[原文章]

1. 导论

1.1. ColBERTv2

1 残差压缩的原理

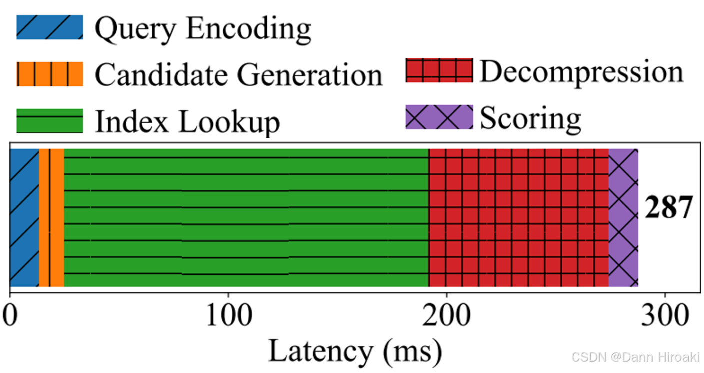
1. 聚类：对所有段落的全部Token向量的集合$\{p_j\}$进行聚类，为每个段落Token向量p_j分配了一个质心C_{p_j}
2. 残差：就是段落Token向量p_j与其质心C_{p_j}的距离$r=p_j-C_{p_j}$
3. 压缩：将浮点数压缩为二进制表示
    - 1-bit压缩：完整的残差r表示$\to$每维用2个离散状态近似表示$\to$每维用1bit二进制表示
    - 2-bit压缩：完整的残差r表示$\to$每维用4个离散状态近似表示$\to$每维用2bit二进制表示
4. 编码：将每个段落Token向量p_j表示为质心$C_{p_j}+$压缩的近似残差，通过反向解压缩残差即可得到近似的p_j

2 离线索引的流程(PLAID完整保留这一过程)

1. 采样嵌入：避免直接对全部$n_{\text{embeddings}}$规模的段落进行嵌入，而是抽取$k_1\sqrt{n_{\text{embeddings}}}$规模的段落进行部分嵌入
2. 近似质心：再对部分的嵌入向量执行$k_2\sqrt{n_{\text{embeddings}}}$-Means得到$k_2\sqrt{n_{\text{embeddings}}}$个质心，并且$k_2<k_1$
3. 压缩编码：对所有段落Token向量，按照分Chunk的形式对每个段落Token向量p_j执行以下操作
    - 用BERT编码器得到p_j的全精度向量，此时不对全精度向量进行任何存储
    - 找到与p_j最近的质心，由此得到p_j的残差压缩表示，此时才对残差压缩向量进行存储
4. 倒排索引：构建质心$\to$质心所包含的嵌入的ID的列表，存储到磁盘中

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

3 在线查询流程(PLAID将改进这一过程)



1. 查询嵌入：原始查询$Q\xrightarrow[\text{预处理(嵌入)}]{\text{BERT}}$段落的Token级多向量表示$\{q_1,q_2,...,q_n\}$
2. 候选生成：找到每个q_i最近的$n_{\text{probe}}\geq1$个质心，基于倒排索引，收集每个质心下属的所有段落Token向量的ID
3. 索引查找：收集候选ID集中所有对应的段落Token向量(候选向量集$\{p_j\}$)，传输其残差表示进内存
4. 残差解压：对所有候选向量执行残差解压，得到其近似的段落Token嵌入表示
5. 相似计算：基于近似的段落Token嵌入表示，与$\{q_1,q_2,...,q_n\}$进行MaxSim交互，得到近似距离(距离下界)

```
👉举个简单例子
  Q: {q1, q2, q3}
  P: {p1, p2, p3, p4} → {pj}集合中仅有{p1, p2, p3}
👉完整的距离计算
  Maxsim-1-full = Max{<q1,p1>,<q1,p2>,<q1,p3>,<q1,p4>}
  Maxsim-2-full = Max{<q2,p1>,<q2,p2>,<q2,p3>,<q2,p4>}
  Maxsim-3-full = Max{<q3,p1>,<q3,p2>,<q3,p3>,<q3,p4>}
👉近似的距离计算
  Maxsim-1-part = Max{<q1,p1>,<q1,p2>,<q1,p3>} ≤ Maxsim-1-full
  Maxsim-2-part = Max{<q2,p1>,<q2,p2>,<q2,p3>} ≤ Maxsim-2-full
  Maxsim-3-part = Max{<q3,p1>,<q3,p2>,<q3,p3>} ≤ Maxsim-3-full
👉所以一定是下界
```

6. 段落重排：保留初排的前若干段落并解压其所有Token向量，对若干段落执行完整的MaxSim与加和，得到最相似段落

1.2. PLAID

1 对ColBERTv2的Review

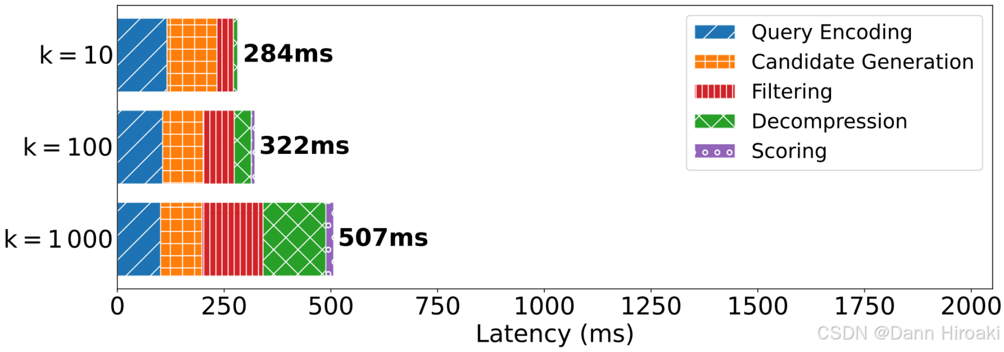
1. 仅质心就能识别高相关段落：用以下两种方式对段落进行粗筛，效果几乎没差别

模型	粗筛操作	段落评分
ColBERTv2	质心集$\{c_k\}$→倒排回所属向量$\{p_j\}$集→与$\{p_j\}$中向量有关的段落	查询与段落解压后向量的交互
PLAID	质心集$\{c_k\}$→与$\{c_k\}$中向量有关的段落	查询与段落向量中质心的交互

2. 很多质心是非常鸡肋的：并非所有质心都对段落评分有很大的贡献，对评分贡献超过0.2的质心只有0.5%左右

```
👉举个简单例子
  Q: {q1, q2, q3}
  P: {p1, p2, p3, p4} → 基于质心-段落的倒排索引, 可表示为{c1, c2, c3}
👉完整的质心交互
  Maxsim-1-full = Max{<q1,c1>,<q1,c2>,<q1,c3>}
  Maxsim-2-full = Max{<q2,c1>,<q2,c2>,<q2,c3>}
  Maxsim-3-full = Max{<q3,c1>,<q3,c2>,<q3,c3>}
👉假设c1对评分贡献远大于其它
  Maxsim-1-full = Max{<q1,c1>,<q1,c2>,<q1,c3>} = <q1,c1>
  Maxsim-2-full = Max{<q2,c1>,<q2,c2>,<q2,c3>} = <q2,c1>
  Maxsim-3-full = Max{<q3,c1>,<q3,c2>,<q3,c3>} = <q3,c1>
👉相当于可不可以直接剪掉c2,c3即P:{c1}
  Maxsim-1-part = Max{<q1,c1>} = <q1,c1>
  Maxsim-2-part = Max{<q2,c1>} = <q2,c1>
  Maxsim-3-part = Max{<q3,c1>} = <q3,c1>
👉很大程度上并不影响评分
```

2 PLAID的在线查询



1. 查询嵌入：将查询文本进行BERT编码，生成Token级的多向量$Q=\{q_1,q_2,...\}$(查询矩阵)
2. 候选生成：先对查询嵌入得到查询矩阵$Q=\{q_1,q_2,...\}$，与质心列表矩阵$C=\{c_1,c_2,...\}$两矩阵的交互
   - 查询输入：查询矩阵Q(所有Token的嵌入向量)+质心列表矩阵C(所有质心的向量)

```
👉举个简单例子
  Q: {q1, q2, q3, q4}
  C: {c1, c2, c3, c4, c5}
```

   - 得分计算：直接计算$S_{c,q}=CQ^T$，其中$S_{c,q}[i][j]$是质心c_i与查询词元q_j的相关性得分

```
👉构成的S_cq矩阵
  <c1,q1>  <c1,q2>  <c1,q3>  <c1,q4>
  <c2,q1>  <c2,q2>  <c2,q3>  <c2,q4>
  <c3,q1>  <c3,q2>  <c3,q3>  <c3,q4>
  <c4,q1>  <c4,q2>  <c4,q3>  <c4,q4>
  <c5,q1>  <c5,q2>  <c5,q3>  <c5,q4>
```

   - 质心排序：对每个q_j选取其排名从高到低前t_{nprobe}个质心

```
👉选定S_cq矩阵每列的Top-t(此处以Top-2)为例
  <c1,q1>  *<c1,q2>* *<c1,q3>*  <c1,q4>
 *<c2,q1>* *<c2,q2>*  <c2,q3>  *<c2,q4>*
  <c3,q1>   <c3,q2>   <c3,q3>   <c3,q4>
 *<c4,q1>*  <c4,q2>  *<c4,q3>*  <c4,q4>
  <c5,q1>   <c5,q2>   <c5,q3>  *<c5,q4>*
👉选取每个q的Top-2质心
  q1 -> c2, c4
  q2 -> c1, c2
  q3 -> c1, c4
  q4 -> c2, c5
```

- 质心候选：合并每个q的候选质心，得到最终质心候选集C'

```
👉合并所有q的Top-2质心得到候选集
C' = {c1, c2, c4, c5}
```

- 段落候选：若一个段落中存在q属于C'，则将该段落候选之，并保存质心$\rightarrow$(唯一)段落ID的倒排索引

```
👉每个质心只能有一个相关段落,例如
c1->P1, c2->P1, c4->P2, c5->P3
👉建立段落到质心的索引
P1 -> {c1, c2}
P2 -> {c4}
P3 -> {c5}
```

3. 候选过滤：对于所有的候选段落$\{P_1, P_2, ...\}$，按照如下方式处理以进行初排
   - 质心索引：其实就是合并每个P_i所对应的质心的ID

```
👉合并每个段落的质心ID
I = {c1, c2, c4, c5}
```

   - 质心得分：无需重复计算，所有质心和每个q_i的相似度，已经包含在预先计算的$S_{c,q}=CQ^T$中了

```
👉从S_cq矩阵中抽取有用的行(质心)
 ✅c1  <c1,q1>  <c1,q2>  <c1,q3>  <c1,q4>
 ✅c2  <c2,q1>  <c2,q2>  <c2,q3>  <c2,q4>
 ❌c3  <c3,q1>  <c3,q2>  <c3,q3>  <c3,q4>
 ✅c4  <c4,q1>  <c4,q2>  <c4,q3>  <c4,q4>
 ✅c5  <c5,q1>  <c5,q2>  <c5,q3>  <c5,q4>
```

   - 质心剪枝：检查每个质心的最大得分，对于最大得分都小于阈值t_{cs}的直接丢弃

```
👉对S_cq矩阵中的剩余行(质心)进一步筛选(剪枝)
  ✅c1   <c1,q1>  <c1,q2>  <c1,q3>  <c1,q4> -> 四个至少有一个大于t_cs
  ✅c2   <c2,q1>  <c2,q2>  <c2,q3>  <c2,q4> -> 四个至少有一个大于t_cs
  ❌c3   <c3,q1>  <c3,q2>  <c3,q3>  <c3,q4>
  ✅c4   <c4,q1>  <c4,q2>  <c4,q3>  <c4,q4> -> 四个至少有一个大于t_cs
  ❌c5   <c5,q1>  <c5,q2>  <c5,q3>  <c5,q4> -> 全部小于t_cs故剪枝掉
👉别忘了更新列表I
  I = {c1, c2, c4}
```

- 质心交互：相当于就是用段落的质心表示段落如$P_1 \approx \{c_1, c_2\}$，然后同样的质心MaxSim

```
👉构建矩阵P_hat
  P_hat[1] = S_cq[I[1]=c1] = <c1,q1> <c1,q2> <c1,q3> <c1,q4>
  P_hat[2] = S_cq[I[2]=c2] = <c2,q1> <c2,q2> <c2,q3> <c2,q4>
  P_hat[3] = S_cq[I[3]=c4] = <c4,q1> <c4,q2> <c4,q3> <c4,q4>
👉以计算Q和P1相似度为例, 提取P_hat[1]行和P_hat[2]行
  Maxsim-1 = Max{<q1,c1>,<q1,c2>}
  Maxsim-2 = Max{<q2,c1>,<q2,c2>}
  Maxsim-3 = Max{<q3,c1>,<q3,c2>}
  Maxsim-4 = Max{<q4,c1>,<q4,c2>}
👉计算Q与其它段落的相似度, 只需提取P_hat中其它行就行了
```

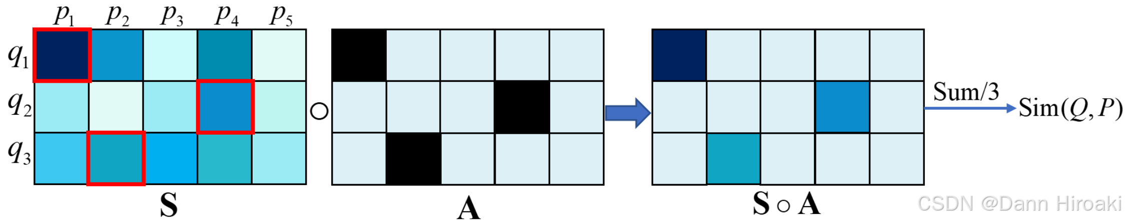
- 段落粗筛：根据质心交互的结果，选取前Top-nDos的段落以进入下一个阶段，设定nDocs的超参数
- 再次粗筛：将Top-nDos段落作为新候选集，再不质心剪枝地质心一次交互，由新的交互结果选出Top-$\frac{\text{nDos}}{4}$段落

4. 残差解压：解压Top-$\frac{\text{nDos}}{4}$段落中(除质心外)的所有向量，注意质心是全精度的所以要除质心外
5. 段落重排：基于Top-$\frac{\text{nDos}}{4}$段落的全精度表示，与查询向量集进行完整的MaxSim操作以得到精确评分

1.3. XTR

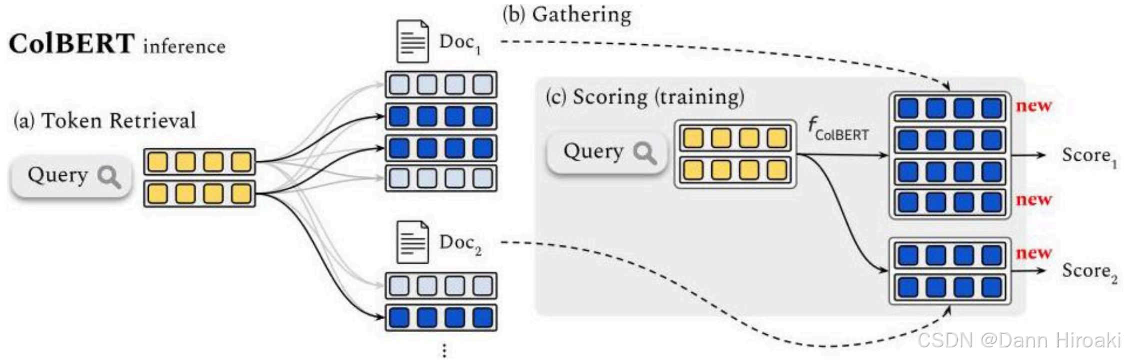
1 从ColBERT到XTR

1. 评分函数：$S_{p,q} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}(\mathbf{P} \circ \mathbf{A})_{ij}$，即调整对齐矩阵$\mathbf{A}$以择评分矩阵$\mathbf{S}$每行最大值，最后除以$Z$归一化



- 评分矩阵$\mathbf{S}$：令查询$Q = \{q_1, q_2, \ldots, q_n\}$以及文档$P = \{p_1, p_2, \ldots, p_m\}$，记子内积为$s_{ij} = q_i^\top p_j$，由此构成$\mathbf{S} \in \mathbb{R}^{n \times m}$
- 对齐矩阵$\mathbf{A}$：让每个元素$a_{ij} \in \{0, 1\}$来对$\mathbf{P}$中的元素进行不同强度的选择，由此构成$\mathbf{A} \in \mathbb{R}^{n \times m}$

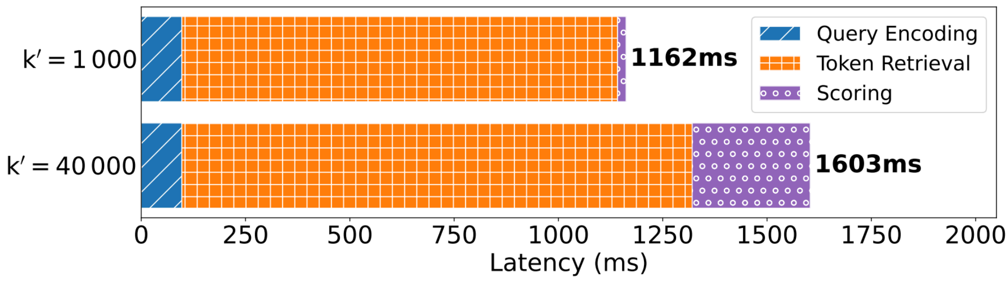
2. 传统ColBERT的流程：



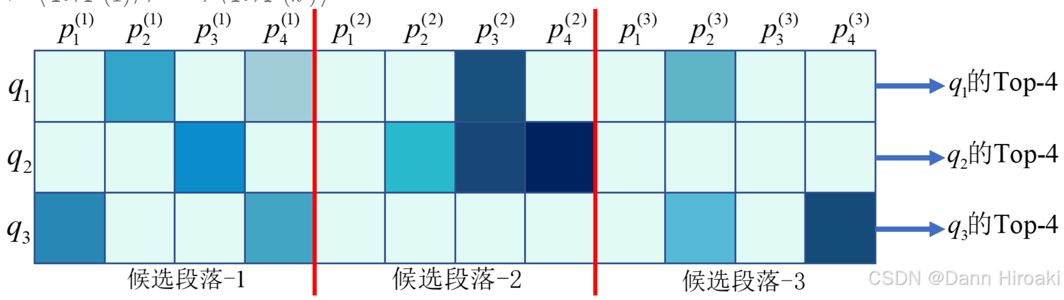
- Token检索：用查询单向量集中每个q_i检索k'个段落Token，最多产生$n\times k'$个候选Token
- 收集向量：加载$n\times k'$个候选Token所属的段落，收集这些段落中所有的Token向量
- 评分与重排：对这些段落应用全精度的ColBERT非线性相似度以进行重排

3. XTR的改良动机：
   - 训练上：传统的训练旨在优化最终ColBERT评分，而推理过程旨在获得Top-k的Token，故XTR重构了目标函数
   - 开销上：收集Top-k候选段落的所有Token开销巨大，故省去收集步骤，只用检索到的段落Token来构成相似度

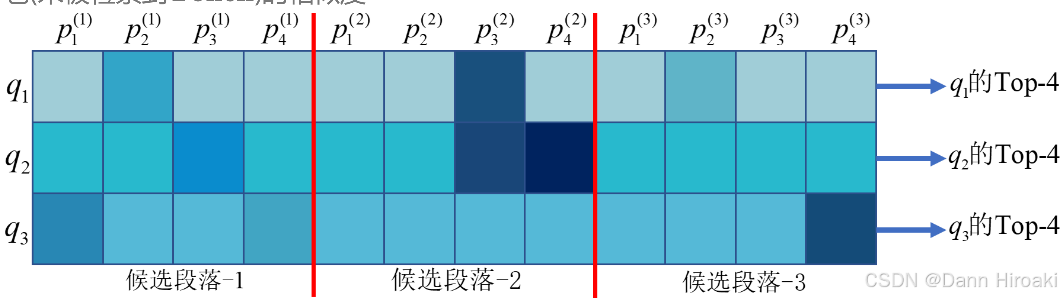
2 XTR在线检索的流程



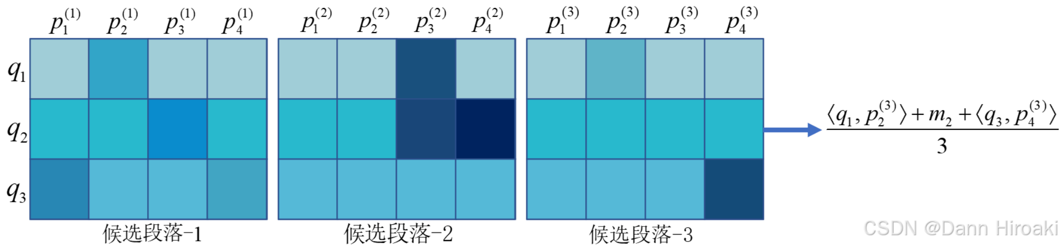
1. 查询嵌入：将查询文本进行BERT编码，生成Token级的多向量$Q=\{q_1,q_2,...,q_n\}$
2. 候选生成：对所有n个查询向量q_i执行Top-k'检索，回溯这nk'个Token所属的文档，确定C个候选文档
3. 段落评分：$S_{p,q}=\frac{1}{n}\sum_{i=1}^{n}\max_{1\leq j\leq m}\left[\mathbf{A}_{ij}\langle q_i,p_j\rangle+(1-\mathbf{A}_{ij})m_i\right]$，其中对齐矩阵 $\mathbf{A}_{ij}=\mathbb{1}_{\left[j\in\text{Top-}k'_j(p_{ij'})\right]}$
   - 排序：以检索得q_i的Top-k为$p_{(1)},p_{(2)},...,p_{(k')}$，假设这些Token与$q_i$的相似度从高到低为$\langle q_i,p_{(1)}\rangle,...,\langle q_i,p_{(k')}\rangle$



- 填充：令被检索到的Token中的相似度最低者为$m_i=\langle q_i,p_{(k')}\rangle$，直接用$m_i$去填充一切其它(未被检索到Token)的相似度



- 评分：填充后再取每个段落相似度矩阵每行相似度的最大值相加，然后除以行数归一化



2. WARP原理

2.1. WARP的整体流程

2.1.1. 离线索引

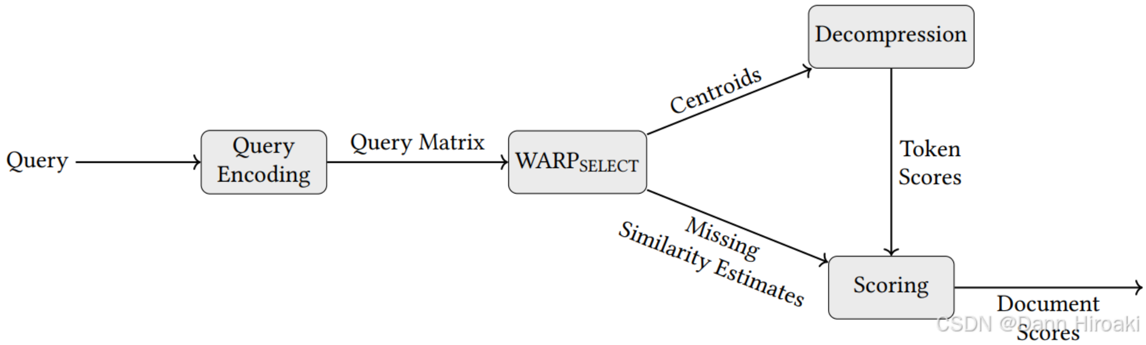
1 WARP的残差压缩原理：区别于ColBERTv2，此处基于分桶处理了残差的非均匀分布

1. 聚类：对所有段落的全部Token向量的集合$\{p_j\}$进行聚类，为每个段落Token向量p_j分配了一个质心C_{p_j}
2. 残差：就是段落Token向量p_j与其质心C_{p_j}的距离$r=p_j-C_{p_j}$
3. 分桶：对每维残差，将所有数据划分到2^b个非均匀连续桶中，保证每桶中样本数量差不多，每桶用一个b位二进制表示
4. 压缩：残差每维用其所属桶的b位二进制编码，当$b=4$时每Byte可编码两维的残差(相比浮点数压缩了八倍)
5. 编码：将每个段落Token向量p_j表示为质心C_{p_j}+压缩的近似残差，通过反向解压缩残差即可得到近似的p_j

2 与ColBERTv2采用一样的残差压缩过程

1. 采样嵌入：避免直接对全部$n_{\text{embeddings}}$规模的段落进行嵌入，而是抽取$k_1\sqrt{n_{\text{embeddings}}}$规模的段落进行部分嵌入
2. 近似质心：再对部分的嵌入向量执行$k_2\sqrt{n_{\text{embeddings}}}$-Means得到$k_2\sqrt{n_{\text{embeddings}}}$个质心，并且$k_2<k_1$
3. 压缩编码：对所有段落Token向量，按照分Chunk的形式对每个段落Token向量p_j执行以下操作
   - 用BERT编码器得到p_j的全精度向量，此时不对全精度向量进行任何存储
   - 找到与p_j最近的质心，由此得到p_j的残差压缩表示，此时才对残差压缩向量进行存储
4. 倒排索引：构建质心→质心所包含的嵌入的ID的列表，存储到磁盘中

2.1.2. 在线索引



1️⃣ 查询编码：将查询文本编码成查询矩阵$Q=\{q_1,q_2,...,q_n\}\in\mathbb{R}^{n\times d}$，获取质心列表$C=\{c_1,c_2,...\}$

```
👉举个简单例子
Q: {q1,q2,q3,q4}
C: {c1,c2,c3,c4,c5,c6}
```

2️⃣ 候选生成：为每个q_i确定最相似的$\text{Top-}n_{\text{probe}}$质心以供解压，为每个$q_i$算出一个相似度填充$m_i$以辅助评分

1. 得分计算：计算$S_{c,q}=CQ^T$，其中$S_{c,q}[ik][i]$是质心c_{ik}与查询词元q_i的相关性得分

   ```
   👉构成的S_cq矩阵
     <c1,q1>=0.8  <c1,q2>=0.9  <c1,q3>=0.8  <c1,q4>=0.7
     <c2,q1>=0.9  <c2,q2>=0.8  <c2,q3>=0.8  <c2,q4>=0.8
     <c3,q1>=0.5  <c3,q2>=0.7  <c3,q3>=0.6  <c3,q4>=0.9
     <c4,q1>=0.7  <c4,q2>=0.6  <c4,q3>=0.7  <c4,q4>=0.6
     <c5,q1>=0.6  <c5,q2>=0.5  <c5,q3>=0.5  <c5,q4>=0.5
     <c6,q1>=0.1  <c6,q2>=0.2  <c6,q3>=0.1  <c6,q4>=0.2
   ```

2. 质心排序：对每个q_i选取其排名从高到低前$\text{Top-}n_{\text{probe}}$个质心构成候选质心，$\text{Top-}n_{\text{probe}}$质心是下一步解压的基础

   ```
   👉以选取每个q的Top-3质心，获得候选列表{c1,c2,c3,c4}
     q1 -> c1,c2,c4
     q2 -> c1,c2,c3
     q3 -> c1,c2,c4
     q4 -> c1,c2,c3
   ```

3. 缺失估计：类似XTR对于每个q_i都要设置一个填充分数m_i，此处将m_i设为质心得分列表的第一个元素(见下例)

   ```
   👉假定每个质心的簇大小(与质心关联的p向量数)
     |c1|=100, |c2|=50, |c3|=30, |c4|=200, |c5|=80
   👉以q1为例，将其Top-3质心按照最大相似度排序
     q1 -> c2, c1, c4
   👉计算q1每个质心簇大小的累加值
     050 <- c2
     150 <- c2 + c1
   ```

```
350 <- c2 + c1 + c4
👉设定累积簇大小阈值t'=125,则选取第一个超过阈值的质心
050 <- c2
150 <- c2 + c1✅
350 <- c2 + c1 + c4
👉用m1=<q1,c1>去填充其余与q1有关的相似度,依此类推
m1用来填充其余与q1有关的相似度
m2用来填充其余与q2有关的相似度
m3用来填充其余与q3有关的相似度
m4用来填充其余与q4有关的相似度
```

3 残差解压：识别出候选质心所关联的所有向量集$\{p_j\}$，得到q_i与其每个p_j的评分(无需显式解压p_j/细节见后)

```
👉对每个质心倒排索引
{c1,c2,c3,c4} -> 候选向量集{P1-p1,P1-p2,P2-p1,P2-p3,P3-p1,P4-p1}
 c1 -> P1-p1, P2-p3
 c2 -> P1-p2
 c3 -> P4-p1
 c4 -> P2-p1, P3-p1
👉解压所有候选向量并计算出相似度
<P1-p1,q1>  <P1-p1,q2>  <P1-p1,q3>  <P1-p1,q4>
<P1-p2,q1>  <P1-p2,q2>  <P1-p2,q3>  <P1-p2,q4>
<P2-p1,q1>  <P2-p1,q2>  <P2-p1,q3>  <P2-p1,q4>
<P2-p3,q1>  <P2-p3,q2>  <P2-p3,q3>  <P2-p3,q4>
<P3-p2,q1>  <P3-p2,q2>  <P3-p2,q3>  <P3-p2,q4>
<P4-p1,q1>  <P4-p1,q2>  <P4-p1,q3>  <P4-p1,q4>
```

4 候选评分：残差解压阶段得到的向量级评分＋候选生成阶段得到的向量级缺失评分估计⟶最终段落级评分

1. Token级归约：先合并每个q_i下所有质心簇的所有向量(大簇)，让q_i在大簇中基于段落Group-by求MaxSim

```
👉合并每个qi下所有簇的所有段落Token向量
q1 -> c1, c2, c4 -> P1-p1, P1-p2, P2-p1, P2-p3, P3-p1
q2 -> c1, c2, c3 -> P1-p1, P1-p2, P2-p3, P4-p1
q3 -> c1, c2, c4 -> P1-p1, P1-p2, P2-p1, P2-p3, P3-p1
q4 -> c1, c2, c3 -> P1-p1, P1-p2, P2-p3, P4-p1
👉Token归约,以q1为例
MaxSim(q1,P1) = Max{<q1,P1-p1>,<q1,P1-p2>}
MaxSim(q1,P2) = Max{<q1,P2-p1>,<q1,P2-p3>}
MaxSim(q1,P3) = Max{<q1,P3-p1>}
MaxSim(q1,P4) = Max{}
👉Token归约,以q2为例
MaxSim(q2,P1) = Max{<q2,P1-p1>,<q2,P1-p2>}
MaxSim(q2,P2) = Max{<q2,P2-p3>}
MaxSim(q2,P3) = Max{}
MaxSim(q2,P4) = Max{<q2,P4-p1>}
👉显然这里存在缺失别如MaxSim(q1,P4),这将在后续段落归约中填充
```

- 由于以上操作的所有相似度评分都在残差解压阶段得到，所以可以直接在残差解压阶段隐式完成Token级归约
- 另外这只是一个逻辑上的描述，详细的实现见后

2. 段落级归约：用每个q_i的缺失估计进行填充，再合并每个段落的所有MaxSim，同样详细的实现见后

```
👉段落级归约，以q1为例先用m1进行缺失填充
  MaxSim(q1,P1) = Max{<q1,P1-p1>,<q1,P1-p2>}
  MaxSim(q1,P2) = Max{<q1,P2-p1>,<q1,P2-p3>}
  MaxSim(q1,P3) = Max{<q1,P3-p1>}
  MaxSim(q1,P4) = m1
👉段落级归约，以q2为例先用m2进行缺失填充
  MaxSim(q2,P1) = Max{<q2,P1-p1>,<q2,P1-p2>}
  MaxSim(q2,P2) = Max{<q2,P2-p3>}
  MaxSim(q2,P3) = m2
  MaxSim(q2,P4) = Max{<q2,P4-p1>}
👉合并每个段落的MaxSim得到最终相似度
  WARP(Q,P1) = MaxSim(q1,P1) + MaxSim(q2,P1) + MaxSim(q3,P1) + MaxSim(q4,P1)
  WARP(Q,P2) = MaxSim(q1,P2) + MaxSim(q2,P2) + MaxSim(q3,P2) + MaxSim(q4,P2)
  WARP(Q,P3) = MaxSim(q1,P3) + MaxSim(q2,P3) + MaxSim(q3,P3) + MaxSim(q4,P3)
  WARP(Q,P4) = MaxSim(q1,P4) + MaxSim(q2,P4) + MaxSim(q3,P4) + MaxSim(q4,P4)
```

2.2. WARP的优化细节

2.2.1. 残差解压的实现细节

1 一些基本设置

1. 解压输入：候选质心，即$Q=\{q_1,q_2,...,q_n\}\in\mathbb{R}^{n\times d}$中每个$q_i$排名从高到Top-$n_{\text{probe}}$低前个质心
2. 符号表示：让q_i为$Q=\{q_1,q_2,...,q_n\}\in\mathbb{R}^{n\times d}$中任意一个向量

符号	含义
c_{ik}	令q_i的Top-n_{probe}质心为$\{c_{i1},c_{i2},....,c_{in_{\text{probe}}}\}$，$c_{ik}$就是$q_i$的Top-$n_{\text{probe}}$质心之一
p_{ikj}	令c_{ik}簇中的向量为$\{p_{ik1},p_{ik2},p_{ik3},...\}$，$p_{ikj}$就是$c_{ik}$簇中的向量之一
r_{ikj}	质心c_{ik}与其下属向量p_{ikj}间存在一个残差，r_{ikj}就是这个残差的压缩编码
s_{ikj}	查询向量q_i和段落向量p_{ikj}的相似度，解压步得到的是每个q_i和其所有Top-n_{probe}质心簇下所有向量的相似度

3. 标量表示：当要从向量中提取标量时，遵循类C++风格，例如q_i中第α维的元素记作$q_i[\alpha]$

2 压缩编码的结构

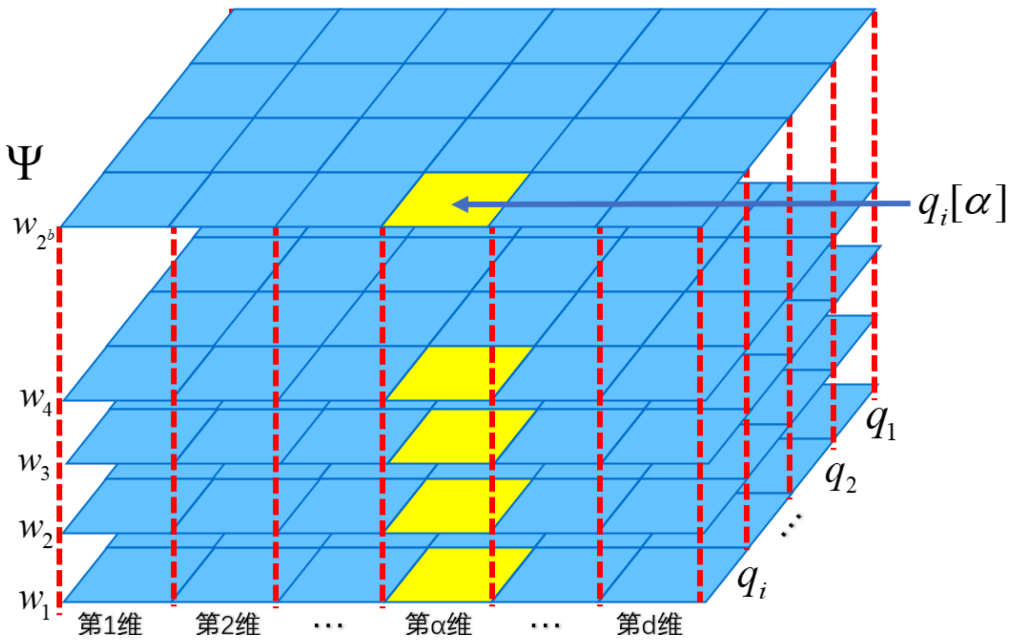
1. 索引结构：残差一共有d维，所以对应r_{ikj}中一共有d组桶索引，每个索引宽(桶编码宽)b-bit
2. 分桶结构：每个桶都给出一个残差值(用于加到某一维度上)，所有桶依次构成了残差值向量$\Psi=\{w_1,w_2,\ldots,w_{2^b}\}\in\mathbb{R}^{2^b}$
   - 压缩残差第α维编码是$r_{ikj}[\alpha]=00\ldots0\mathrm{B}$时，解压后残差第$\alpha$维加上残差值$\Psi[r_{ikj}[\alpha]]=\Psi[00\ldots0\mathrm{B}]=\Psi[0]=w_1$
   - 压缩残差第α维编码是$r_{ikj}[\alpha]=00\ldots1\mathrm{B}$时，解压后残差第$\alpha$维加上残差值$\Psi[r_{ikj}[\alpha]]=\Psi[00\ldots1\mathrm{B}]=\Psi[1]=w_2$
   - 压缩残差第α维编码是$r_{ikj}[\alpha]=11\ldots1\mathrm{B}$时，解压后残差第$\alpha$维加上残差值$\Psi[r_{ikj}[\alpha]]=\Psi[11\ldots1\mathrm{B}]=\Psi[2^b]=w_{2^b}$

3 残差解压及其评分

1. 解压：以c_{ik}为质心，对于第$\alpha=1,2,\ldots,d$维，其残差编码是$r_{ikj}[\alpha]$，需要为该维度加上残差值$\Psi[r_{ikj}[\alpha]]$
   - 向量化：让$e_\alpha=\{0_1,0_2,\ldots,1_\alpha,\ldots,0_d\}$后，就有
     $$\mathrm{Decompress}(c_{ik},p_{ikj},r_{ikj})=c_{ik}+\sum_{\alpha=1}^{d}\Psi[r_{ikj}[\alpha]]e_\alpha$$
   - 展开后：其中$\displaystyle\sum_{\alpha=1}^{d}\Psi[r_{ikj}[\alpha]]e_\alpha=\{\Psi[r_{ikj}[1]],\Psi[r_{ikj}[2]],\ldots,\Psi[r_{ikj}[d]]\}\in\mathbb{R}^d$
2. 评分：已有p_j的解压表示，将其与q_i内积即得
   $$s_{ikj}=\langle q_i,\mathrm{Decompress}(c_{ik},p_{ikj},r_{ikj})\rangle=\langle c_{ik},q_i\rangle+\sum_{\alpha=1}^{d}\Psi[r_{ikj}[\alpha]]q_i[\alpha]$$

4 评分加速计算

1. 对$S_{c,q}$的复用：$S_{c,q}$在进行Top-n_{probe}质心选择前就已经预计算好了，此处直接有$\langle c_{ik},q_i\rangle=S_{c,q}[ik][i]$
2. 对V的预计算：将$Q\in\mathbb{R}^{n\times d}$和$\Psi\in\mathbb{R}^{2^b}$扩展至$\hat{Q}\in\mathbb{R}^{n\times d\times1}$和$\hat{\Psi}\in\mathbb{R}^{1\times2^b}$，鉴于$\Psi$固定$\rightarrow$可解压前预计算$V=Q\times\Psi=\hat{Q}\times\hat{\Psi}$



   - 对于$\Psi=\{w_1,w_2,\ldots,w_{2^b}\}\in\mathbb{R}^{2^b}$，变换得$(\Psi\times q_i[\alpha])=\{w_1q_i[\alpha],w_2q_i[\alpha],\ldots,w_{2^b}q_i[\alpha]\}\in\mathbb{R}^{2^b}$

- 于是$(\Psi\times q_i[\alpha])[r_{ikj}[\alpha]]=\Psi[r_{ikj}[\alpha]]q_i[\alpha]$，并且注意到$(\Psi\times q_i[\alpha])$向量**就是**图中**黄标**部分即$V[i,\alpha]$
- 所以$\sum_{\alpha=1}^{d}\Psi[r_{ikj}[\alpha]]q_i[\alpha]=\sum_{\alpha=1}^{d}(\Psi\times q_i[\alpha])[r_{ikj}[\alpha]]=\sum_{\alpha=1}^{d}V[i,\alpha,r_{ikj}[\alpha]]$

3. 复杂度分析：$s_{ikj}=S_{c,q}[ik][i]+\sum_{\alpha=1}^{d}V[\alpha,i,r_{ikj}[\alpha]]$，当$S_{c,q}$和$V$都预计算好时，$s_{ikj}$可在$O(1)$时间内得到

2.2.2. 候选评分的实现细节

1 Token级归约的详细描述

1. 数据结构：构建小数据块S_{ik}
   - 定义：用来描述q_i下属的Top-n_{probe}质心c_{ik}下属的簇，共$n\times n_{\text{probe}}$组(每簇都对应一组)
   - 结构：为$S_{ik}=\{(\text{pid}_j,\text{sim}_j)\}$，其中$\text{pid}_j$指示$q_i$的质心$c_{ik}$下的向量$p_{ikj}$所属的段落(ID)，$\text{sim}_j=\langle q_i,p_{ikj}\rangle$(解压步已得)

```
S-ik -> {(qi的第k个质心簇下某个候选向量所属段落的ID, qi与这个候选向量的相似度)}
```

2. 部分函数：S_{ik}隐式地定义了$f_{S_{ik}}$
   - 给出某个pid_j，如果该pid_j存在于$S_{ik}=\{(\text{pid}_j,\text{sim}_j)\}$中，则给出与该$\text{pid}_j$对应的相似度$\text{sim}_j$
   - 给出某个pid_j，如果该pid_j存在于$S_{ik}=\{(\text{pid}_j,\text{sim}_j)\}$外，则给出空值$\perp$
3. Token归约函数：定义r_{token}为保留最大值的归约，以$\text{Reduce}\left(r_{\text{token}},S_{ik_1},S_{ik_2}\right)$对$S_{ik_1}/S_{ik_2}$归约为例

pid_j在S_{ik_1}中	pid_j在S_{ik_2}中	如何操作归约集$\text{Reduce}\left(r_{\text{token}},S_{ik_1},S_{ik_2}\right)$
✅ $(\text{pid}_j,\text{sim}_{j_1})$	✅ $(\text{pid}_j,\text{sim}_{j_2})$	将$(\text{pid}_j,\text{sim}_{j_1})$和$(\text{pid}_j,\text{sim}_{j_2})$归约成$\left(\text{pid}_j,\max\{\text{sim}_{j_1},\text{sim}_{j_2}\}\right)$后加进去
✅ $(\text{pid}_j,\text{sim}_{j_1})$	❌(NULL)	将$(\text{pid}_j,\text{sim}_{j_1})$直接加进去
❌(NULL)	✅ $(\text{pid}_j,\text{sim}_{j_2})$	将$(\text{pid}_j,\text{sim}_{j_2})$直接加进去
❌(NULL)	❌(NULL)	不进行操作

4. Token级归约：就是对每个q_i，在其所有质心(数据块)上以r_{token}方式进行归约
   - 定义：对q_i的Token归约，归约后的集合为$S_i=\text{Reduce}\left(r_{\text{token}},S_{ik_1},S_{ik_2},\ldots,S_{in_{\text{probe}}}\right)$，称之为**大数据块**
   - 含义：归约是基于最大值并且所有段落ID唯一，所以最终得到的归约集是**q_i和有关段落的MaxSim值**的集

```
S-i -> {(qi能关联到的段落的ID, qi与该段落的MaxSim)}
```

2 段落级归约的详细描述

1. 数据结构：输入每个q_i的Token级归约的结果(大数据块)，即$\{S_1, S_2, \ldots, S_n\}$
   - 结构构建：$S_{\beta_1\to\beta_2}$为按r_{pssge}对从S_{β_1}到S_{β_2}的大数据块进行归约，即$S_{\beta_1\to\beta_2}=\text{Reduce}\left(r_{\text{pssge}}, S_{\beta_1}, S_{\beta_1+1}, \ldots, S_{\beta_2}\right)$
   - 递归定义：考虑到r_{pssge}是线性的，还可以给出递归定义$S_{\beta_1\to\beta_2}=\text{Reduce}\left(r_{\text{pssge}}, S_{\beta_1\to\gamma}, S_{\gamma\to\beta_2}\right)$，$S_{\beta_1\to\beta_2}$值与$\gamma$无关
2. 段落归约函数：即用r_{pssge}对两个大数据块进行归约，以$S_{\beta_1\to\beta_2}=\text{Reduce}\left(r_{\text{pssge}}, S_{\beta_1\to\gamma}, S_{\gamma\to\beta_2}\right)$对$S_{\beta_1}\to S_{\beta_2}$归约为例

pid_j在S_{β_1}中	pid_j在S_{β_2}中	如何操作归约集$S_{\beta_1\to\beta_2}=\text{Reduce}\left(r_{\text{pssge}}, S_{\beta_1\to\gamma}, S_{\gamma\to\beta_2}\right)$
✅ $(\text{pid}_j, \text{sim}_{j_1})$	✅ $(\text{pid}_j, \text{sim}_{j_2})$	将$(\text{pid}_j, \text{sim}_{j_1})$和$(\text{pid}_j, \text{sim}_{j_2})$归约成$(\text{pid}_j, \text{sim}_{j_1}+\text{sim}_{j_2})$后再加进去
✅ $(\text{pid}_j, \text{sim}_{j_1})$	❌(NULL)	将$(\text{pid}_j, \text{sim}_{j_1})$填充为$\left(\text{pid}_j, \text{sim}_{j_1}+\sum\limits_{t=\gamma+1}^{\beta_2} m_t\right)$后再加进去
❌(NULL)	✅ $(\text{pid}_j, \text{sim}_{j_2})$	将$(\text{pid}_j, \text{sim}_{j_2})$填充为$\left(\text{pid}_j, \text{sim}_{j_2}+\sum\limits_{t=\beta_1}^{\gamma} m_t\right)$后再加进去
❌(NULL)	❌(NULL)	不进行操作

3. 段落级归约：以一个类二叉树的结构展开
   - 操作：不断递归合并不同的$S_{\beta_1\to\beta_2}$，最后覆盖所有q_i构成$S=S_{1\to n}$(最大数据块)
   - 含义：S本质就是一个包含了XTR填充机制的**最终评分**

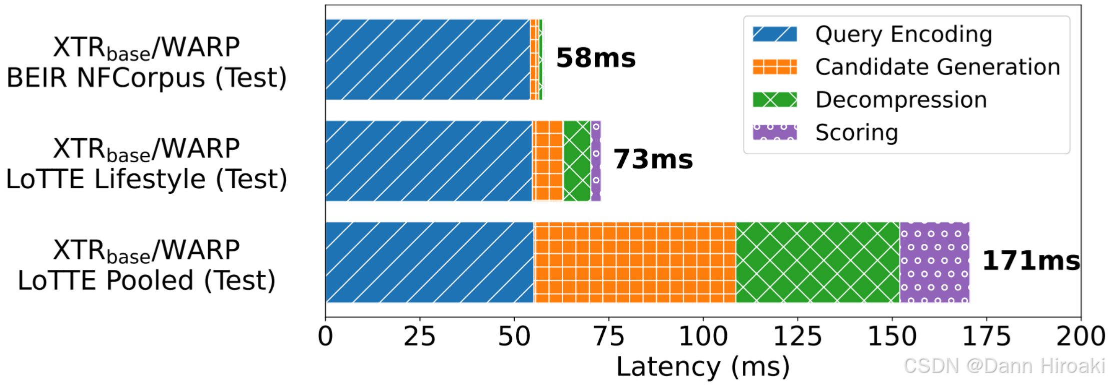
```
S -> {(段落的ID, 整个查询Q与该段落的相似度)}
```

3. 实验与结果

1 超参数

超参数	含义	对性能的影响
n_{probe}	每个q_i检索到的最邻近的质心数	当然是越大越好，但是到了$n_{\text{probe}}=32$后性能提升变慢，故就选$n_{\text{probe}}=32$
t'	候选生成时累积簇大小的阈值	当然也是越大越好(但存在提升上限)，经验上设为$k\times$数据集大小的平方根
b	压缩残差每维的编码比特数	当然还是越大越好($2\to4$)，当k越小对于nRecall@k的提升就越大

2 端到端：LoTTe和BEIR上检索准确性和速度都优于优化后的XTR，另外查询编码是最耗时的



3 可扩展性：考虑不同数据集合不同并行度

1. 数据集大小的可扩展性：WARP延迟$\propto$数据集大小$^{1/2}$，这源于超参数的设置，由此避免了延迟线性增长
2. 并行处理的可扩展性：从单线程扩展到多线程也可带来WARP速度的提升，并且n_{probe}越大这种提升的幅度就越大

4 内存占用：$b\text{=}4$时较ScaNN显著减小了索引，$b\text{=}2$时(激进压缩)时检索质量还是要比FAISS好

5 性能比较：WARP本来是对XTR的优化，但是用WARP去加速ColBERTv2也不赖(与PLAID差不多)