

👉 前情提要：[Transformer与注意力机制概述](#)

📚 相关论文：

## 1. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

- 提出了基于双向深度Transformer的BERT交叉编码器
- [BERT的总结](#)

## 2. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT

- 提出了基于BERT编码的后期Token级交互模式
- [ColBERTv1的总结](#)

## 3. ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction

- 保留了ColBERT的后期交互架构，但从训练策略/嵌入压缩/数据集上优化
- [ColBERTv2的总结](#)

## 4. PLAID: An Efficient Engine for Late Interaction Retrieval

## 5. EMVB: Efficient Multi-Vector Dense Retrieval Using Bit Vectors

### 1. 背景与导论

#### 2. ColBERTv2

#### 3. LoTTE: 长尾域外评估

#### 4. 模型评估

# 1. 背景与导论

## 1.1. 研究背景

### 1.1.1. IR系统的范式

#### ① 信息检索(IR)

1. IR的含义：理解用户[查询↔文档/段落]之间的语义关系及其匹配

2. IR的范式：

- 传统信息检索：基于关键词捕捉语义的匹配
- 神经信息检索：基于深度学习捕捉语义的匹配



② 单向量IR的优化：高度调优的监督策略(负样本挖掘/蒸馏)→单向量模型可以匹敌ColBERT

1. 正/负样本挖掘：

- 正/负样本对：分别指[查询↔段落(文档)]/[查询↔段落(文档)]两种Couple
- 训练挖掘方式：在训练阶段，以让模型学会区分正/负样本为目标

2. 蒸馏训练：

- 含义：训练一个小型的Student模型，学习来自更大Teacher模型的知识以模仿其行为
- 目的：在保证模型的表达能力的同时，减小模型的规模

### 3 后期交互模型的缺陷

1. 存储需求：传统ColBERT需要离线存下所有嵌入，空间占用比单向量模型大一个数量级
2. 检索误差：Token级后期交互→存在固有的Token级偏差
  - 含义：即ColBERT在后期交互中，会更关注词级相似性，而非全局(上下文)相似性
  - 影响：适用于单向量模型的调优监督训练策略，可能难以给ColBERT带来太多性能提升

## 1.1.2. IR系统的测试

### 1 IR系统的两种测试

测试	含义	备注
域内检索	在模型的训练集/训练任务上检索	训练集 可延伸为与训练集相似的开发集
域外检索	在模型未见过的测试集/测试任务上检索	零样本 aka 模型未见过的测试集

### 2 IR系统的零样本泛化

1. 泛化目标：
  - 一般认为的：模型能在特定领域的零样本上表现良好
  - 本文附加的：模型能在长尾主题上表现良好
2. 长尾主题：
  - 含义：某个领域内数量庞大但极为低频(非主流)的话题，可类比于Zipf分布定律
  - 现状：在公共数据集中罕见，比如维基百科没有"意大利面拌42号混凝土"这一词条(主题)

### 3 域外检索的更多背景

1. 特点：IR领域的评估，高度依赖于大规模下域外数据集(如BEIR)
2. 困境：
  - 分布局限：现有数据集的查询主题集中于热门话题，以至于无法代表实际用户检索需求
  - 领域差异：用户的IR/QA请求往往涉及特定领域，某些特定领域几乎没有标记数据库

## 1.2. 本文的主要贡献

### 1 对于ColBERTv2的架构

方面	本文的做法(相较ColBERTv1)
模型结构	保持一致
训练策略	参考单向量模型的训练调优，增加了基于蒸馏的降噪监督+强负样本挖掘机制
向量表示	通过应用残差压缩得到轻量级的Token嵌入表示，大幅降低存储需求
运行机制	针对残差压缩改进了离线嵌入+后期交互(查询)过程

### 2 对于ColBERTv2的测试：提出了新的LoTTE数据集，旨在更关注自然搜索查询&长尾主题

## 1.3. 文献综述

## 1 IR中的Token嵌入

1. 传统神经IR的嵌入方法:

方法	描述	成本	语义捕捉	典型模型
单向量嵌入	查询/文档(段落) $\xrightarrow{\text{编码}}$ 单一高维向量	低	弱	ANCE
多向量嵌入	查询/文档(段落) $\xrightarrow{\text{编码}}$ 多个稠密向量	高	强	BERT(交叉编码)

2. 对多向量IR嵌入的改进:

- 后期交互: ColBERT
- 注意力重排: 先用词袋法对文档大规模初排, 再用注意力(BERT)小规模重排, 如MORES

3. 其它基于BERT的嵌入:

模型	概述
ME-BERT	为每个文档生成词级多向量表示, 但为查询只生成单向量表示
COIL	为文档/查询都生成词级嵌入, 但是交互仅限于[文档 $\longleftrightarrow$ 查询]词汇
uniCOIL	交互原理同COIL, 区别在于将COIL的词级嵌入压缩到一维(权重标量)
SPLADE	为查询/文档生成词汇级的稀疏向量, 查询和文档进行词级的后期交互

## 2 IR的向量压缩表示

1. PQ压缩方法:

方法	描述
原始PQ压缩	训练查&文档编码器 $\rightarrow$ 计算嵌入 $\rightarrow$ 再对嵌入执行PQ压缩(得到质心)
共同PQ压缩	编码器训练/PQ压缩过程合二为一, 在编码器训练同时就得到质心分布
降维PQ压缩	原始嵌入 $\xrightarrow{\text{自编码器}}$ 低维语义空间向量表示 $\rightarrow$ 在低维语义空间执行PQ压缩
残差PQ压缩	用PQ压缩原始嵌入 $\rightarrow$ 计算压缩前后残差 $\xrightarrow{\text{由此}}$ 原始数据=质心索引+残差编码

2. BPR二进制编码: 原始嵌入(浮点型32/64 bit)  $\xrightarrow[\text{HASH}]{\tanh \text{激活函数}}$  二进制编码(2/4/8 bit)

## 3 单向量IR的训练调优

1. 硬负采样:

- 含义: 训练过程中引入一定数量硬负样本对, 以提升模型对于负样本的区分能力
- 硬负样本: 与查询语义相似但与文档实际不相关的样本, 示例如下

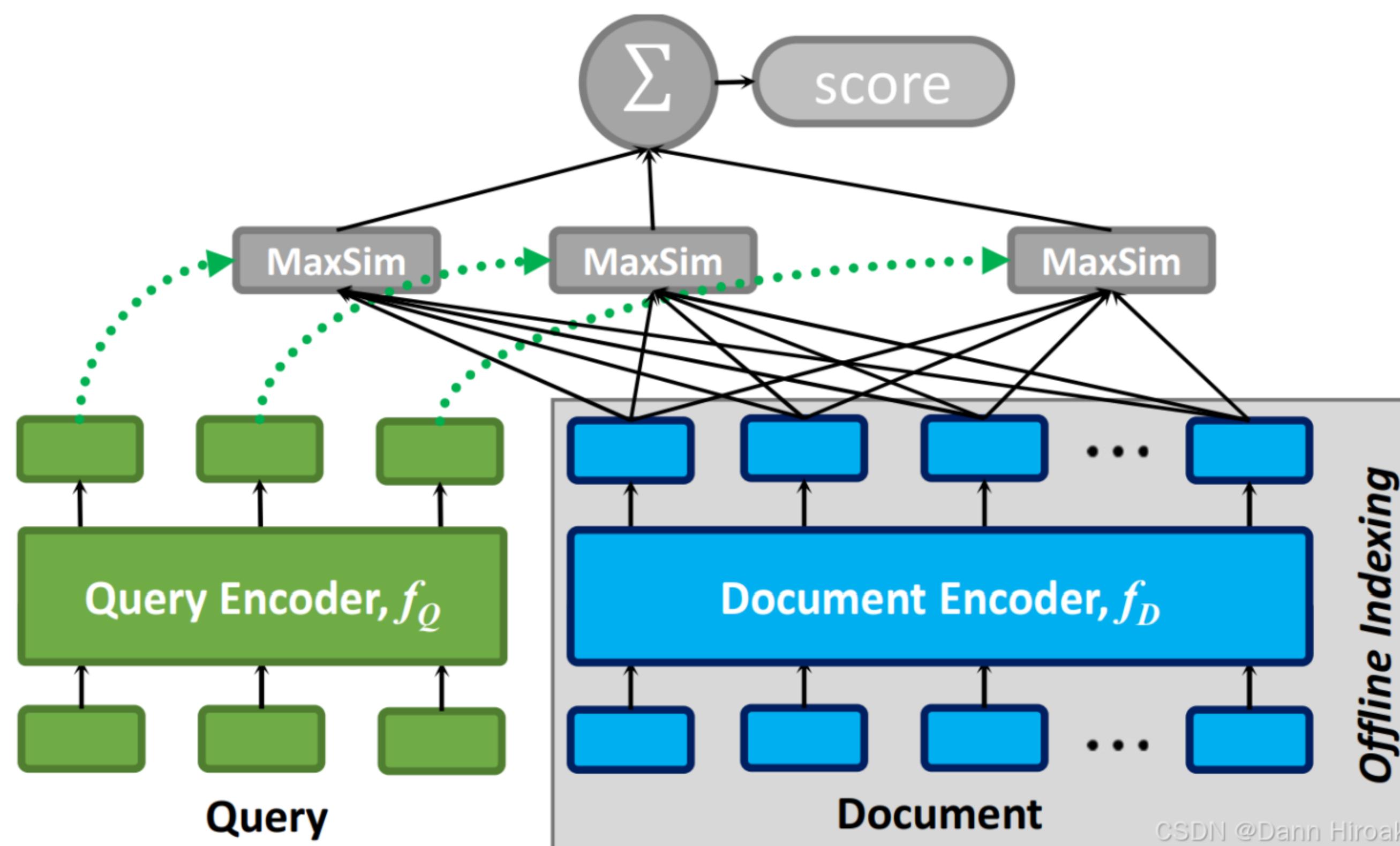
- 1 | 查询: 机器学习的优点
- 2 | ->负样本: 完全不相关的文本(比如关于旅游的文章)
- 3 | ->硬负样本: 一段介绍深度学习但是没有阐明其优点的文章

2. 超大模型蒸馏: 将强大的教师模型GPT/BERT的表现力, 蒸馏给极其简单的单向量学生模型

3. 预训练改进: 对特定任务训练/对抗训练/多任务学习

# 2. ColBERTv2

## 2.1. 模型结构: 与ColBERTv1一致



### 1 编码系统

1. 编码结构: 查询/段落(文档)别独立地输入BERT→得到上下文嵌入→皆投影到较低维度
2. 编码流程:

阶段	操作	备注
离线	编码数据库中所有段落 $d$ 为词级嵌入 $D_{M \times k'}$	$M$ 为段落词数, $k'$ 为(压缩)嵌入维度
在线	在给出查询 $q$ 后再将其编码为词级嵌入 $Q_{N \times k}$	$N$ 为查询词数, $k$ 为嵌入维度

2 交互系统:  $S_{q,d} = \sum_{i=1}^N \max_{j=1}^M Q_i \cdot D_j^T$

1. MaxSim:

模型	处理方法
ColBERTv1	让 $d$ 每个嵌入和 $q$ 所有嵌入计算相似度并取最大值, 最终得到 $N$ 个MaxSim
ColBERTv2	原理一致, 但在执行层面多了一个重排优化(详细过程见后面)

2. 最终得分: 将所有MaxSim求和, 即得到查询对一个文档的最终相似度

## 2.2. 模型训练: 通过蒸馏+降噪优化

## 2.2.1. 传统ColBERTv1的训练

### 1 训练方式

1. 数据结构：来自MS MARCO，与查询 $q$ 匹配的三元组 $\langle q, d^+, d^- \rangle$

样本(段落)类型	含义	来源
正样本 $d^+$	与查询相关的段落	数据集的人工标注
负样本 $d^-$	与查询无关的段落	来源于通过BM25检索算法得到的未标注段落

2. 训练目标：增强 $\langle q, d^+ \rangle$ 相似性，降低 $\langle q, d^- \rangle$ 相似性

### 2 存在的问题：

1. BM25采样所得负样本不够有挑战性
2. 三元组范式倾向于奖励假阳性/假阴性：

## 2.2.2. ColBERTv2的改进：借鉴单向量的高度监督调优

### 2.2.2.1. 样本生成

#### 1 流程

1. 段落初选：[基于待训练的查询]  $\xrightarrow{\text{BM25采样}}$   $k$ 个相关/无关段落  $\rightarrow$  构成 $k$ 个[查询 $\leftrightarrow$ 段落]对
2. 段落重排：用蒸馏后得到的MiniLM对所有[查询 $\leftrightarrow$ 段落]对评分，截取排名前 $w$ 的段落

#### 2 蒸馏：让小型交叉编码器MiniLM学习大型编码器的评分分布，用来给初选段落评分

1. 对齐策略：

- 难题：交叉编码器的评分为实数/ColBERT分数(余弦相似度) $\in [-1, 1]$ ，无法直接蒸馏
- 策略：采用KL散度，即一种衡量两个概率分布之间差异的损失函数，让二者分数对齐

2. 实现流程：

- 对齐：老师模型(交叉编码器)  $\xleftarrow[\text{SoftMax}]{\text{KL散度}}$  学生模型(MiniLM)  $\rightarrow$  二者评分分布归一化到 $[-1, 1]$
- 蒸馏：训练MiniLM，使其对齐后的分数分布，与交叉编码器的趋同

### 2.2.2.2. 模型训练

#### 1 训练的数据结构

1. 训练基本单元：与查询 $q$ 匹配的 $w$ -路元组 $\langle q, \mathbf{d}_w \rangle$ 即 $\langle q, d_1^+, d_2^-, \dots, d_w^- \rangle$

样本(段落)类型	来源
正样本 $d_1^+$	交叉编码器MiniLM评分较高的段落，或者标注的正样本
负样本 $d_2^-, \dots, d_w^-$	交叉编码器MiniLM评分较低的段落， <b>里面大概率含硬负样本</b>

2. 批内负样本：查询以 $Q$ (批)为单位进入GPU，对其中每个查询 $q_i$ ，将以下二者都视做其负样本

负样本	释义
自身负样本	$\langle q_i, d_{i1}^+, d_{i2}^-, \dots, d_{iw}^- \rangle$ 中的 $\langle d_{i2}^-, \dots, d_{iw}^- \rangle$
批内负样本	所有 $q_k (k \neq i)$ 的所有对应段落，无论该段落对 $q_k$ 是正或负

#### 2 降噪训练的细节

## 1. 训练目标:

- 损失函数: 每个查询的正样本/负样本(自身+批内)的交叉熵
- 优化方向: 扩大正样本/负样本的交叉熵距离

## 2. 刷新机制:

- 操作: 训练若干轮后用"半吊子"预训练模型重新再进行一遍样本生成→用新样本继续训练
- 意义: 加速训练, 防止过拟合

## 2.3. 模型运行: 残差压缩&基于压缩的运行机理

### 2.3.1. 残差压缩原理

#### 1 残差+残差编码流程: 一种对PQ压缩的自然扩展

1. 运行聚类: 对所有文档的所有嵌入组成的空间执行聚类, 每个嵌入 $t$ 分配到其最近的质心 $C_t$

⚠ 注意: 此处质心是对于语料库中所有文档的所有嵌入而言, 并非一个文档对应一组质心

2. 残差编码: 计算每个嵌入的 $t$ 的残差 $r=t-C_t$ , 并将其近似量化编码为 $\tilde{r} \approx t-C_t$

3. 压缩表示:

- 存储时:  $t$ 的嵌入  $\xrightarrow{\text{编码为}}$  (离 $t$ 最近质心 $C_t$ 的索引)+(残差向量 $r=t-C_t$ 的量化近似 $\tilde{r} \approx t-C_t$ )
- 检索时: 近似地将 $t$ 还原为 $\tilde{t}=C_t+\tilde{r}$

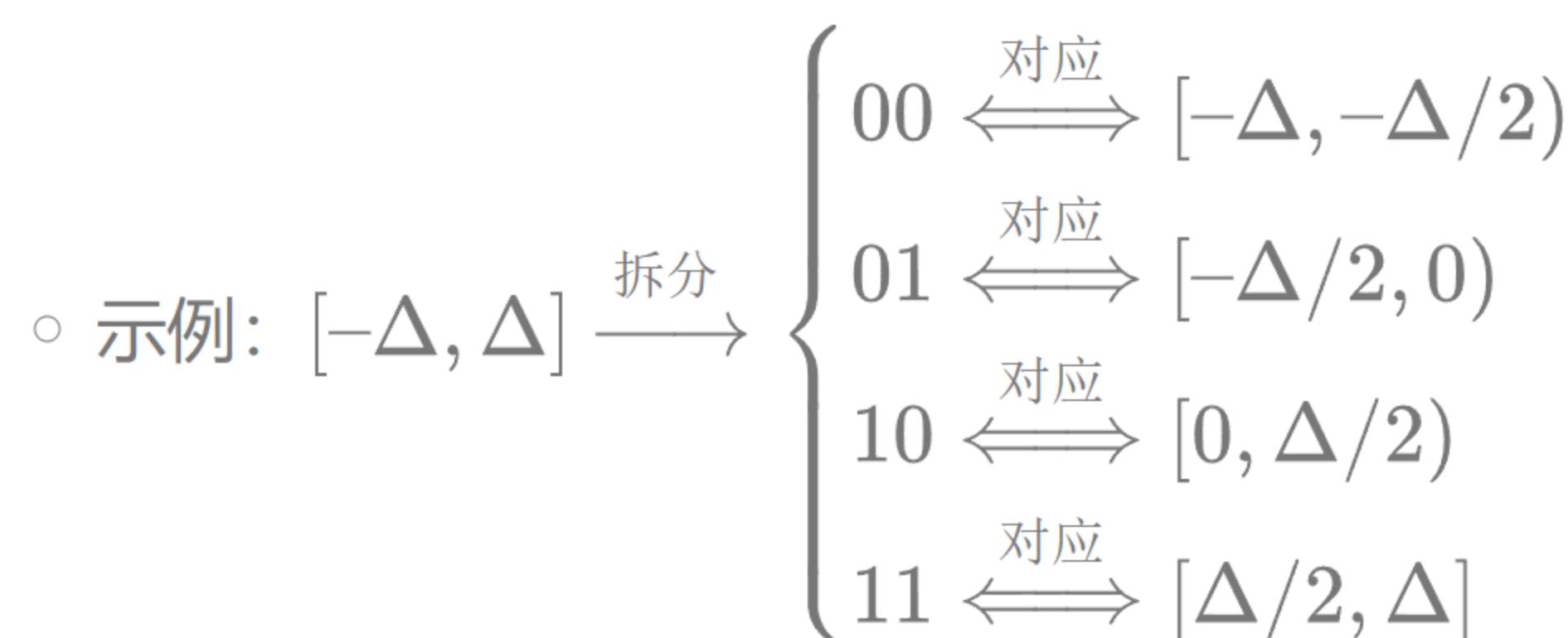
#### 2 残差的近似量化

1. 背景: BERT交叉编码向量的语义聚集性假设

- 含义: BERT对Token的嵌入倾向于聚集, 而对相同Token在不同上下文的嵌入更甚之
- 意义: 使 $|r_i|$ (残差每个维度) $\ll |t_i|$ (嵌入每个维度)  $\Rightarrow$  对残差极激进地压缩也不会损失太多信息

2. 量化: 文中实验了2bit/1bit两种压缩, 以下2bit压缩为例

- 操作: ( $r$ 每个维度的连续分布  $\xrightarrow{\text{拆分为}}$  4个离散状态)  $\xleftrightarrow{\text{对应}}$  2个bit所表示的四种状态



#### 3 空间占用分析

1. 残差压缩的空间占用:  $\lceil \log |C| \rceil + bn \rightarrow$  实际层面为20/36Byte

编码类型	超参数	大小(bit)	实际层面
质心索引	$C$ 个质心	$\lceil \log  C  \rceil$	支持至多 $2^{32}$ 个质心 $\rightarrow$ 占用32bit
残差量化	$b$ bit压缩+残差 $n$ 维	$bn$	残差128维 $\rightarrow$ 2/1bit压缩时占128/256bit

2. 原始嵌入的空间占用: 嵌入为128维, 每维为单/双精度浮点数(16/32bit)时, 占256/512Byte

### 2.3.2. 残差压缩后的离线索引实现

### 2.3.2.1. ColBERTv2的离线索引流程

#### 1 第一步：质心选择

1. 质心数量：实验证明如果嵌入数量为  $n_{embeddings}$ , 质心数量为  $\sqrt{n_{embeddings}}$  效果良好
2. 实现细节：聚类对象不必是全体嵌入
  - 抽样：随机抽取一部分Token,  $(\text{抽样大小} \propto \sqrt{\text{语料库总大小}})$
  - 嵌入：先对抽样部分用BERT完成嵌入
  - 聚类：对抽样的嵌入执行  $\sqrt{n_{embeddings}}$ -Means, 所得质心近似视作整体嵌入的质心

#### 2 第二步：段落编码

1. 嵌入：用BERT计算所有文章的全精度嵌入，**不存储**
2. 压缩：用所得到的近似质心对全精度嵌入执行残差压缩，**存储**压缩后的向量

#### 3 第三步：倒排索引

1. 含义：构建质心→嵌入的嵌入ID列表，存储到磁盘中

```
1 | 质心a -> 属于质心a的嵌入ID=a1, a2, a3, ...
2 | 质心b -> 属于质心b的嵌入ID=b1, b2, b3, ...
3 | 质心c -> 属于质心c的嵌入ID=c1, c2, c3, ...
4 | .....
```

2. 意义：在后续检索时，可[先找到离查询最近的质心→直接从质心的列表中快速找到相关嵌入]

### 2.3.2.2. 文章以外的探讨：为何质心选择时要用取样近似

#### 1 假象方案的描述

方案	质心选择	段落编码
本文	计算 <b>并存储</b> 部分嵌入 $\xrightarrow{\text{得到}}$ 近似质心	再算( <b>但不存储</b> )完整嵌入 → 计算并存储压缩向量
假象	计算 <b>并存储</b> 完整嵌入 $\xrightarrow{\text{得到}}$ 精确质心	已有完整嵌入 → 计算并存储压缩向量

#### 2 假象方案的分析

1. 好处：计算量更少(免去部分计算的计算)+压缩所用质心更精确
2. 弊端：
  - 存储方面：计算质心时所用嵌入必须被存储，故假象方法无法避免对全体嵌入的存储
  - 并行方面：[计算完整嵌入↔压缩]被质心计算过程分割，无法像本文方案一样批间并行

### 2.3.3. 残差压缩后的在线检索实现

### 2.3.3.1. ColBERTv2的在线检索流程

1 查询输入：原始查询  $Q \xrightarrow[\text{预处理(嵌入)}]{\text{BERT}}$  多向量表示  $\{Q_1, Q_2, \dots, Q_N\}$

2 候选生成：解压所有嵌入不现实，所以需(借助倒排索引)缩小解压的范围

1. 质心查找：对于每个  $Q_i \xrightarrow{\text{质心查找}}$  离  $Q_i$  最近的  $n_{\text{probe}} \geq 1$  个质心

2. 倒排查找：对于每个  $Q_i \xrightarrow{\text{质心查找}}$  离  $Q_i$  最近的  $n_{\text{probe}} \geq 1$  个质心  $\xrightarrow{\text{嵌入查找}}$  每个质心的所有嵌入

3. 候选还原：将所得嵌入采用(索引质心+残差)解压还原，生成候选集

3 相似度计算

1. 嵌入得分：计算  $Q_i \xleftarrow{\text{余弦相似度}}$  候选集中每个解压后的嵌入，当作每个嵌入的得分

2. MaxSim下界：

◦ 嵌入分组：将所有嵌入+得分，按照嵌入所属的文档/段落分组

◦ 最大规约：记录每组中的最大得分，当作相应段落的MaxSim的近似下界

3. 段落得分：

◦ 循环：对查询的每个  $Q_i$  执行以上操作以计算所有MaxSim近似下界

◦ 加合：将每个段落的所有MaxSim近似下界相加，即为该段落得分的近似下界

4 文档排序

1. 初排：根据每个文档得分的近似下界排序，选取前面若干个段落

2. 重排：加载选定文档的完整嵌入 → 按ColBERTv1模式计算精确相似度 → 排序返回最终结果

### 2.3.3.2. 示例&为何是下界

1 基本情况：

1 对唯一查询：嵌入为  $\{q_1, q_2, q_3\}$

2 对三个文档：嵌入为  $\{d_1, d_2, d_3, d_4\} \{p_1, p_2, p_3, p_4\} \{o_1, o_2, o_3, o_4\}$

2 ColBERTv1的非重排做法

1. 分数计算：

- 1 计算距离  $q_1 \leftrightarrow \{d_1, d_2, d_3, d_4\}$ ，得到  $\text{MaxSim-d}_1$
- 2 计算距离  $q_2 \leftrightarrow \{d_1, d_2, d_3, d_4\}$ ，得到  $\text{MaxSim-d}_2$
- 3 计算距离  $q_3 \leftrightarrow \{d_1, d_2, d_3, d_4\}$ ，得到  $\text{MaxSim-d}_3$
- 4 最后  $q-d$  的相似得分为  $\text{score-d} = \text{MaxSim-d}_1 + \text{MaxSim-d}_2 + \text{MaxSim-d}_3$

- 1 计算距离  $q_1 \leftrightarrow \{p_1, p_2, p_3, p_4\}$ ，得到  $\text{MaxSim-p}_1$
- 2 计算距离  $q_2 \leftrightarrow \{p_1, p_2, p_3, p_4\}$ ，得到  $\text{MaxSim-p}_2$
- 3 计算距离  $q_3 \leftrightarrow \{p_1, p_2, p_3, p_4\}$ ，得到  $\text{MaxSim-p}_3$
- 4 最后  $q-p$  的相似得分为  $\text{score-p} = \text{MaxSim-p}_1 + \text{MaxSim-p}_2 + \text{MaxSim-p}_3$

- 1 计算距离  $q_1 \leftrightarrow \{o_1, o_2, o_3, o_4\}$ ，得到  $\text{MaxSim-o}_1$
- 2 计算距离  $q_2 \leftrightarrow \{o_1, o_2, o_3, o_4\}$ ，得到  $\text{MaxSim-o}_2$
- 3 计算距离  $q_3 \leftrightarrow \{o_1, o_2, o_3, o_4\}$ ，得到  $\text{MaxSim-o}_3$
- 4 最后  $q-o$  的相似得分为  $\text{score-o} = \text{MaxSim-o}_1 + \text{MaxSim-o}_2 + \text{MaxSim-o}_3$

2. 文档排序：对  $\text{score-d/p/o}$  排序，得分最高者为最相似文档

2 ColBERTv2的重排做法

### 1. 嵌入候选:

- 初选结果: 假设是  $\{d_2, d_3, d_4, p_1, p_2, o_1\}$
- 有关背景: 候选后的嵌入集可以只包含某个文档的部分嵌入, 为后续为什么是下界的基础

### 2. 文档初排:

- MaxSim下界: 嵌入分组+最大规约

- 1 文档d对应的组:
- 2 距离 $q_1 \leftrightarrow \{d_2, d_3, d_4\}$ , 取距离最大值作为近似MaxSim, 其一定小于实际值MaxSim-d1
- 3 距离 $q_2 \leftrightarrow \{d_2, d_3, d_4\}$ , 取距离最大值作为近似MaxSim, 其一定小于实际值MaxSim-d2
- 4 距离 $q_3 \leftrightarrow \{d_2, d_3, d_4\}$ , 取距离最大值作为近似MaxSim, 其一定小于实际值MaxSim-d3
- 5 将3个近似的MaxSim相加作为文档d的得分, 也一定小于实际值

- 1 文档p对应的组
- 2 距离 $q_1 \leftrightarrow \{p_1, p_2\}$ , 取距离最大值作为近似MaxSim, 其一定小于实际值MaxSim-q1
- 3 距离 $q_2 \leftrightarrow \{p_1, p_2\}$ , 取距离最大值作为近似MaxSim, 其一定小于实际值MaxSim-q2
- 4 距离 $q_3 \leftrightarrow \{p_1, p_2\}$ , 取距离最大值作为近似MaxSim, 其一定小于实际值MaxSim-q3
- 5 将3个近似的MaxSim相加作为文档p的得分, 也一定小于实际值

- 1 文档o对应的组
- 2 距离 $q_1 \leftrightarrow \{o_1\}$ , 取距离最大值作为近似MaxSim, 其一定小于实际值MaxSim-o1
- 3 距离 $q_2 \leftrightarrow \{o_1\}$ , 取距离最大值作为近似MaxSim, 其一定小于实际值MaxSim-o2
- 4 距离 $q_3 \leftrightarrow \{o_1\}$ , 取距离最大值作为近似MaxSim, 其一定小于实际值MaxSim-o3
- 5 将3个近似的MaxSim相加作为文档p的得分, 也一定小于实际值

- 文档初排: 对 d/p/o 三者的近似MaxSim排序, 即得到初排结果(此处假设为  $d > p > o$ )
- 结果截选: 选取初排结果的前若干个进入重排, 例如此处为两个(即 d/p)

### 3. 文档重排:

- MaxSim精确值: 解压得到 d/p 的全精度嵌入  $\{d_1, d_2, d_3, d_4\} \{p_1, p_2, p_3, p_4\}$

- 1 计算距离 $q_1 \leftrightarrow \{d_1, d_2, d_3, d_4\}$ , 得到MaxSim-d1
- 2 计算距离 $q_2 \leftrightarrow \{d_1, d_2, d_3, d_4\}$ , 得到MaxSim-d2
- 3 计算距离 $q_3 \leftrightarrow \{d_1, d_2, d_3, d_4\}$ , 得到MaxSim-d3
- 4 最后 $q-d$ 的相似得分为MaxSim-d1+MaxSim-d2+MaxSim-d3

- 1 计算距离 $q_1 \leftrightarrow \{p_1, p_2, p_3, p_4\}$ , 得到MaxSim-p1
- 2 计算距离 $q_2 \leftrightarrow \{p_1, p_2, p_3, p_4\}$ , 得到MaxSim-p2
- 3 计算距离 $q_3 \leftrightarrow \{p_1, p_2, p_3, p_4\}$ , 得到MaxSim-p3
- 4 最后 $q-p$ 的相似得分为MaxSim-p1+MaxSim-p2+MaxSim-p3

- 文档重排: 再将 d/p 的精确相似度排名, 例如如果  $p > d$  则 p 为最相似文档

## 3. LoTTE: 长尾域外评估

## 3.1. LoTTE是什么

1 LoTTE的目标：用于对长尾主题进行分层评估的数据集，强调用户自然查询

概念	释义
长尾主题	某个领域内数量庞大但极为低频(非主流)的话题，类似于Zipf定律
分层评估	将LoTTE数据集按主题显式划分，针对每个主题(每层)单独评估IR系统的性能
自然查询	模拟用户在真实情况下的提问

2 LoTTE原理：为何能针对长尾主题进行评估

1. 一般数据集(比如BEIR)：不会(难以)收录某一领域的长尾主题内容
2. LoRRE数据集：收录内容大多是不同领域的长尾主题，于BEIR结合评估可覆盖更多长尾话题

## 3.2. LoTTE的数据

1 数据组成

1. 测试集：

- 组成部分：包含12个主题不同互不重叠的测试集(每个包含500-2000个查询+10-200w个段落)
- 汇总操作：将所有主题的测试集数据合并，以测试模型在跨领域语料库上的域外检索性能

2. 验证集：

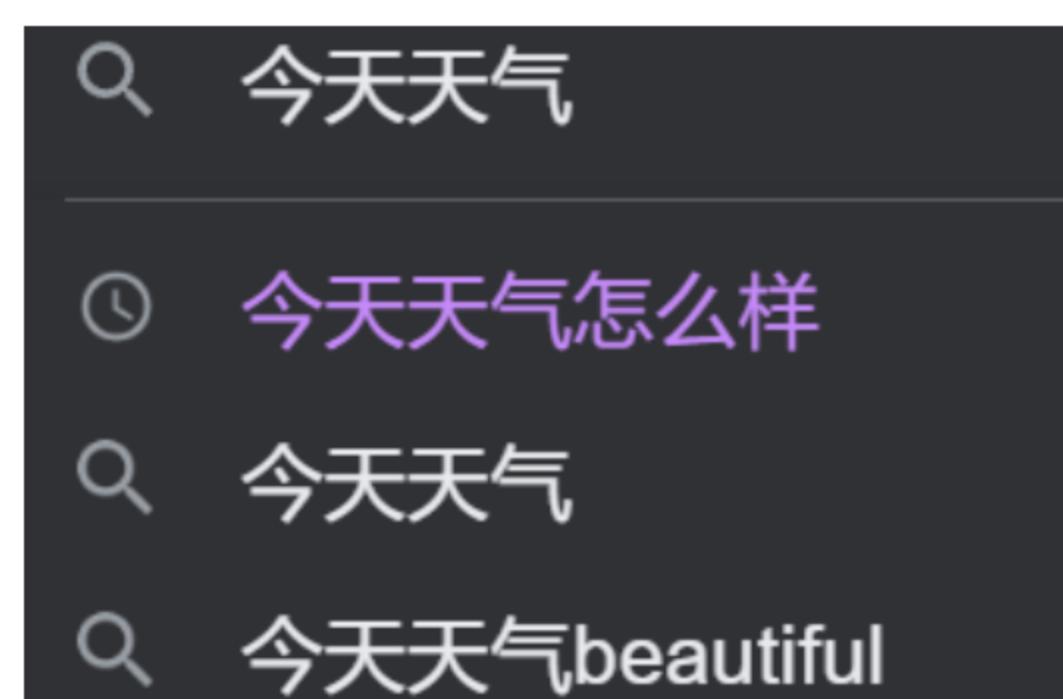
- 组成：每个测试集都有一验证集，其查询/段落与测试集的查询/段落相关但不重叠
- 作用：验证IR系统的域外迁移能力

2 语料来源

1. StackExchange：围绕特定主题的问答社区集合，类似于早期还没那么抽象的知乎

2. GooAQ：由最新的[Google自动完成<sup>对应</sup>→答案框]对，构成的[查询<sup>对应</sup>←结果]对数据库

- 自动完成：根据历史/语义相关性/热门搜索，生成的对用户输入查询的自动补全



1 用户输入查询：今天天气

2 候选自动完成：今天天气怎么样，今天天气温度，今天天气好不好，....

- 答案框：从搜索结果的网页中，提取的简短答案摘要



1 用户输入查询：今天天气

2 选定自动完成：今天天气怎么样

3 检索得答案框：新加坡天气，目前在新加坡，15:43-9十二月，周一，+25°C.....

### 3 语料构建

#### 1. LoTTE段落语料:

- 获取方法: 直接从StackE.抽取答案帖子, 并去处所有超链接/HTML标签/QA组件等
- 内容分布: 涵盖并分为写作/娱乐/科学/技术/生活五个主题, 并且不同话题占比均匀

#### 2. LoTTE查询语料:

- 两种查询的内容:

查询类型	筛选法则
搜索查询	若一GooAQ查询能搜到某StackE.答案贴, 则将此[查询↔答案]加入语料库
论坛查询	收集StackE.中标题, 并与其对应所有答案贴组成[答案↔答案]加入语料库

- 两种查询的特点:

查询类型	查询长短	回答模式	是否超出维基百科知识库
搜索查询	较短	简单(快速直白的信息呈现)	是(即大概率是长尾主题)
论坛查询	较长	复杂(往往带有讨论性质)	是(即大概率是长尾主题)

## 3.3. LoTTE的评估

1 评估集: 一个查询+StackE.目标答案帖子集(查询的标签)

2 评估指标: Success@5, 即查询返回的前5个结果中有位于目标集的帖子→系统得1分

## 4. 模型评估

### 4.1. 性能评估

#### 4.1.1. 域内评估

##### 1 官方评估

###### 1. 基本情况:

- 数据集: 官方预定义好的MS MARCO Passage Ranking上, 与训练集高度相似
- 评估对象: ColBERT/ColBERTv2/单向量系统

###### 2. 评估结果:

结果	解释
ColBERT>普通单向量模型	体现了细颗粒度后期交互的价值
ColBERT<高度调优的单向量模型	改进监督能让单向量模型性能大幅提升
ColBERTv2>高度调优的单向量模型	ColBERT架构也能从改进监督中大幅提升性能

##### 2 本地评估: 对模型泛化能力的初步评估

###### 1. 基本情况:

- 数据集: 从MS MARCO Passage Ranking中抽取的若干查询集, 与训练集严格不重叠
- 评估对象: ColBERTv2/SPLADEv2/RocketQAv2

###### 2. 评估结果: ColBERTv2远超Baseline

## 4.1.2. 域外评估

数据集	基本情况
Wikipedia开放问答	信息检索数据库，旨在整个开放域(Wiki总库)找到与查询相关的段落
BERI基准测试	广泛用于域外评估，包含13个多样化任务(如自然查询/语义相似性检索)
LoTTE基准	本文引入的新库，旨在更关注自然搜索查询&长尾主题

### 1 在BEIR上的评估

#### 1. 评估的模型：

- 无蒸馏：ColBERT/DPR-MARCO/ANCE/MoDIR
- 有蒸馏：TAS-B/SPLADEv2/RocketQAv2

#### 2. 评估任务：搜索(自然查询)/语义相关性

#### 3. 评估结果：

- 无蒸馏组：原始ColBERT的表现优于单向量系统(DPR/ANCE/MoDIR)
- 有蒸馏组：蒸馏后的模型都更强了，ColBERTv2/SPLADEv2强于其它且各有千秋

模型	优势
ColBERTv2	在自然查询有关的基准上表现更佳
SPLADEv2	在语义相关性更复杂/查询偏离自然语言的数据集上表现更优

### 2 Wikipedia开放问答任务

#### 1. 语料库概况：

- 查询来源：BM25(维基百科中的短文本)/TriviaQA/SQuAD
- 段落来源：维基百科所有内容聚合的总库

#### 2. 评估模型：ColBERTv2/ColBERT/SPLADEv2/BM25

#### 3. 评估结果：以Success@5为指标，ColBERTv2优于所有模型，SPLADEv略逊之

### 3 LoTTE评估结果

#### 1. 传统模型：BM25特别烂，ANCE/ColBERT显著好很多

#### 2. 蒸馏模型：SPLADEv2/RocketQAv2/ColBERTv2整体都很强，但ColBERTv2略微胜出

## 4.2. 其它实验结果

### 1 训练成本：有一定程度增加

### 2 残差压缩的效果：在MS MARCO上的编码索引大小

1. ColBERT要154GB，ColBERTv2在 $b=1/b=2$ 时只需16/25GB(缩小6-10倍)
2. 甚至只与采取激进压缩策略的单向量模型齐平

### 3 其它&不足

1. 仅在英文基准上评估
2. 评估的数据集上假阴性标注普遍存在
3. 难以与经过了高度调优的基线一致性对比