全文检索引擎

优势：

1）基于Lucene，具备全文检索能力，使用便捷

2）工业界实际使用较为广泛，可靠性高

3）具备分布式存储能力，可支持海量数据检索

劣势：

1）基于文字匹配的全文检索对于各阶段分词、查询扩展、索引扩展等额外模块的效果要求较高。否则容易出现意思相同，但是字面不相同无法找回的情况

2）向量检索引入较晚，检索速度相对较慢

Milvus

向量检索引擎

优势：

1）可以支持海量的向量计算相似度匹配

2）理论上可以减少文本匹配遇到的相同语义，字面匹配无法召回的情况

劣势：

1）实际上对于长度相近的短文本、词等的向量匹配效果尚佳。但是对于如用户问题这样的短文本，和书籍章、节等这样的长文本进行相关性匹配的时候，会有非常明显的误召回。

InfinityFlow

向量检索引擎

优势：

1）支持Colbert模型进行MaxSim的Token级别的语义相似度计算。

2）可以在百ms内完成万级别的doc排序，以接近模型内交互排序的效果实现更大的排序窗口。

劣势：

1）Colbert的Token向量仅应用在排序过程。底层召回依然要依赖于字面层次的全文检索，或者文本向量检索。这两者存在的问题还是依然存在。

需求背景

思考与挑战

思考：

是否可以在Token级别上进行向量检索，获得一批Token之后，再进行类似全文检索的倒排链表合并打分的召回粗排序过程。这样将Colbert的Token向量完整的覆盖召回和粗排的过程。

挑战：

1）空间

将每一个Token对应的向量都要保存下来，如果向量维度不变，从存储上来看较每个文本保存一个向量会随着Token的数量有几百甚至上千倍的增长。

同时，召回的Token因为不像全文检索那样一个TermID在各个文本中都一样，每一个Token在不同segment（文本分段）中对应的向量受上下文的影响，都会有所不同，哪怕原来对应位置上的文本内容是相同的。

2）时间

因为空间上的膨胀，带来了时间上的处理也相应增加。势必要求实现的过程中，索引检索和促排打分，不能完全和Token的数量线性相关，要依然保持百ms级别的响应速度。

挑战：Token的Tensor差异示例

Doc1:CMOS超大规模集成电路设计/第1章 引论·

[[-0.0467529296875, 0.1533203125, -0.1064453125, -0.273193359375,0.101318359375,...]...]

Doc2:CMOS超大规模集成电路设计/第6章 互连线

[[0.040283203125, 0.1572265625, -0.12115478515625, -0.304443359375,0.1209716796875, ...]...]

介绍提纲

前期调研

解决空间方面挑战的调研验证

Token向量的维度：
在大模型中，Token的维度一般是千级别的，但是在Colbert中，可以减少到128维。

常见大模型及其Token维度等参数

1.GPT-3（OpenAI）
•参数量：1750亿（175B）
•每个Token的维度：通常为1280维
•上下文长度：2048 Tokens
2.BERT（Google）
•参数量：1.1亿（Base版本）或3.4亿（Large版本）
•每个Token的维度：768维（Base版本）或1024维（Large版本）
•上下文长度：512 Tokens
3.T5（Google）
•参数量：11亿（Base版本）或110亿（Large版本）
•每个Token的维度：512维（Base版本）或1024维（Large版本）
•上下文长度：512 Tokens

4.RoBERTa（Facebook AI）
•参数量：1.1亿（Base版本）或3.4亿（Large版本）
•每个Token的维度：768维（Base版本）或1024维（Large版本）
•上下文长度：512 Tokens
5.Llama（Meta AI）
•参数量：7B、13B、30B、65B等不同版本
•每个Token的维度：通常为4096维
•上下文长度：2048 Tokens

前期调研

解决空间方面挑战的调研验证

Token向量的维度：
在大模型中，Token的维度一般是千级别的，但是在Colbert中，可以减少到128维。

常见大模型及其Token维度等参数

1.GPT-3（OpenAI）
•参数量：1750亿（175B）
•每个Token的维度：通常为1280维
•上下文长度：2048 Tokens
2.BERT（Google）
•参数量：1.1亿（Base版本）或3.4亿（Large版本）
•每个Token的维度：768维（Base版本）或1024维（Large版本）
•上下文长度：512 Tokens
3.T5（Google）
•参数量：11亿（Base版本）或110亿（Large版本）
•每个Token的维度：512维（Base版本）或1024维（Large版本）
•上下文长度：512 Tokens

4.RoBERTa（Facebook AI）
•参数量：1.1亿（Base版本）或3.4亿（Large版本）
•每个Token的维度：768维（Base版本）或1024维（Large版本）
•上下文长度：512 Tokens
5.Llama（Meta AI）
•参数量：7B、13B、30B、65B等不同版本
•每个Token的维度：通常为4096维
•上下文长度：2048 Tokens

前期调研

解决空间方面挑战的调研验证

Token向量的精度：

进行MaxSim的打分方式，对于从精度float64降低到float16，打分结果相差没有特别大，但存储空间可以降低为原先的1/4。

精度float64时计算结果

```
Q:CMOS超大规模集成电路设计/第1章 引论·
0       23.4489695610077        CMOS超大规模集成电路设计/第1章 引论·
45      22.850114469899463      CMOS超大规模集成电路设计/第1章 引论/2.1 引言
71      22.691941783104774      CMOS超大规模集成电路设计/第1章 引论/3.1 引言
228     22.647621259677155      CMOS超大规模集成电路设计/第1章 引论/7.1 引言
166     22.637848847656638      CMOS超大规模集成电路设计/第1章 引论/5.1 引言
205     22.597997755227887      CMOS超大规模集成电路设计/第1章 引论/6.1 引言
204     22.586938408154545      CMOS超大规模集成电路设计/第1章 引论/5.7 历史透视
40      22.555785990679908      CMOS超大规模集成电路设计/第1章 引论/1.10 物理设计/1.10.5 阵列
42      22.541141819687038      CMOS超大规模集成电路设计/第1章 引论/1.11 设计验证
5       22.53393574423149       CMOS超大规模集成电路设计/第1章 引论/1.1 集成电路简史
280     22.511648864932905      CMOS超大规模集成电路设计/第1章 引论/7.8 历史透视
128     22.491758726053952      CMOS超大规模集成电路设计/第1章 引论/4.1 引言
37      22.43600124255804       CMOS超大规模集成电路设计/第1章 引论/1.10 物理设计/1.10.2 标准单元
38      22.39063837977563       CMOS超大规模集成电路设计/第1章 引论/1.10 物理设计/1.10.3 节距匹配
35      22.3736525135125 75     CMOS超大规模集成电路设计/第1章 引论/1.9 电路设计
30      22.35016649101137       CMOS超大规模集成电路设计/第1章 引论/1.8 逻辑设计
39      22.349907049232502      CMOS超大规模集成电路设计/第1章 引论/1.10 物理设计/1.10.4 分片规划
36      22.33434699675538       CMOS超大规模集成电路设计/第1章 引论/1.10 物理设计/1.10.1 平面规划
199     22.32015568743133       CMOS超大规模集成电路设计/第1章 引论/5.5 低功耗体系结构
```

精度float16时计算结果

```
Q:CMOS超大规模集成电路设计/第1章 引论·
0       23.4482421875   CMOS超大规模集成电路设计/第1章 引论·
45      22.850341796875 CMOS超大规模集成电路设计/第1章 引论/2.1 引言
71      22.69140625     CMOS超大规模集成电路设计/第1章 引论/3.1 引言
228     22.6484375      CMOS超大规模集成电路设计/第1章 引论/7.1 引言
166     22.6376953125   CMOS超大规模集成电路设计/第1章 引论/5.1 引言
205     22.597900390625 CMOS超大规模集成电路设计/第1章 引论/6.1 引言
204     22.587890625    CMOS超大规模集成电路设计/第1章 引论/5.7 历史透视
40      22.555908203125 CMOS超大规模集成电路设计/第1章 引论/1.10 物理设计/1.10.5 阵列
42      22.540771484375 CMOS超大规模集成电路设计/第1章 引论/1.11 设计验证
5       22.53466796875  CMOS超大规模集成电路设计/第1章 引论/1.1 集成电路简史
280     22.5107421875   CMOS超大规模集成电路设计/第1章 引论/7.8 历史透视
128     22.490478515625 CMOS超大规模集成电路设计/第1章 引论/4.1 引言
37      22.434814453125 CMOS超大规模集成电路设计/第1章 引论/1.10 物理设计/1.10.2 标准单元
38      22.391845703125 CMOS超大规模集成电路设计/第1章 引论/1.10 物理设计/1.10.3 节距匹配
35      22.373291015625 CMOS超大规模集成电路设计/第1章 引论/1.9 电路设计
30      22.350830078125 CMOS超大规模集成电路设计/第1章 引论/1.8 逻辑设计
39      22.3505859375   CMOS超大规模集成电路设计/第1章 引论/1.10 物理设计/1.10.4 分片规划
36      22.333740234375 CMOS超大规模集成电路设计/第1章 引论/1.10 物理设计/1.10.1 平面规划
199     22.3212890625   CMOS超大规模集成电路设计/第1章 引论/5.5 低功耗体系结构
```

前期调研

解决时间方面挑战的调研验证

矩阵乘：

之前使用numpy进行矩阵乘，当矩阵比较大的时候，一次矩阵乘可能要花0.3s，将numpy换为cupy利用gpu加速之后，耗时降低到0.002左右

```
CMOS超大规模集成电路设计/第6章 互连线
Time taken to create q tensor: 0.0001945495605468 75 seconds
Time taken to create q and doc token simi matrix: 0.0025169849395751953 seconds
Time taken to compute segment score: 0.0544586181640625 seconds
Time taken to search segment: 0.05730700492858887 seconds
CMOS超大规模集成电路设计/第11章 数据通路子系统
Time taken to create q tensor: 0.0002171993255615234 4 seconds
Time taken to create q and doc token simi matrix: 0.0024607181549072266 seconds
Time taken to compute segment score: 0.05425786972045898 4 seconds
Time taken to search segment: 0.057030677795410156 seconds
```

前期调研

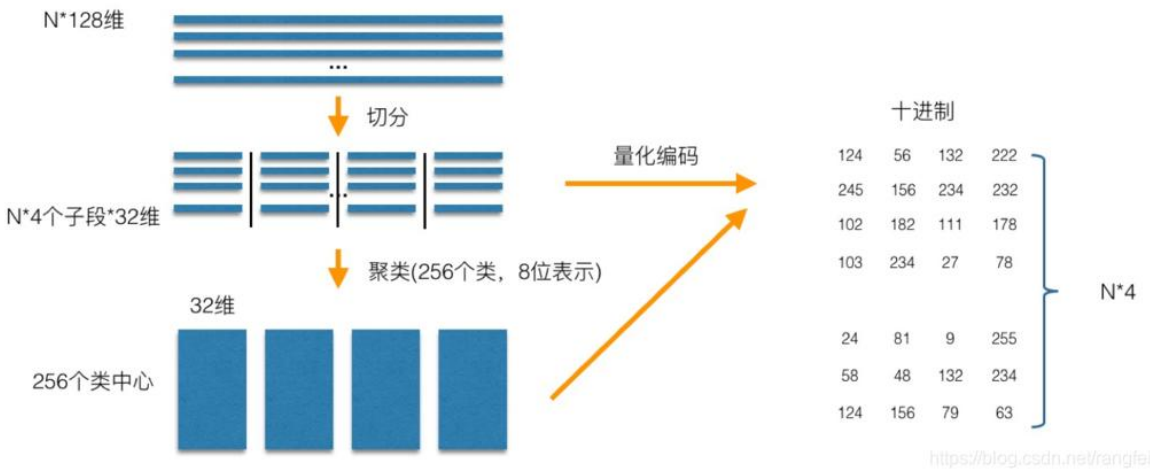
解决时间方面挑战的调研验证

向量聚类、乘积量化（Produce Quantizer、PQ）：

当faiss使用最简单的IndexFlatL2，因为是暴力检索，虽然精度比较高，但是要都加载到内存中，随着向量数量增加时间增长明显。如果使用IndexIVFPQ这样通过聚类+乘积量化的方式，可以将检索时间降低到十几ms。

每来一个查询向量，首先计算其与IVF阶段粗聚类簇心的距离，然后选择距离最近的top_n个簇，再基于PQ计算查询向量与这几个簇底下的向量的距离。

PQ原理图




```python
IndexIVFPQ(
    Index *quantizer,      # 指向量化器的指针，用于生成聚类中心。通常是一个 brute-force 索引，如 IndexFlatL2。
    size_t d,              # 向量的维度。即每个向量包含的元素数量。
    size_t nlist,          # 聚类中心的数量。即 IVF 部分的聚类数量，决定了倒排文件的数量。
    size_t M,              # 向量分割成的子向量数量。在 PQ 中，每个向量被分割成 M 个子向量。
    size_t nbits_per_idx, # 每个子向量量化索引的比特数。决定了每个子空间的量化中心数量，通常是 8 或 4。
    MetricType metric = METRIC_L2 # 度量类型，默认为 L2 距离（欧氏距离）。也可以选择其他度量类型，如 INNER_PRODUCT（内积）。
)
```

前期调研

解决时间方面挑战的调研验证

数据列存：

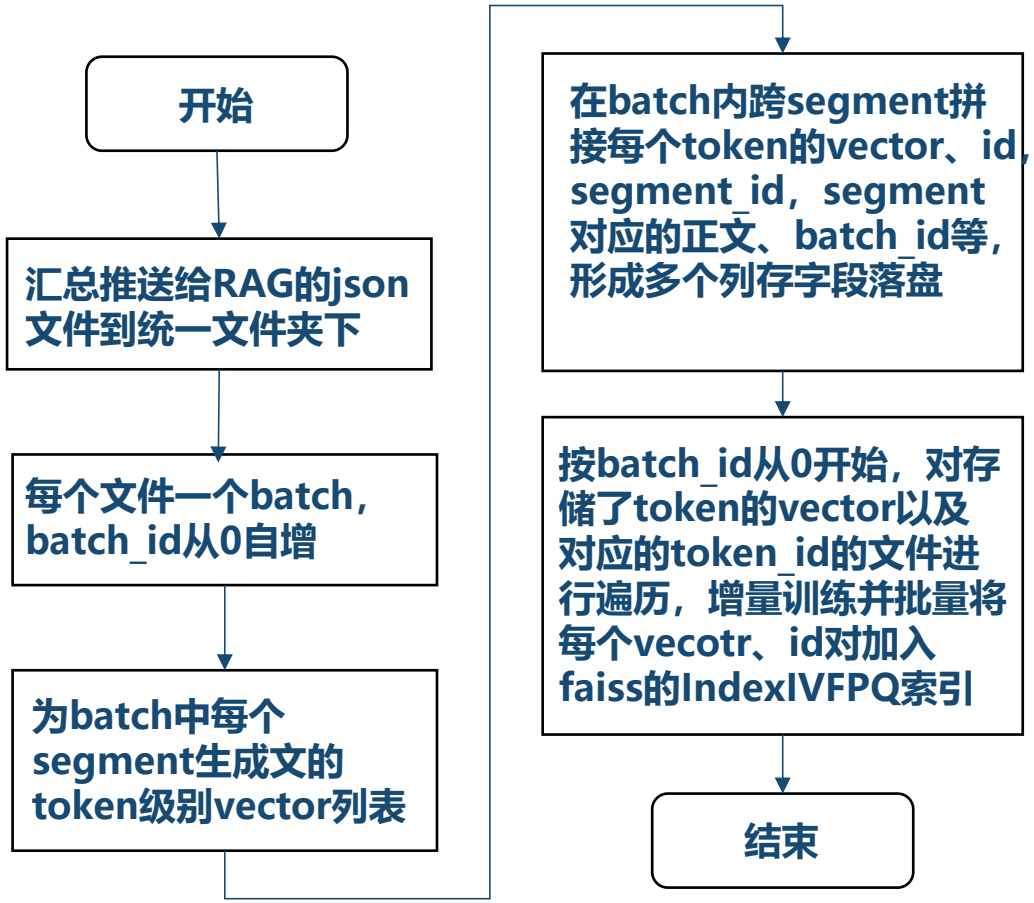
如果依然使用每行一个记录的方式去存储token和所在的segment，segment所在的书籍，每个segment的id以及原文等信息的话，定位每次所需的信息就需要顺序遍历文件去定位获取。如果改成h5列存的方式，对数据做好分区，可以在几ms内完成如根据segmentid获取原始文本的过程。

```
CMOS超大规模集成电路设计/第11章 数据通路子系统
len(de_token_ids)
244
len(de_segment_ids)
140
len(batch_ids2seg_ids)
1
search_faiss_time_str
Time taken to search faiss: 0.01990652084350586 seconds
get_de_token_ids_time_str
Time taken to get de token_ids time str: 0.0006489753723144531 seconds
get_de_segment_ids_time_str
Time taken to get de segment_ids time str: 0.00016379356384277344 seconds
get_batch_ids2seg_ids_time_str
Time taken to get batch_ids2seg_ids time str: 0.0003597736358642578 seconds
load_matched_time_str
Time taken to load matched time str: 0.015313386917114258 seconds
compute_all_score_time_str
Time taken to compute all scores time str: 0.014393329620361328 seconds
Time taken to get segment ids: 0.0509638786315918 seconds
Time taken to get text: 0.0008172988891601562 seconds
Time all: 0.05178117752075195 seconds
```

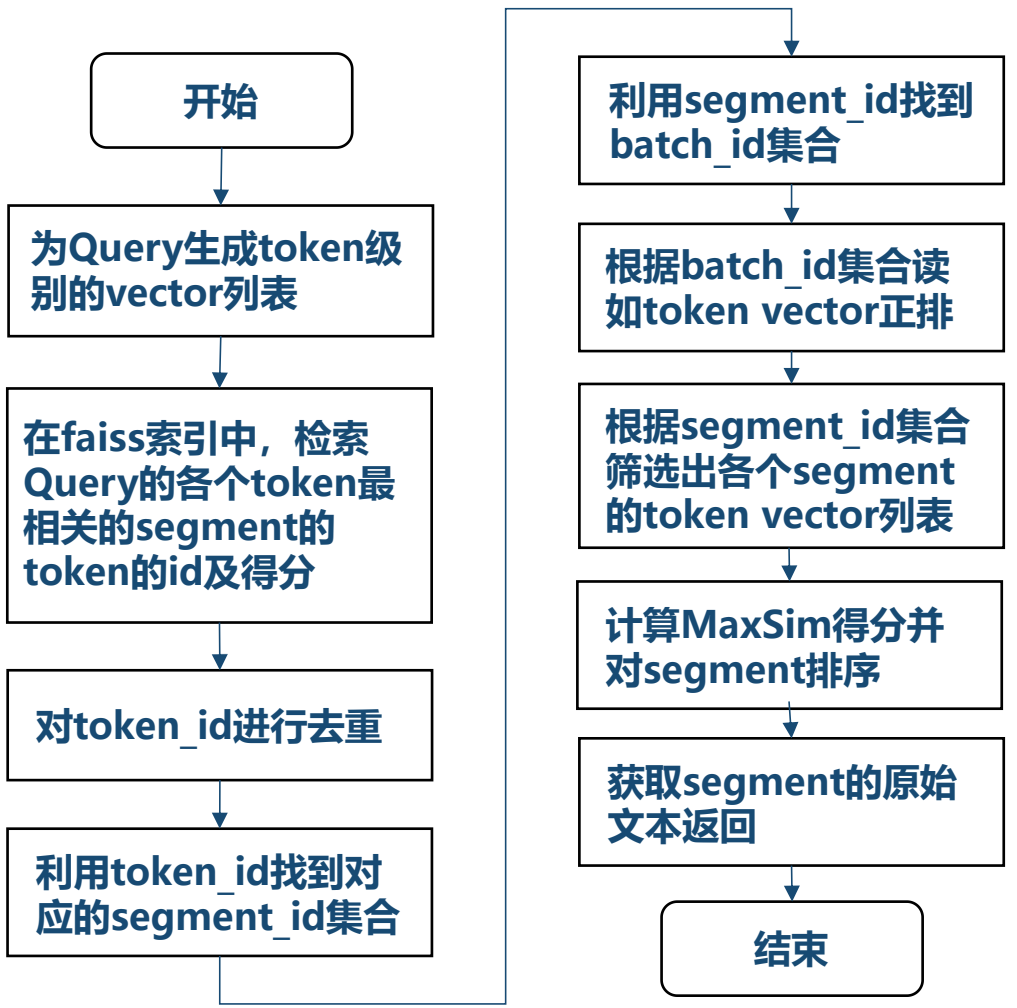
介绍提纲

实验效果

索引构建流程



索引查询流程



实验效果

测试时间	20241225_1423
测试人	尹越
测试集	Test01
题目数量	130
模型	Qwen2.5-72B-AWQ
测试目标	验证新的自研检索引擎（faiss+colbert）的方式，是否比当前ES的检索引擎召回和粗排序的效果更好。
输入输出及算法流程	1）对Test01对应书籍调用Colbert模型，为context字段生成各自的doc_tensor字段。 2）将新生成的包含了doc_tensor字段的结果灌入到自研的检索引擎中。 3）将之前两阶段检索过程中调用es获取参考段落的过程替换为从自研检索引擎中获取。 4）分别测试直接使用自研引擎粗排结果以及和线上一样经过精排后的结果送入最后的生成模块。 附：线上精排过程：对粗排结果取top_n（本次实验是top20）再进行自然段分段后合并为150个字一个的小段落，再调用排序服务rerank，取top_k（本次实验是top50）的方式
直接使用自研引擎召回粗排结果	第一次仅用模型生成正确个数:87 第一次仅用模型生成生成正确率:66.9% 第二次检索增强生成正确个数:109 第二次检索增强生成正确率:83.8%
在自研引擎召回粗排之后增加线上相同精排过程的结果	第一次仅用模型生成正确个数:87 第一次仅用模型生成生成正确率:66.9% 第二次检索增强生成正确个数:109 第二次检索增强生成正确率:83.8%
历史最佳第二次检索生成正确率	77%
测试结论	无论是否增加精排模块，使用了自研检索引擎之后正确率都达到了83.8%，都较之前使用ES进行检索的最佳生成正确率的77%，有6.8%的提升。 基本可以说明基于语义token粒度的检索+maxsim打分的方式，较ES能够更好的提升长短文本相关性匹配的效果。

介绍提纲

一、需求背景

二、前期调研

三、实验效果

四、后续计划

后续计划

短期：
将更多的文档进行索引化并进行测试
增加文件并行读取、segment_id集合粗排等策略优化性能
Maxsim打分方式优化

长期：
构建更规范化通用的工程框架
开源相关框架
完成论文和专利

之江实验室 ZHEJIANG LAB

谢谢