面向快速学习型稀疏检索的引导遍历优化

Yifan Qiao、Yingrui Yang、Haixin Lin、Tao Yang
计算机科学系，加利福尼亚大学圣塔芭芭拉分校，加利福尼亚州，美国
{yifanqiao,yingruiyang,haixinlin,tyang}@cs.ucsb.edu

arXiv:2305.01203v1 [cs.IR] 2 May 2023

摘要

近期研究表明，由 BM25 驱动的动态索引跳过能够显著加速基于 DeepImpact 生成的学习型稀疏表示进行的 MaxScore 文档检索。本文研究了在采用 SPLADE 和 uniCOIL 等其他模型进行 top k 检索时，这种遍历引导策略的有效性，并发现当 BM25 模型与学习型权重模型未能良好对齐，或检索深度 k 较小时，不受约束的 BM25 驱动跳过可能导致明显的相关性下降。本文在前人工作的基础上，采用两级剪枝控制方案和模型对齐，对 BM25 引导的索引遍历进行了优化，从而实现基于稀疏表示的快速检索。尽管该方案可能会增加延迟，但在保持相关性效果的同时，其速度远快于没有 BM25 引导的原始 MaxScore 方法。本文分析了这种两级剪枝方案的竞争力，并在多个测试数据集上进行检索，评估其在排序相关性和时间效率之间的权衡。

ACM 参考文献格式：
Yifan Qiao、Yingrui Yang、Haixin Lin、Tao Yang。2023 年。面向快速学习型稀疏检索的引导遍历优化。载于 *2023* 年 *ACM Web* 会议（*WWW '23*），*2023* 年 *5* 月 *1–5* 日，美国德克萨斯州奥斯汀。ACM，纽约，纽约州，美国，11 页。https://doi.org/10.1145/3543507.3583497

1 引言

在大型数据集上进行文档检索时，通常会使用一种稀疏表示来表示文档特征向量，并通过倒排索引实现；倒排索引会将每个搜索词与包含该词项的文档列表关联起来。近年来，人们开发了学习型稀疏表示，使用神经模型（例如基于 Transformer 的检索器 [1, 9, 12, 14, 24, 30]）计算词项权重，并取得了出色的相关性结果，同时还可进行文档扩展（例如 [5]）。其缺点是，使用学习型稀疏表示进行 top k 文档检索的延迟远高于使用 BM25 模型时的延迟，正如 [29, 30] 中所述。在采用加性排名的传统 BM25 文档检索中，基于 top k 阈值的动态索引剪枝策略非常有效：它会在索引遍历过程中实时计算每个已访问文档的排名得分上界，从而跳过得分较低、无法进入最终 top k 列表的文档。广为人知的遍历采用此类动态剪枝策略的算法包括 MaxScore [41] 和 WAND [2], , 以及它们基于块的版本 Block-Max WAND（BMW） [11] 和 Block-Max MaxScore（BMM） [4, 10]。

Mallia 等人 [31] 提出使用 BM25 引导遍历，称为 GT，以实现快速的学习型稀疏检索，因为学习得到的权重分布导致可进行的剪枝机会较少；他们还使用检索模型 DeepImpact [30]进行了评估。他们提出的一种变体是将学习得到的权重与 BM25 权重进行线性组合，以计算最终的排名评分，记为 GTI。GT 是 GTI 的特例，本文将 GTI 作为主要基线。由于文档-词项对的 BM25 权重可能不存在于学习型稀疏索引中，Mallia 等人 [31]使用零填充来对齐 BM25 模型与学习得到的权重模型。在我们使用 GT 对 SPLADE v2 及其修订版本 SPLADE++ [12, 13]进行评估时，我们发现，随着检索深度 k 降低，BM25 驱动的跳过机制在丢弃那些基于学习得到的词项权重、理应进入前 k 名的文档时变得过于激进，从而可能导致显著的相关性下降。此外，GTI 的索引对齐仍有进一步改进的空间，以实现更准确的 BM25 驱动剪枝。

为解决上述问题，我们改进了早期关于结合 BM25 与学习得到的权重 [36]进行双重引导剪枝的研究。我们的工作通过约束 BM25 的剪枝影响，并提供一种替代性的平滑方法，使 BM25索引与学习得到的权重实现对齐，从而推广了 GTI。在第4节中，我们提出了一种带索引对齐的两级参数化引导方案，称为 2GTI，用于管理基于 MaxScore 的遍历过程中的剪枝决策。我们分析了 2GTI 在相关性行为方面的一些形式化性质，以及当 2GTI 在相关性上优于针对查询的两阶段 top k 检索算法时所需的配置条件。

第5节和附录A介绍了在 MS MARCO 数据集上使用 MaxScore 时，对采用 SPLADE++ [12–14] 和 uniCOIL [15, 24] 的 2GTI 进行的评估，同时还包括 DeepIm- pact [30] 。评估表明，当检索深度 k 较小时，或当 BM25索引与底层的学习型稀疏表示未能良好对齐时，2GTI 可以优于 GTI，并更有效地保持相关性。在某些情况下，这存在权衡：基于 2GTI 的检索可能比基于 GTI 的检索更慢，但 2GTI 仍然远快于不采用 BM25 引导的原始 MaxScore方法。在零样本相关性和检索延迟方面，2GTI 对 BEIR 数据集同样有效。在附录B中，我们将 2GTI 的应用扩展到了基于 BMW 的算法，例如 VBMW [32]。我们证明，对于一类短查询，并且当 k 较小时，结合 VBMW 的 2GTI 可能是有效的。

2 背景与相关工作

Top-k 文档检索问题旨在识别与查询相匹配的排名靠前的结果。文档表示使用特征向量来捕获文档语义。如果这些向量包含

允许免费制作本作品的部分或全部数字副本或纸质副本，用于个人或课堂使用，但前提是不得出于营利或商业利益而制作或分发副本，并且副本必须带有此声明以及首页上的完整引文。本作品中第三方组件的版权必须得到尊重。如需其他用途，请联系版权所有者/作者。*WWW '23*，*2023* 年 *5* 月 *1–5* 日，美国德克萨斯州奥斯汀 © 2023 版权所有归版权所有者/作者所有。ACM ISBN 978-1-4503-9416-1/23/04。https://doi.org/10.1145/3543507.3583497

零值远多于非零值时，这种表示就被认为是稀疏的。对于大型数据集，文档检索通常在搜索的第一阶段使用简单的加法公式，并计算每篇文档 d 的排名得分：

$$\sum_{t \in Q} w_t \cdot w(t,d), \tag{1}$$

其中，Q 是所有搜索词的集合，$w(t,d)$ 是词项 t 在文档 d 中的权重贡献，而 w_t 是独立于文档或特定于查询的词项权重。假设 $w(t,d)$ 可以进行静态或动态缩放，为简化表述，本文采用 $w_t = 1$。此类公式的一个例子是广泛使用的 BM25 [18]。对于稀疏表示，检索算法通常使用包含一组词项的倒排索引，以及每个词项对应的文档倒排列表。列表中的一条倒排记录包含文档标识符及其对应词项的权重。

基于阈值的跳过。在文档检索过程中遍历倒排列表时，先前的研究提出了动态剪枝策略，以跳过得分较低、不会出现在最终 top-k 列表 [2, 37]中的文档。为了跳过对某个文档的评分，剪枝策略会计算候选文档 d 的排序分数上界，记为 $Bound(d)$。

如果 $Bound(d) \leq \theta$ 其中 θ 是最终前 k 个列表中的排序分数阈值，则可以跳过此文档。例如，WAND [2]使用每个倒排列表中文档的最大词项权重，来确定枢轴文档的排名得分上界；而 BMW [11] 及其变体（例如 [32]）则通过基于块的最大权重来优化 WAND，以计算得分上界。MaxScore [41] 使用词项划分和前-k 阈值，跳过不必要的索引访问和评分计算。此前的工作还采用了一种“排序不安全”的跳过策略，故意将当前前-k 阈值高估 [2, 7, 27, 39]倍。

学习型稀疏表示。早期的稀疏表示研究包括 [43], DeepCT [9], 和 SparTerm [1]。近期相关工作包括 SPLADE [12–14]，它通过稀疏性控制学习用于文档扩展的词元重要性。DeepImpact [30] 则在经 DocT5Query [5]扩展的文档上学习神经词项权重。类似地，uniCOIL [24] 扩展了 COIL [15] 在上下文化词项权重方面的工作。研究发现，使用 Transformer 学得的词项权重进行文档检索速度较慢 [29, 31]。Mallia 等人 [31] 指出，MaxScore 检索算法无法有效利用 DeepImpact 得分。

Mackenzie 等人 [29] 认为，学习得到的稀疏词项权重会影响检索过程中的文档跳过，因此将其称为“古怪”，并主张采用按分数处理的遍历进行排序近似。

我们的方法采用 BM25 与学习得到的词项权重进行混合组合，其动机源于结合词法与神经排序的既有研究 [16, 22, 25, 26, 42]。GTI 将这一方法用于最终排序。我们工作的一个关键区别在于，混合评分用于两级剪枝控制，其公式可以不同于最终排序。多级混合评分的差异为进一步剪枝及其质量控制提供了机会。因此，2GTI 的结果并非 BM25 与学习得到的权重的简单线性排序组合；两级引导式剪枝会产生非线性集成效应，从而提高时间效率，同时保持相关性。我们的评估将与采用简单线性组合的 MaxScore 进行相关性和效率比较。

本文主要关注 MaxScore，因为已有研究表明，对于相对较长的查询，它的效果更好 [34]。我们还考虑 VBMW [32]，因为通常认为，在许多情况下它代表了最先进水平 [29]，尤其是当 k 较小时且查询长度较短时 [34]。

3 设计考量

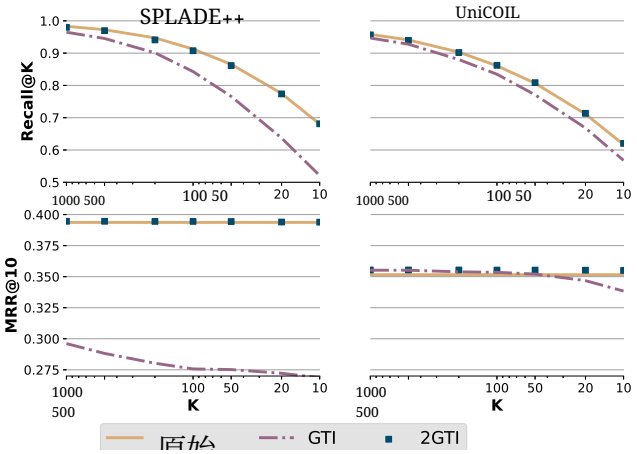

图 1：当 k 变化时的 Recall@k 和 MRR@10。

图 1 展示了在 MS MARCO 段落的 Dev 查询集上搜索时，原始 MaxScore 检索算法（不使用 BM25 引导）、GTI 和所提出的 2GTI 方案在 MRR@10 和 Recall@k 方面的表现，其中 top k 不断变化。这里，k 表示要检索的前排文档数量，在相关文献中有时也称为检索深度。第5节提供了更详细的数据集和索引信息。对于 SPLADE++和 uniCOIL，我们按照 [31] 构建 BM25 模型：首先使用 DocT5Query 扩展段落，然后使用 BERT 的 Word Piece 分词器对文本进行分词，并将 BM25 的词元选择与这些学习模型对齐。从图 1 可以看出，当 k 从 1,000 变化到 10 时，使用 GTI 会导致召回率和 MRR 显著下降。相关性下降有两个原因。

(1) 当排名靠前的文档数量 k 相对较少时，相关性会显著下降。由于 k 较小，动态更新的前 k 名得分阈值会更接近最佳文档的最高排名得分。落入前 k名的文档更少，而低于更新后的前 k 名得分阈值的更多文档会被提前移除。此时，跳过的准确性会变得更加敏感。BM25 评分与学习权重评分之间的差异可能导致 BM25 引导的剪枝不恰当地移除优质候选文档，从而使较小的 k出现显著的相关性下降。

(2) 对于采用 BM25 引导剪枝的 SPLADE++，其相关性下降幅度明显远大于 uniCOIL。这可能与 SPLADE++ 对每篇文档的词元进行不同且更激进的扩展有关

，相比之下，uniCOIL 并非如此。因此，SPLADE++索引中 98.6% 的词项-文档对即使经过 DocT5Query 文档扩展后，仍不存在于 BM25索引中；而 uniCOIL 的这一比例为 1.4%。因此，BM25 引导可能会变得不够准确，并不恰当地跳过更多相关文档。

鉴于上述考虑，我们的目标是在受约束的范围内控制 BM25 权重的影响，以谨慎地保障相关性，并在 BM25 索引与学习型稀疏索引未能良好对齐时，实现更好的权重对齐。在图 1 中，以蓝色方块标记的 2GTI 的 Recall@k 数值与未使用 BM25 引导的原始方法相近。两者的 MRR@10 数值相互重叠，形成近乎平坦的曲线，这表明即使k 下降，两者的 MRR@10 数值仍然相近。接下来的两个章节将分别介绍我们针对上述两个问题提出的解决方案。

4 两级引导遍历

4.1 MaxScore 的两级引导

我们假设每个词项的倒排列表都按文档 ID 升序排列。MaxScore 算法 [41]可视为执行一系列遍历步骤；在每个遍历步骤中，它都会进行词项划分，然后检查是否应跳过对选定文档的评分。我们将面向剪枝的操作分为以下两个层次。

• **全局层级**。MaxScore 使用每个词项的最大得分（上界）和当前已知的 top k 阈值，在每个索引遍历步骤中将词项划分为两个列表：必要词项列表和非必要词项列表。不包含必要词项的文档不可能出现在 top k 结果中，因此可以将其排除。在下一步索引遍历中，算法仅从必要词项的倒排列表中，选取最小的未访问文档 ID 作为起点。因此，索引访问由必要词项列表中最小文档 ID 指针的移动驱动。

我们将这一层剪枝称为全局剪枝，因为它指导跳过多个文档，并利用最大词项权重所隐含的文档间关系。图 2(a)展示了 MaxScore 中全局剪枝流程的一个示例，其中包含 4 个词项，并且每个倒排列表都维护一个指针，指向遍历步骤中当前正在访问的文档。词项划分确定了两个必要词项：t_3 和 t_4。这些必要词项的当前文档指针中最小的文档 ID 为 d_3，而任何小于 d_3 的文档 ID 都会在此遍历步骤中跳过，不再考虑。非必要词项列表中倒排列表的当前访问指针也会移动到大于或等于 d_3的最小文档 ID。

• **局部层级**。 一旦选定某个文档进行可能的完整评估，就可以利用各查询词对该文档的最大权重贡献或实际权重，估算该文档的排序分数上界并逐步收紧这一上界。将逐步优化后的分数上界与动态更新的 top k 阈值进行比较，从而获得另一次机会，完整或部分跳过对该文档的评估。我们将这一层级的跳过决策称为局部跳过，因为这种剪枝针对的是特定选定文档。图 2(b)展示了 MaxScore 中局部剪枝的一个示例。经过词项划分后选定的文档是 d_3

用于局部剪枝决策的是文档 d_3 在所有倒排列表中对应的最大权重或实际权重。

我们不直接使用 BM25 在全局层级和局部层级引导剪枝，而是提出在每个层级使用 BM25 权重与学习得到的权重的线性组合来引导跳过，如下所示。这样可以通过参数控制二者的影响。

• 我们为每个文档增量维护三个累积得分： $Global(d)$、 $Local(d)$ 和 $RankScore(d)$。 $Global(d)$ 用于全局剪枝， $Local(d)$ 用于局部剪枝，而 $RankScore(d)$ 用于最终排序。

$$Global(d) = \alpha RankScore_B(d) + (1-\alpha) RankScore_L(d)$$
$$Local(d) = \beta RankScore_B(d) + (1-\beta) RankScore_L(d)$$
$$RankScore(d) = \gamma RankScore_B(d) + (1-\gamma) RankScore_L(d)$$

其中 $0 \le \alpha,\ \beta,\ \gamma \le 1$， $RankScore_B(d)$ 使用 BM25 权重，遵循表达式 1； $RankScore_L(d)$ 使用学习得到的权重，遵循表达式 1。排名分数公式遵循 [31], 中的 GTI 设置； $\alpha = \beta = 1$ 的 2GTI 与 GTI 类似。$\alpha = \beta = \gamma$ 的 2GTI 与 MaxScore 检索相同，并且仅在 $\gamma = 0$ 时使用学习得到的神经网络权重。

• 对于每个已评估文档的上述三个得分，我们分别维护三个队列： Q_{Gl}、 Q_{Lo}、 Q_{Rk}，用于存放依据 $Global$（d）、d（$RankScore$）和 d（ k ）得分最高的 k 个文档。当某个队列中的得分最低的文档被移除时，不会与其他队列协调。这些队列用途各异：前两个队列用于控制全局剪枝和局部剪枝，最后一个队列则用于生成最终的前 Q_{Gl} 个结果。当某个文档因局部剪枝而不再纳入后续考虑时，该文档不会被加入全局和局部队列 Q_{Lo} 和 $RankScore$。但该文档可能已为其 d（ Q_{Rk} ）累积了部分得分，因此仍会被加入 k，以防该文档凭借部分得分和最新的 d（）值符合前 $RankScore$ 个结果的条件。

这三个队列产生了三个动态 top-k阈值 θ_{Gl}、θ_{Lo} θ_{Rk} 和 。它们可用于剪枝决策，以避免为获取或细化 $RankScore(d)$ 而进行任何进一步的评分工作。

修订后的 MaxScore 剪枝控制流： 图 2(c) 展示了为修订后的 MaxScore 算法新增的控制流。设 N 为查询词的数量。我们定义：

• 给定与 N 个查询词对应的 N 个倒排列表，每个 i倒排列表包含一系列倒排记录，每条记录包含文档 ID d、其 BM25 权重 $w_B(i,\ d)$ 和学习权重 $w_L(i,\ d)$。倒排记录按文档 ID 递增排序。

• 一个数组 σ_L，其中 N 表示 i 个词项对任意文档的学习权重最大贡献。$\sigma_L[i]$

• 一个 σ_B 数组， N，其中 $\sigma_B[i]$ 是第 i 个词项的 BM25 权重对任意文档的最大贡献。

• N搜索词已预先排序，使得 $\alpha\sigma_B[i] + (1-\alpha)\sigma_L[i] \le (\ )(\ )$，其中 。$\alpha\sigma_B[$**采用词项划分的全局剪枝**。对于每个查询词$1 \le i \le N$，我们找到从 1 到 N 的最大整数 $pivot$，使得$\sum_{j=1}^{pivot-1}(\alpha\sigma_B[j] + (1-\alpha)\sigma_L[j]) \le \theta_{Gl}$。从 $pivot$ 到 N 的所有词项均被视为必需的。 如果文档 d 不包含任何

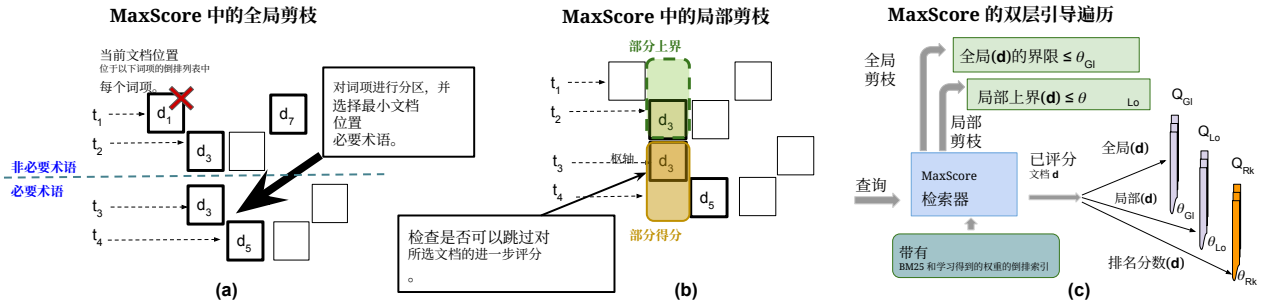

图 2：(a) 一个 d (b)：MaxScore 中两级剪枝的示例。(c) MaxScore 的两级引导遍历。

必要词项，$Global(d) \le \sum_{j=1}^{pivot-1} \alpha\sigma_B[j] + (1-\alpha)\sigma_L[j] \le \theta_{Gl}$ 的上界。根据全局得分，此文档无法进入最终前 k 个列表。随后跳过此文档，不将其加入三个队列中的任何一个。

确定上述 *pivot* 位置之后的必要术语列表后，令所有必要术语的倒排列表中当前 *pivot* 位置指针所指向的文档标识符中的下一个最小值为文档 d。我们也将其称为 *pivot* 文档。

局部剪枝。 接下来，我们检查是否可以完全或部分避免对所选枢轴文档 d 进行详细评分。按照 [40],中的实现，我们对该过程进行如下描述，并作出修改以使用混合评分：重复执行以下三个步骤，词项位置的初始值 x 为 *pivot* 位置，并且在每次循环迭代中 x 减 1。

- 令 $PartialScore_{Local}(d)$ 表示线性组合后，从位置 x 到 N 的倒排列表中文档 d 的所有词项权重之和。也就是说，当第 i 个倒排列表包含 d 时，$PartialScore_{Local}(d) = \sum_{i=x}^{N-1} \beta w_B(i, d) + (1-\beta) w_L(i, d)$，否则该值为 0。

  随着 x 减小，如果存在，则从 x 词项的倒排列表中提取枢轴文档 d 的词项权重。
- 设 $PartialBound_{Local}(d)$ 为文档 d 在前 x 个查询词的倒排列表中的局部部分得分上界。

$$PartialBound_{Local}(d) = \sum_{j=1}^{x} \beta\sigma_B[j] + (1-\beta)\sigma_L[j].$$

- 在上述计算过程中的任何时刻，如果

$$PartialBound_{Local}(d) + PartialScore_{Local}(d) \le \theta_{Lo},$$

将跳过对 *pivot* 文档 d 的进一步排名评分，且该文档不会出现在三个队列中的任何一个。图 2(b) 描述了为辅助剪枝决策而计算枢轴文档 d_3 的 $Local(d_3)$ 部分上界和部分得分。

复杂度。 2GTI 的复杂度与 MaxScore 和 GTI 相同。内存空间开销包括存储此查询所涉及的倒排索引以及三个队列所需的空间。时间复杂度与查询所涉及的倒排记录总数成正比，并乘以队列更新的 $\log k$ 开销。

倒排列表可以按块划分并压缩，Block MaxScore 也可以类似地使用 2GT，而先前的一项研究 [34] 表明，在几种压缩方案下，Block-Max MaxScore 实际上比 MaxScore 更慢。我们将在附录B中讨论 2GT 在基于块的BMW中的应用。

4.2 2GTI 的相关性属性

2GTI 除了进行排名得分组合外，还将 BM25 权重与学习得到的权重结合起来进行剪枝，从而生成一个 top k 排名列表，该列表可能不同于使用学习得到的权重进行加性排名所得的结果，也可能不同于 BM25 权重的线性组合所得的结果。因此，与这些基线中的任何一种相比，2GTI 都不具备排名安全性。两级剪枝由 2GTI 中配置的不同组合系数 α, β 和 γ 驱动；它们之间的数值差距为进一步剪枝提供了机会，同时 2GTI 会尽力保留相关性效果。如果此类剪枝有时会错误地跳过相关文档，2GTI 能否提供相关性保证？为从分析角度回答这个问题，本小节介绍了 2GTI 检索在相关性结果和竞争力方面的三个性质。

我们的分析将使用以下术语。给定查询 Q，令 R_x 表示给定数据集中所有文档的排名列表，这些文档按照其排名得分降序排列；排名得分基于其 BM25 权重与学习得到的权重按系数 x 进行线性组合，即文档 d 的 $\sum_{t \in Q} x * w_B(t, d) + (1-x) w_L(t, d)$。具体而言，存在三个排名列表：$R_\alpha$、$R_\beta$ 和 R_γ。2GTI 维护 3 个队列 Q_{Gl}、Q_{Lo} 和 Q_{Rk}，以及 3 个动态更新的 top k 阈值 θ_{Gl}、θ_{Lo}、θ_{Rk}。令 Θ_{Gl}、Θ_{Lo}、Θ_{Rk} 表示 2GTI 结束时这 3 个队列最终的 top k 阈值。也就是说，它是相应队列中第 k 个文档的排名得分。以下事实成立：

$$\theta_{Gl} \le \Theta_{Gl},\ \theta_{Lo} \le \Theta_{Lo},\ \text{and}\ \theta_{Rk} \le \Theta_{Rk}.$$

命题1 假设在任意交换得分相同的文档的排名位置后，每个 R_α、R_β和 R_γ中排名前 k 的文档子集均唯一。那么，出现在 R_α、R_β和 R_γ排名前k位置的任何文档，都位于*2GTI*的前k个结果中。

证明。对于出现在三个排序列表的前 k 个位置中的任意文档 d，

$$Global\ d \ge \Theta_{Gl} \ge \theta_{Gl}\ \ Local\ d \ge \Theta_{Lo} \ge (\ ),\ (\ )\ \text{和}\ (\ )\ \theta_{Lo}\ \ RankScore\ d \ge \Theta_{Rk} \ge \theta_{Rk}.$$

如果文档 d 在2GTI检索过程中被全局剪枝淘汰，则 $Global(d) = \Theta_{Gl} = \theta_{Gl}$，且文档 d 和排名列表 R_α 中第 $(k+1)$ 个文档的 R_α得分必须为 Θ_{Gl}。那么，在任意交换得分相同的文档的排名位置后，R_α 中的前 k 个文档子集并不唯一，这与假设矛盾。

同理，我们可以论证，在2GTI检索过程中将文档 d 添加到 Q_{Rk} 时，它不可能被局部剪枝淘汰，也不可能被 θ_{Rk} 拒绝。因此，该文档必须出现在2GTI的最终结果中。□

以下两个命题分析了在什么情况下，2GTI 的相关性优于一种称为$R2_{\alpha,\ \gamma}$的两阶段搜索算法。该算法从列表 R_α中获取前 k个结果，然后使用 R_γ的评分公式重新排序。

命题 2. 假设在任意交换得分相同的文档的排名位置后，R_α、R_β和 R_γ中各自排名前 k 的文档子集均唯一。若将 *2GTI* 配置为 $\alpha = \beta$或 $\beta = \gamma$，则 *2GTI* 生成的前 k个文档的平均值 R_γ基础排名得分不低于两阶段算法 $R2_{\alpha,\ \gamma}$生成的得分。

证明。令 $R2[k]$ 表示$R2_{\alpha,\ \gamma}$结果中的前 k 个文档子集。为证明此命题，我们比较 2GTI 结束时 $R2[k]$ 中文档的平均值 R_γ基础排名得分与 Q_{Rk} 中的得分。注意，对于任何满足$d \in R2[k]$的文档 d，它都位于排名列表 R_α的前 k 个结果中，并且根据此命题的假设，该前 k 个文档子集是确定的。因此， d不会被 2GTI 的全局剪枝消除。

对于在 2GTI 结束时满足 $d \in R2[k]$ 和 $d \notin Q_{Rk}$ 的任意文档 d，它要么通过阈值为 Θ_{Lo} 的局部剪枝被淘汰，要么通过对队列 Q_{Rk} 进行阈值为 θ_{Rk} 的 top k 筛选被淘汰。在后一种情况下， $RankScore(d) \le \theta_{Rk} \le \Theta_{Rk}$。当 d 通过局部剪枝被淘汰时，全局剪枝必须使用不同的公式，因为 d 不会被全局剪枝淘汰，因此 2GTI 必须配置为 $\beta = \gamma$，而不是 $\alpha = \beta$。在这种情况下，局部剪枝等同于使用阈值为 k 的 Q_{Rk} 进行淘汰。于是， $RankScore(d) \le \theta_{Rk} \le \Theta_{Rk}$。

由于 $R2[k]$ 和 Q_{Rk} 的大小均为 k，$|R2[k]-R2[k]\cap Q_{Rk}| = |Q_{Rk} - R2[k] \cap Q_{Rk}|$，我们可以推导出：

$$
\begin{aligned}
&\sum_{d\in R2[k]} RankScore(d) \\
&= \sum_{d\in R2[k]\cap Q_{Rk}} RankScore(d) + \sum_{d\in R2[k], d\notin Q_{Rk}} RankScore(d) \\
&\le \sum_{d\in R2[k]\cap Q_{Rk}} RankScore(d) + \sum_{d\in R2[k], d\notin Q_{Rk}} \Theta_{Rk} \\
&= \sum_{d\in R2[k]\cap Q_{Rk}} RankScore(d) + \sum_{d\notin R2[k], d\in Q_{Rk}} \Theta_{Rk} \\
&\le \sum_{d\in R2[k]\cap Q_{Rk}} RankScore(d) + \sum_{d\notin R2[k], d\in Q_{Rk}} RankScore(d) \\
&= \sum_{d\in Q_{Rk}} RankScore(d).
\end{aligned}
$$

Thus

$$
\frac{1}{k}\sum_{d\in R2[k]} RankScore(d) \le \frac{1}{k}\sum_{d\in Q_{Rk}} RankScore(d).
$$

□

定义 1. 对于一个数据集，其中针对任何测试查询，文档仅被标记为相关或不相关，我们称排名列表 R_x 排序匹配 R_y，如果每当 R_y 针对某个查询正确排列一对相关文档和不相关文档时， R_x 也会将它们正确排列。

命题 3。 假设数据集中的文档对于测试查询仅标记为相关或不相关。给定一个查询，当 R_γ 优于 R_β，而 R_β 又优于 R_α 时，*2GTI* 在前 k 个位置检索到的相关文档数量不少于两阶段算法 $R2_{\alpha,\ \gamma}$。

证明。当 2GTI 完成对某个查询的检索时，我们分别统计列表 R_α、队列 Q_{Lo} 和队列 Q_{Rk} 中前 k 个位置上的相关文档数量，并将其记为 c_α,c_β 和 c_γ。为了证明$c_\alpha \le c_\beta$，我们首先将它们初始化为 0，然后运行以下循环，迭代计算 c_α 和 c_β。循环索引变量 i 从 k、$k-1$ 依次变化，直到 1。在每次迭代中，我们查看 R_α 中位置 i 处的文档 x，以及 Q_{Lo} 中位置 i 处的文档 y。设 L_x 和 L_y 分别为它们的二元标签，其中值 1 表示相关，值 0 表示不相关。

- 如果 $L_x = L_y$，我们将 L_x添加到 c_α和 c_β中。继续此循环。
- 现在 $L_x \ne L_y$。如果 $L_x = 0$、 $L_y = 1$，我们将 1 加到 c_β，并继续循环。如果 $L_x = 1$、 $L_y = 0$，则有两种情况：– 如果 x位于当前 Q_{Lo}的前 i个位置内，我们将 1 分别加到 c_α和 c_β。交换文档 x 和 y在 Q_{Lo}中的位置。继续此循环。– 如果 x不在 Q_{Lo}的前 i个位置内，由于 x位于 R_α的前 k个位置内，因此无法被全局剪枝，并且将由 2GTI 进行评估，以判断其是否有可能进入 Q_{Lo}。如果 x 在列表 R_α 中排在 y 之前，并且由于 R_β 胜过 R_α，因此 x 必须在 R_β 和 Q_{Lo}中都排在 y之前。这是矛盾的。如果 x在 R_α中排在 y之后，我们就交换 x和 y在 R_α中的位置。继续此循环。

上述过程会不断重复并移至更高的位置，直到$i = 1$。当 $i = 1$时，若 R_α 中的排名第一文档为 x，且 Q_{Lo}中的排名第 1 y位文档为 $L_x = L_y$，则仅有以下可能情况： $L_x = 0$和 $L_y = 1$。因此，在上述过程结束时，$c_\beta \ge c_\alpha$。

同样，我们可以验证 $c_\gamma \ge c_\beta$，因为 R_γ胜过 R_β。因此，$c_\gamma \ge c_\beta \ge c_\alpha$。对于 2GTI，检索到的位置 k 以内的相关文档数量为 c_γ；而对于 $R2_{\alpha,\ \gamma}$，检索到的位置 k 以内的相关文档数量为 c_α。因此，该命题成立。□

上述分析表明，三个排序结果 R_α、 R_β 和 R_γ一致排在前列的文档，在 2GTI 中始终会保留在前列；并且，经过适当配置的 2GTI 算法在相关性方面可以优于两阶段检索与重排序算法，尤其是在针对某个查询时，排序结果 R_γ 胜过 R_β，且 R_β 胜过 R_α 的情况下。由于先进行 BM25 检索、再进行神经重排序的两阶段搜索已在相关文献中广泛采用，这项分析为 2GTI 在两级剪枝条件下的“最坏情况”相关性竞争力提供了有益的见解。在使用相同索引时，GTI 可视为 $\alpha = \beta = 1$ 的 2GTI 特例；上述三个命题对 GTI 也成立。与 GTI 相比，2GTI 在带质量控制的剪枝方面提供了更大的灵活性，第5节将进一步评估二者的相关性差异。

4.3 词元与权重的对齐

BM25 模型通常基于原始文档集或扩展文档集上的词级分词构建，而流行的扩展方法使用 DocT5Query，并采用相同的分词方法。当学习型表示采用不同的分词方法

例如 BERT 的 WordPiece，它基于 BERT 词汇表中的子词，我们需要将其与 BM25 对齐，以保持词项引用的一致性。例如，当使用 BM25 引导 SPLADE 索引的遍历时，会先使用 WordPiece 分词器对通过 DocT5Query 扩展的文档进行处理，然后再将 BM25 权重应用于每个词元。一旦词元完成对齐，从索引的角度来看，同一个词元会根据 BM25 权重和 SPLADE 分别对应两个不同的倒排列表。当倒排项无法一一对齐时，为了合并这些倒排列表，按照 [31] 中提出的方法，将缺失的权重设为零。我们将这种方法称为零填充对齐。作为替代方案，我们还提出了另外两种方法，用于以更好的权重平滑性填充缺失权重。

• **一填充对齐**。当某个词元缺失于 BM25 模型中，但出现在文档的学习得到的词元列表中时，我们将其词频设为 1。理由是，如果神经模型学习得到的结果表明某个词项对文档有用，那么将其权重设为 0 未免过于突兀。词频为 1 意味着该词元存在于文档中，只是取值最低。

• **缩放对齐**。这种替代方案根据缩放后的学习得到的分数，替换 BM25 模型中缺失的权重；缩放比例取两个模型中非零权重均值的比值。对于包含词项 t 的文档标识符 d，设其 BM25 权重为 $w_B(t,d)$，学习得到的权重为 $w_L(t,d)$。设 $w_B^*(t,d)$ 为调整后的 BM25 权重。集合 P_B 包含所有 BM25 权重非零的倒排记录。集合 P_L 包含学习得到的权重非零的倒排记录。然后，$w_B^*(t,d)$ 定义为：

$$w_B^*(t,d) = \begin{cases} w_B(t,d), & w_B(t,d) \neq 0, \\ \dfrac{\sum_{(t',d')\in P_B} w_B(t',d')/|P_B|}{\sum_{(t',d')\in P_L} w_L(t',d')/|P_L|} w_L(t,d), & w_B(t,d) = 0. \end{cases}$$

5 评估

表 1：MS MARCO 开发集的模型特征

索引	平均查询长度	倒排项数	Size	合并
MS MARCO 段落				
BM25-T5	4.5 (4.5)	644M	1.2G	-
DeepImpact	4.2 (4.2)	644M	2.6G	2.6G
BM25-T5-B	6.6 (6.6)	699M	1.2G	-
UniCOIL	6.6（686.3）	592M	1.5G	1.7G
SPLADE++	23.3 (867.6)	2.62B	5.6G	8.3G
MS MARCO 文档				
BM25-T5-B	6.8 (7.0)	3.39B	5.4G	-
UniCOIL	6.6（685.0）	3.04B	7.0G	8.3G

数据集与设置。 我们的评估使用 MS MARCO 文档和段落集合 [3, 8], 以及 13 个公开可用的 BEIR 数据集 [38]。BEIR 数据集的结果见附录 A。对于 MS MARCO，文档集合中的内容在建立索引时进行分段，并在检索后按照 [23]所述的“最大段落”策略重新分组。分段集合中包含 880 万个段落，平均长度为 55 个词；分段前包含 320 万个文档，平均长度为 1131 个词。用于段落和文档排序的开发集查询集分别包含 6980 个和 5193 个查询，每个查询约有一个相关性判断标签。TREC 深度学习（DL）2019 和 2020 赛道中的段落/文档排序任务分别提供了 43 个和 54 个查询，每个查询有多个相关性判断标签。

构建倒排索引时，所有单词均使用小写字母。按照 GT 的做法，我们将学习得到的分数和词频打包到同一个整数中。对于 DeepImpact，我们直接采用 GT 的索引[1]。BM25-T5 的索引由 DeepImpact 的索引导出。BM25-T5 和 DeepImpact 均采用自然词语分词。

SPLADE 和 uniCOIL 使用 BERT 的 Word Piece 分词器。为了与它们保持一致，以下表格中报告的 BM25-T5-B 索引也使用相同的分词器。uniCOIL 的影响力分数取自 Pyserini [23] [2]。对于 SPLADE，为了获得最佳性能，我们按照 SPLADE++ [13] 中的设置重新训练了模型。我们以预训练模型 coCondenser [6] 为起点，使用来自交叉编码器教师模型 [35] 的 sentenceBERT 硬负例 [3] 进行蒸馏，并采用 MarginMSE 损失。对于 FLOP 正则化，我们分别为查询和文档使用 0.01 和 0.008。我们构建倒排索引，将其转换为 PISA 格式，并按照 [31, 34]使用 SIMD-BP128 [21] 进行压缩。

表 1 展示了 MS MARCO 开发集上不同加权模型的数据集和索引特征。随后[29], 我们假设，可以使用“伪文档”技巧对查询进行预处理，从而为 uniCOIL 和 SPLADE 中的查询词分配自定义权重。因此，每个查询中可能会出现词元重复，以反映词元权重。第 1 列是查询长度的平均值，分别表示不计重复词元和计入重复词元时的长度。第 3 列是倒排索引的大小，最后一列是索引中合并 BM25 和学习得到的权重后的大小。

2GTI 的 C++实现采用修改后的 MaxScore 和 VBMW 算法，已集成到 PISA [33],中，代码将在 https://github.com/Qiaoyf96/2GTI 发布。使用该实现进行的评估在一台配备 Intel i5-8259U 2.3GHz 处理器和 32GB 内存的 Linux 服务器上以单线程运行。权重通过从 MS MARCO 训练数据集中抽样查询来确定。

指标。 对于 MS MARCO 开发集，我们按照官方排行榜标准，以平均倒数排名报告相关性（段落采用 MRR@10，文档采用 MRR@100）。我们还报告 recall@k 比率，即最终前-k 个结果中相关性标注结果所占的百分比。对于 TREC DL 测试集，我们报告归一化折损累计增益（nDCG@10） [17]。上述报告方式遵循以往工作的常见做法（例如 [12, 15, 16, 30]）。

在对查询计时之前，按照 [19, 32]中的相同假设，先将待测查询的所有压缩倒排列表和元数据预加载到内存中。检索平均响应时间（MRT）以毫秒为单位报告。第99百分位时间（P_{99}）在下方表格中以括号标出，对应于 [28]中称为尾延迟的第99百分位时间。

统计显著性。 对于本节其余部分报告的 MS MARCO 段落集和文档 Dev 集上的数值，我们对 2GTI 与 GTI 基线之间，以及 2GTI 与未经 BM25 引导的原始学习型稀疏检索之间的相关性差异进行了成对 t 检验。未发现具有统计显著性的

[1] https://github.com/DI4IR/dual-score [2] https://github.com/castorini/pyserini/blob/master/docs/experiments-unicoil.md [3] https://huggingface.co/datasets/sentence-transformers/msmarco-hard-negatives

表 2：针对 MaxScore，对比 2GTI、GTI 和未使用 BM25 引导的原始方法

	$k = 10$						$k = 1000$					
	MS MARCO 开发集		DL'19		DL'20		MS MARCO 开发集		DL'19		DL'20	
方法	平均倒数排名（召回率）	MRT（P_{90}）	归一化折损累计增益（召回率）	MRT	归一化折损累计增益（召回率）	MRT	平均倒数排名（召回率）	MRT (P_{90})	归一化折损累计增益(召回率)	MRT	归一化折损累计增益(召回率)	MRT
SPLADE++，段落。$\alpha = 1$。对于 2GTI-Accurate：$\beta = 0$；对于 2GTI-Fast：$\beta = 0.3$ ($k = 10$)，0.9 ($k = 1000$)。对于 GTI 和 2GTI：$\gamma = 0.05$。												
BM25-T5-B	0.2611 (0.5179)	1.7 (8.7)	0.5931 (0.1556)	0.8	0.5981（0.2034）	1.0	0.2611（0.9361）	9.2 (28.4)	0.5931（0.7608）	6.4	0.5981 (0.7641)	7.4
SPLADE++-组织	0.3937 (0.6801)	121 (483)	0.7304 (0.1776)	135	0.7290 (0.2437)	138	0.3937 (0.9832)	278 (819)	0.7304 (0.8286)	317	0.7290 (0.8287)	307
-GT	0.2720（0.5246）	121 (438)	0.6379 (0.1677)	130	0.6106 (0.2192)	139	0.2973 (0.9648)	336 (1048)	0.6636 (0.8030)	330	0.6605 (0.8072)	344
-GTI	0.2687 (0.5209)	118 (440)	0.6352 (0.1669)	131	0.6083 (0.2190)	139	0.2961 (0.9648)	332 (1059)	0.6595 (0.8025)	318	0.6587 (0.8066)	333
-2GTI-准确	**0.3939**† **(0.6812)**	31.1 (171)	**0.7401 (0.1846)**	31.9	**0.7278 (0.2480)**	36.8	**0.3946**† (0.9799)	109 (478)	**0.7394** (0.8209)	123	0.7297 **(0.8339)**	132
-2GTI-快速	0.3934† (0.6792)	**22.7 (116)**	0.7380 (0.1837)	**23.5**	**0.7278 (0.2480)**	**26.2**	0.3937† (0.9662)	**43.1 (144)**	**0.7394** (0.8218)	**42.1**	**0.7306** (0.8205)	**45.0**
uniCOIL，段落。$\alpha = 1$。对于 2GTI-Accurate：$\beta = 0$；对于 2GTI-Fast：$\beta = 0.3$（$k = 10$），1（$k = 1000$）。对于 GTI 和 2GTI：$\gamma = 0.1$。												
BM25-T5-B	0.2611 (0.5179)	1.7 (8.7)	0.5931 (0.1556)	0.8	0.5981 (0.2034)	1.0	0.2611 (0.9361)	9.2 (28.4)	0.5931 (0.7608)	6.4	0.5981 (0.7641)	7.4
UniCOIL-Org	0.3516 (0.6168)	10.5 (102)	0.7027 (0.1761)	10.4	0.6746 (0.2346)	14.2	0.3516 (0.9582)	35.3 (197)	0.7027 (0.7822)	38.6	0.6746 (0.7758)	42.8
-GT	0.3347 (0.5639)	**2.1 (11.2)**	0.6990 (0.1770)	1.7	0.6769 (0.2444)	**2.4**	0.3514 (0.9458)	**10.6** (33.9)	0.7028 (0.7857)	**10.3**	0.6746 (0.7741)	**10.8**
-GTI	0.3384 (0.5678)	**2.1 (11.2)**	0.6959 (0.1733)	**1.6**	0.6739 (0.2422)	**2.4**	0.3552 (0.9468)	**10.6 (33.2)**	**0.7130 (0.7917)**	**10.3**	**0.6899 (0.7857)**	**10.8**
-2GTI-准确	**0.3550**† **(0.6205)**	3.3 (19.2)	**0.7135 (0.1769)**	2.4	**0.6891 (0.2451)**	3.4	**0.3554** (0.9566)	16.9 (68.4)	**0.7130** (0.7904)	15.5	**0.6899** (0.7823)	16.5
-2GTI-快速	0.3548† (0.6193)	2.6 (14.3)	**0.7135 (0.1769)**	1.9	**0.6891 (0.2451)**	2.8	0.3552 (0.9468)	**10.6 (33.2)**	0.7129 (**0.7917**)	**10.3**	**0.6899 (0.7857)**	**10.8**
UniCOIL，文档。$\alpha = 1$。对于 2GTI-Accurate：$\beta = 0$；对于 2GTI-Fast：$\beta = 0.5$ ($k = 10$)，1 ($k = 1000$)。对于 GTI 和 2GTI：$\gamma = 0.1$。												
BM25-T5-B	0.2716 (0.4749)	2.7（13.9）	0.4246 (0.0741)	3.6	0.4463（0.1693）	3.1	0.2950 (0.9197)	12.4 (50.0)	0.5594 (0.5690)	15.0	0.5742 (0.7148)	15.7
UniCOIL-Org	0.3313 (0.5638)	26.0 (252)	0.5477 (0.0880)	28.1	0.4996 (0.1920)	27.3	0.3530 (0.9426)	71.0 (447)	0.6415 (0.5864)	79.2	0.6059 (0.7502)	86.0
-GT	0.3280 (0.5455)	**6.8 (36.2)**	0.5199 (0.0779)	6.2	0.4903 (0.1871)	**7.1**	0.3530 (0.9361)	20.9 (73.8)	0.6445 (0.5842)	**21.6**	0.6059 (0.7444)	22.7
-GTI	0.3334 (0.5531)	6.9 (36.6)	0.5223 (0.0781)	**6.1**	0.4905 (0.1857)	7.2	0.3639 (0.9368)	**20.6（72.9）**	**0.6581** (0.5920)	21.7	**0.6156** (0.7445)	22.6
-2GTI-准确	**0.3423**† **(0.5710)**	10.2 (56.4)	**0.5486（0.0852）**	8.9	**0.4998 (0.1886)**	10.5	**0.3644 (0.9422)**	33.2（149）	**0.6581 (0.5932)**	32.8	**0.6156 (0.7477)**	37.4
-2GTI-快速	0.3418† (0.5663)	7.4 (40.4)	0.5453 (0.0847)	6.5	0.4997 (0.1845)	8.2	0.3639 (0.9368)	**20.6** (73.0)	**0.6581** (0.5920)	21.7	**0.6156** (0.7445)	**22.5**
DeepImpact，段落。$\alpha = 1$。对于 2GTI-Accurate：$\beta = 0$；对于 2GTI-Fast：$\beta = 0.5$ ($k = 10$)，1（$k = 1000$）。对于 GTI 和 2GTI：$\gamma = 0.5$。												
BM25-T5	0.2723 (0.5319)	0.7 (4.7)	0.6283 (0.1611)	0.3	0.6321 (0.2218)	0.5	0.2723 (0.9348)	4.9 (21.3)	0.6283 (0.7704)	4.6	0.6321 (0.7598)	5.6
DeepImpact-Org	0.3276 (0.5844)	9.4 (73.3)	0.6964 (0.1698)	4.9	0.6524 (0.2035)	7.8	0.3276 (0.9474)	23.8 (97.0)	0.6964 (0.7623)	25.5	0.6524 (0.7534)	38.9
-GT	0.3276 (0.5805)	**0.7 (4.7)**	0.6997 (0.1698)	**0.3**	0.6682 (0.2194)	**0.5**	0.3276 (0.9454)	**5.1 (21.1)**	0.6964 (0.7745)	4.9	0.6527（0.7574）	5.8
-GTI	0.3375 (0.5866)	**0.7**（**4.7**）	0.6953(0.1690)	**0.3**	0.6846 (**0.2372**)	**0.5**	0.3413 (0.9455)	5.2 (21.7)	**0.7072** (0.7871)	**4.7**	**0.6854** (0.7745)	**5.7**
-2GTI-准确	**0.3405**† **(0.5987)**	1.1 (8.1)	**0.7065 (0.1703)**	0.7	0.6850 (0.2361)	1.2	**0.3414** (0.9469)	7.4 (33.0)	**0.7072 (0.7875)**	7.8	**0.6854**（**0.7778**）	9.1
-2GTI-快速	0.3395† (0.5934)	**0.7** (5.1)	0.7045 (0.1693)	0.4	**0.6882** (0.2371)	0.6	0.3413 (0.9455)	5.2 (21.7)	**0.7072** (0.7871)	**4.7**	**0.6854** (0.7745)	**5.7**

性能下降。95%置信水平下未观察到显著退化。我们还对 2GTI 和 GTI 报告的相关性数值进行了成对 t 检验；如果 2GTI 相较于 GTI 的提升在 95%置信水平下具有统计显著性，我们会在评估表中标注“†”。由于 DL'19 和 DL'20 查询集中的查询数量较少，我们不对这些查询集进行 t 检验。

使用 MS MACRO 的总体结果。表 2 列出了 2GTI 与基线方法的比较，使用三种稀疏表示在 MS MARCO 和 TREC DL 数据集上进行检索。2GTI 默认使用缩放填充对齐，而 GTI 按照 [31] 中的规定使用零填充。对于每种表示，GTI 和 2GTI 选择相同的 γ 值，该值在大多数情况下都是最佳值。“精确”配置表示达到最高相关性得分的配置。“快速”配置表示相关性得分在精确配置的 1% 以内、同时速度快得多的配置。

SPLADE 中的 2GTI 与 GTI++. 表 2 显示，在 BM25 索引未良好对齐的情况下，采用默认缩放填充的 2GTI 在 SPLADE++ 上的性能显著优于采用默认零填充的 GTI。“SPLADE++- Org”表示使用我们自行训练的 SPLADE++ 模型所获得的原始 MaxScore 检索性能，其 MRR@10 数值高于 [13] 中报告的结果。当 $k = 1{,}000$ 时，GT 略优于 GTI；而在快速配置下，对于 Dev 集，2GTI 的 MRR@10 比 GT 高 32.4%，速度则达到 GT 的 7.8 倍。在 DL'19 和 DL'20 上也观察到了 NDCG@10 的显著提升和 MRT 的显著下降。当 $k = 10$，对于所有三个测试集，从 GTI 或 GT 提升至 2GTI，相关性也显著提高，耗时则大幅缩短。例如，对于 Dev集，相关性分别提高了 46.4% 或 44.6%，平均延迟分别快了 5.2 倍或 5.3 倍。

与原始 MaxScore 方法相比，2GTI 在 $k = 10$ 和 $k = 1{,}000$ 时具有几乎相同的相关性得分，同时延迟却显著降低。例如，对于 Dev段落集，当 $k = 1{,}000$ 时，延迟降低了 6.5 倍（278ms 对比 43.1ms）；在使用 2GTI-Fast 快速配置且 $k = 10$ 时，延迟降低了 5.3 倍（121ms 对比 22.7 ms）。

DeepImpact 和 uniCOIL 中的 2GTI 与 GTI 对比。如表 2 所示，在 DeepImpact 和 uniCOIL 中，GTI（或 GT）在 $k = 1{,}000$ 的情况下表现出色，能够加速检索，同时保持与原始检索相近的相关性。用于动态索引剪枝的两级差异化机制并未提高相关性，也未缩短检索时间。这可以解释为 BM25-T5 索引与 DeepImpact 索引以及 uniCOIL 索引具有良好的对齐性。此外，出于同样的原因，在这两种情况下，无需通过填充来解决索引对齐问题，因为这样做并不会带来任何改进。

当 k 从 1,000 降至 10 时，如第 3 节讨论的图 1 所示，召回率开始下降，相关性效果也随之退化。当 $k = 10$ 如表 2 所示时，DeepImpact-2GTI-fast 可将开发集上的 MRR@10 从 GTI 的 0.3375 提高至 0.3395，并且在 DL'19 和 DL'19 数据集上的 MRR@10 或归一化折损累计增益@10 得分略高于 GTI。对于 uniCOIL，2GTI-fast 将开发集上的 MRR@10 从 GTI 的 0.3384 提高至 0.3548，并将 DL'19 上的归一化折损累计增益@10 从 0.6959 提高至 0.7135。存在

对于带有 $k = 10$ 的 DL'20 段落，相关性也略有提升，并且在文档检索任务中观察到了类似的趋势。采用 2GTI 所付出的代价是检索延迟有所增加，但其延迟仍远小于原始检索时间。

权重对齐和阈值高估的设计选项。表3考察了在使用 SPLADE++、且 $k = 10$ 时，权重对齐以及基于阈值高估的替代设计对 MS MARCO 段落 Dev 集的影响。在表格上半部分中，原始检索算法在没有 BM 引导的情况下，将阈值高估

F 1 1, 倍（. . 1.3, 1.5 25 和.），这些倍数的选择与 [7, 27, 39]中的相似。这实际上是将 $\alpha = 0$、$\beta = 0$ 和 $\gamma = 0$ 设为原值，同时在 2GTI 中将 θ_{GI} 和 θ_{Lo} 乘以上述倍数。结果表明，尽管阈值高估可以缩短检索时间，但相关性降低显著，这意味着使用过于激进的阈值会错误地丢弃一些目标文档。

表3的第二部分考察了第4.3节中所述的不同权重填充方法在分别应用于 GTI 和 2GTI 以进行对齐时的影响。在这两种情况下，标记为"/s"的缩放填充效果最佳，而标记为"/1"的单值填充也优于标记为"/0"的零值填充。2GTI/s 的 MRT 比 2GTI/0 小 10.5 倍，且其 MRR@10 没有受到负面影响。GTI/s 的 MRT 比 GTI/0 小约 13.0 倍，同时 MRR@10 大幅提升。

表3：设计选项对 MS MARCO 段落的影响

SPLADE++。 $k = 10$	**MRR@10**	**召回率@10**	**MRT**	P_{99}
阈值高估				
原始	0.3937	0.6801	121	483
- $F = 1.1$	0.3690	0.5707	107	457
- $F = 1.3$	0.3210	0.4393	95.0	420
- $F = 1.5$	0.2825	0.3670	88.2	393
GTI 的权重对齐（$\alpha = 1,\ \beta = 1,\ \gamma = 0.05$）				
GTI/0	0.2687	0.5209	118	440
GTI/1	0.3036	0.5544	26.7	114
GTI/s	0.3468	0.5774	9.1	36.1
2GTI-Accurate 的权重对齐（$\alpha = 1,\ \beta = 0,\ \gamma = 0.05$）				
2GTI/0	0.3933	0.6799	3281262	
2GTI/1	0.3933	0.6818	89.3	393
2GTI/s	0.3939	0.6812	31.1	171

对 2GTI 属性的验证。为佐证第4.2节中的竞争力分析，表4给出了在 MS MARCO 段落的 Dev集上，采用 $k = 10$ 和 SPLADE++ 权重时，不同配置的算法的 MRR@10 得分和检索时间（毫秒）。结果表明，所列的 2GTI 配置的 MRR@10 高于两阶段搜索 $R2_{\alpha,\ \gamma}$，以及采用 $\alpha = \beta = 1$ 、表现为 GTI 的 2GTI。使用 BM25 和学习得到的权重进行简单线性组合排序，其 MRR@10 仅略高于 2GTI，但速度慢得多。

权重分布的敏感性。我们以多种方式扭曲了 SPLADE++ 权重分布，以考察 2GTI 的敏感性，并发现 2GTI 依然有效。例如，我们对 MS MARCO 段落中每个词元的神经网络权重应用平方根函数。由于权重失真，原始检索和 2GTI 的相关性得分均降至 0.356 MRR@10，而当 $k = 10$ 时，2GTI 的速度仍比原始 MaxScore 快 5.0 倍。

表4：对 2GTI 属性的验证。 $k = 10$

	MRR@10	**Recall@10**	**MRT**	P_{99}
$R2_{\alpha,\gamma}$(BM25 检索。SPLADE++重排序)	0.3461	0.5179	-	-
GTI/s ($\alpha = \beta = 1, \gamma = 0.05$)	0.3468	0.5774	9.1	36.1
2GTI/s（$\alpha = 1,\ \beta = \gamma = 0.05$）	0.3939	0.6812	29.8	165
2GTI/s 精确（α=1, β=0, γ=0.05）	0.3939	0.6812	31.1	171
2GTI/s-Fast ($\alpha = 1, \beta = 0.3,\ \gamma = 0.05$)	0.3934	0.6792	22.7	116
线性组合。($\alpha = \beta = \gamma = 0.05$)	0.3946	0.6805	120	477

表 5：将 2GTI 用于新的 SPLADE 模型 [20]. $k = 10$

BT-SPLADE-L	**MRR@10**	**Recall@10**	**MRT**	P_{99}
原始 MaxScore	0.3799	0.6626	17.4	59.4
2GTI/s（α=1, β=0.3, γ=0.05）	0.3772	0.6584	8.0	27.5
GTI/s（$\alpha = \beta = 1, \gamma = 0.05$）	0.3284	0.5520	6.6	24.9

高效的 SPLADE 模型。 表 5 展示了将 2GTI 应用于近期发布的一种高效 SPLADE 模型 [20] 的效果，该模型在检索速度方面进行了多项改进。我们使用了这款高效模型已发布的检查点 BT-SPLADE-L。它的 MRR@10 得分较低，但速度显著快于表 2 中报告的、由我们训练的 SPLADE 基线模型。使用这款新的 SPLADE 模型时，2GTI/s-Fast 版本的检索时间比 MaxScore 快 2.2 倍。其 MRR@10 高于 GTI/s，与原始 MaxScore 相比，性能下降不到 1%。

6 结论

本文的贡献是提出一种具有索引对齐的两级参数化引导方案，以优化学习型稀疏表示的检索遍历。我们的形式化分析表明，配置得当的 2GTI 算法（包括 GTI）在相关性方面可以优于两阶段检索与重排序算法。

我们的评估表明，所提出的 2GTI 方案能够使 BM25 剪枝引导更加准确，从而保持相关性。对于 MS MARCO 段落上使用 SPLADE ++ 的 MaxScore，当 $k = 1$ 时，000 时，2GTI 可将相关性提升至多 32.4%，且速度比 GTI 快 7.8 倍；当 $k = 10$ 时，准确度最高可提升 46.4%，速度快 5.2 倍。在所有评估案例中，2GTI 都比不使用 BM25 引导的原始检索快得多。例如，当 $k = 10$ 时，它比使用 SPLADE++ 的 MaxScore 最多快 6.5 倍。在使用 SPLADE++ 学习得到的权重时，我们还观察到，在 BEIR 数据集上，将 2GTI 与 GTI 及原始 MaxScore 进行比较，也呈现出相似的性能趋势。与通过阈值低估来减轻 BM25 权重影响等其他方案相比，两级控制能够以低得多的时间成本，更准确地保持较强的相关性。尽管我们的研究主要聚焦于基于 MaxScore 的检索，2GTI 也可用于 VBMW；我们的评估表明，对于一类不含停用词的短查询，当 k 较小时，VB MW-2GTI 是更可取的选择。

致谢。感谢谢文泰、王熙越、熊天博以及匿名审稿人提供宝贵意见和／或帮助。本研究部分由 NSF IIS-2225942 资助，并使用了 NSF 支持的 XSEDE 和 ACCESS 计划提供的计算资源。本文中表达的任何观点、研究结果、结论或建议均为作者的观点，不一定反映 NSF 的立场。

A 额外评估结果

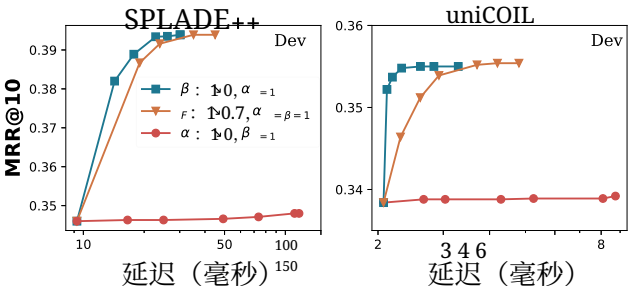

图 3：控制 BM25 对剪枝的影响

对 α 和 β 调整对 2GTI 的影响。图 3 考察了在控制 SPLADE++ （左）和 uniCOIL（右）中 BM25 权重影响时，调整参数 α 和 β 对 MS MARCO Dev 段落测试集全局和局部剪枝的影响。 x 轴表示延迟增幅，而 y 轴表示 MRR@10 或 NDCG@10 的增幅。MS MARCO DL'19 和 DL'20 的结果类似。

红色带点曲线固定 $\beta = 1$，并使 α 从左端的 1 变化到右端的 0。随着 α 从 1 减小到 0，延迟会增加，因为 BM25 在全局剪枝层级的影响逐渐减弱，跳过的文档数量也随之减少。总体而言，该曲线的相关性相对平稳，且低于蓝色曲线。这表明，全局层级的 BM25 引导能够显著缩短耗时，同时对相关性的影响较小。

蓝色方形曲线固定 $\alpha = 1$ 于全局层级，并将 β 从左下端的 1 逐渐调整至右上端的 0。总体而言，降低 β 值通常有利于相关性，因为局部层级的 BM25 影响会逐渐减弱；但超过某个点后，相关性增益会大幅缩小，甚至转为负值。例如，在 SPLADE 的蓝色曲线中，对于 Dev集，当 β 降至 0.3 后，进一步降低它已不会明显提升 MRR@10，而延迟仍在增加，这表明相关性收益在该点已达到峰值。我们在测试数据集上的经验是，2GTI 的参数设置通常在 α 接近 1 且 β 介于 0.3 和 1 之间时达到相关性峰值。

请注意，尽管上述结果也表明 α 接近 1，但对于测试数据而言， α和 β 仍有不同的取值才能取得更好的效果，这体现了两级剪枝控制的实用性。

阈值低估。 在图 3 中，与三角形相连的棕色曲线固定 $\alpha = \beta = 1$，并在局部和全局层级将跳过阈值低估了 F 倍。这相当于 GTI 结合缩放权重填充，是 2GTI 的一种特殊情况。 F 从棕色曲线左下端的 1 变化到右上端的 0.7。随着 F减小，跳过阈值会变得非常宽松，目标文档被跳过的可能性也会降低。这样一来，检索相关性可能会提高，但检索时间也可能大幅增加。与调整 β的蓝色曲线相比，如图所示，棕色曲线需要更长的检索时间才能达到相关性峰值，而且棕色曲线通常位于蓝色曲线的右侧

。例如，在使用 uniCOIL 的 Dev集上，采用阈值低估的棕色曲线在平均延迟为 3.7 毫秒时达到最佳相关性，而采用 β 调整的蓝色曲线在平均延迟为 2.3 毫秒时达到相同的峰值，速度快 1.6 倍。

BEIR 数据集上的零样本性能。我们使用 BEIR 集合中的 13 个搜索和语义相关性数据集，评估 2GTI 的零样本排序效果和响应时间。我们训练的 SPLADE++ 模型仅使用 MS MARCO 数据，不使用任何 BEIR 数据。表 6列出了 SPLADE++上原始 MaxScore、2GTI/s-Fast（α=1， β=0.3， γ=0.05）和 GTI（α=β=1， γ=0.05）的 nDCG@10 得分。检索深度为k = 10 和 k = 1000。该表还报告了以毫秒为单位的平均检索响应时间。我们自行训练的 SPLADE++ 模型的平均 nDCG@10 得分为 0.500，接近 SPLADE++ 论文 [13]中报告的 0.507。当 k = 10 和 k = 1000 时，原始 MaxScore 的 nDCG@10 得分保持不变。

。表 6：使用 SPLADE++ 在 BEIR 数据集上的 NDCG@10 零样本相关性和以毫秒为单位的检索延迟

	原始 MaxScore		2GTI/s-Fast		GTI/s	
数据集	**nDCG**	**MRT**	**nDCG**	**MRT**	**nDCG**	**MRT**
***k*=10**						
DBpedia	0.447	99.0	**0.449**	34.5	0.306	10.6
FiQA	**0.355**	5.1	0.354	3.4	0.256	0.8
NQ	**0.551**	72.9	**0.551**	28.6	0.524	7.3
HotpotQA	**0.681**	453	**0.681**	191	0.549	46.8
NFCorpus	**0.351**	0.3	0.347	0.2	0.327	0.1
T-COVID	0.705	15.9	**0.707**	9.9	0.569	2.6
Touche-2020	**0.291**	8.7	**0.291**	3.2	0.237	1.3
ArguAna	0.446	8.8	0.448	4.0	**0.454**	4.0
C-FEVER	**0.234**	635	0.231	355	0.196	241
FEVER	**0.781**	1028	0.771	655	0.590	160
Quora	**0.817**	21.5	**0.817**	6.7	0.763	1.7
SCIDOCS	**0.155**	3.7	**0.155**	2.1	0.140	1.2
SciFact	**0.682**	3.2	0.680	2.9	0.680	1.7
平均值	**0.500**	-	0.499	2.0x	0.430	6.1x
***k*=1000**						
平均值	0.500	-	**0.501**	2.5x	0.496	2.7x

当 k = 10，对于这些 BEIR 数据集，2GTI 的 NDCG@10 得分几乎与原始 MaxScore 相同，且平均速度比 MaxScore 快 2.0 倍。在相同的索引数据上运行时，GTI 的平均 NDCG@10 得分为 0.43 MRR@10，速度也快于 2GTI；对于这些数据集，其速度平均比原始 MaxScore 快 6.1 倍。2GTI 中的两级剪枝比 GTI 更能保留相关性，这与我们在搜索 MS MARCO 段落时观察到的情况一致。

当 k = 1000 时，引导遍历算法更有可能保留相关性。2GTI 的平均相关性为 0.501 MRR@10，略高于 k = 10 ，平均速度约为原始 MaxScore 的 2.5 倍。对于在相同索引上以相同对齐方式运行的 GTI，平均 MRR@10 为 0.496，平均速度比 MaxScore 快 2.7 倍。由于在较大的 k 值下，BM25 驱动的剪枝仍能保持良好的召回率，其相关性得分接近 2GTI。

B BMW 的两级引导

两级引导也可以用于控制基于 BMW 的算法（例如 VBMW）的索引遍历，因为我们同样可以将此类算法视为执行一系列索引遍历步骤，并且可以区分其在每个遍历步骤中的索引剪枝，即全局文档间层级和局部文档内层级的剪枝。我们使用与上一小节相同的符号表示法，并假设倒排列表按照文档 ID 递增的顺序排列。我们仍然在每个搜索词的倒排列表中保留一个位置指针，以跟踪每个词 t_i 当前正在处理的文档标识符 d_{t_i}，为每个访问过的文档 d 增量累积三个得分 $Global(d)$、$Local(d)$ 和 $RankScore(d)$，并维护三个独立的按分数排序的队列 Q_{Gl}、Q_{Lo} 和 Q_{Rk}。

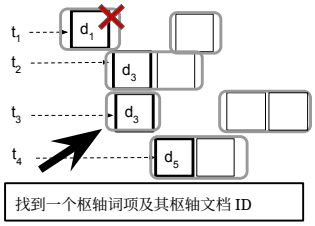

图4：BMW 中的全局剪枝

- **在文档间层级进行剪枝，并识别枢轴**。BMW [11] 会在每个遍历步骤中维护一个已排序的搜索词列表，以便 $d_{t_i} \le d_{t_{i+1}}$ 与 $1 \le i \le N-1$。对当前文档指针进行划分的枢轴位置，是满足 $\sum_{i=1}^{pivot} \alpha \sigma_B[i] +$的最小整数，称为 $pivot$ $1-$（）。这 α $\sigma_L[i] > \theta_{Gl}$. d 个不等式意味着，任何满足 $d_{t_0} \le d < d_{t_{pivot}}$的文档 ID，根据得分 $Global$（d），都不可能进入最终的前 k 个列表。然后，利用上述枢轴检测，对于 $1 \le i < pivot$，第 i个倒排列表的当前访问指针会移动到最近的块，该块包含一个大于或等于$d_{t_{pivot}}$的文档 ID。

图4展示了一个 BMW 中全局剪枝的示例，其中包含 4 个词项，每个倒排列表都维护一个指针，指向遍历步骤中当前正在访问的文档。每个倒排列表中的文档都由弯曲矩形框出，表示这些列表以块为单位存储和压缩。在图中，一个遍历步骤中的枢轴识别定位到文档 d_3，小于 d_3 的文档 ID 在该遍历步骤中将不再纳入考虑。

- **局部剪枝**。 令 d 为枢轴词项 t_{pivot}中对应的枢轴文档。在图4中，枢轴词项 $t_{pivot} = t_3$ 和 $d = d_3$。执行一个遍历过程，以检查是否可以完全或部分避免对文档 d 进行详细评分；此过程可以类似于前文所述修订后的 MaxScore 算法中的过程。由于在 BMW 中，每个倒排列表都按块的方式打包，令 $\Delta_B[x]$ 和 $\Delta_L[x]$ 分别表示包含 d的第 x个倒排列表中该块的 BM25 权重和学习得到的块最大权重；如果此列表中不存在这样的块，则它们均为 0。 $Local(d)$ 的上界可以使用块级最大权重收紧，而不是使用各词项贡献的列表级最大权重，具体如下：$\sum_{i=1}^{N} \beta \Delta_B[i] + (1-$

表7：在 MS MARCO 段落上使用 uniCOIL 的 Guided VBMW 和 MaxScore

数据集	方法	$k = 10$	$k = 20$	$k = 100$
Dev	MaxScore-2GTI	0.355†, 2.6 (14.3)	0.355†, 3.4（18.4）	0.355，5.5（26.0）
	VBMW-2GTI	0.353†, 4.3 (30.6)	0.354†, 5.2 (35.6)	0.355, 8.6 (51.6)
	VBMW-GTI	0.339, 2.4 (14.2)	0.347, 3.0 (17.2)	0.353，5.4（27.1）
DL'19	MaxScore-2GTI	0.714，1.9（12.9）	0.713, 2.3（14.2）	0.713, 4.3 (18.6)
	VBMW-2GTI	0.708，2.0（20.0）	0.708，3.7（23.2）	0.710, 6.6（33.1）
	VBMW-GTI	0.694，1.7（9.0）	0.700, 2.2 (11.7)	0.710，4.3 (17.9)
DL'20	MaxScore-2GTI	0.689，2.8（12.1）	0.689，3.3（13.0）	0.689，5.3（22.2）
	VBMW-2GTI	0.683，3.9（18.4）	0.686, 4.9（22.6）	0.686，8.4（46.8）
	VBMW-GTI	0.676, 2.3 (10.5)	0.680，2.9（13.7）	0.685, 5.3 (24.8)

表8：在不同查询类别下，使用 k =10、uniCOIL 和 MS MARCO 段落 Dev 集的性能

查询长度	≤ 3	4-5	6-7	≥ 8
# 查询 w/ SW	113	1720	2175	2030
MaxScore-2GTI	0.286，1.6（12.2）	0.376，1.7（8.9）	0.347，2.4（11.5）	0.315, 4.3 (20.9)
VBMW-2GTI	0.289，2.1（15.2）	0.373, 2.2 (12.5)	0.346, 3.4 (16.1)	0.313，8.5 (50.8)
VBMW-GTI	0.257, 1.1 (6.1)	0.346, 1.4 (6.8)	0.334，2.6（12.2）	0.307，7.4（43.9）
# 不使用 SW 的 Q	327	445	130	40
MaxScore-2GTI	0.397，1.3（9.0）	0.430，1.9（9.4）	0.439，3.7（12.1）	0.558，5.4（15.9）
VBMW-2GTI	0.399、0.9（4.0）	0.430，1.6 (6.3)	0.438, 3.6 (12.7)	0.565, 5.6 (23.3)
VBMW-GTI	0.385, 0.7 (3.4)	0.420，1.3 (5.1)	0.433, 2.6 (9.9)	0.560，4.1 (12.9)

$\beta)\Delta_L[i]$. 解压倒排列表中所需的块时，上述表达式中对应词项的块最大贡献值可以替换为文档 d的实际 BM25 权重和学习得到的权重。这样， $Local(d)$ 的上界可进一步收紧，并可在每次向下调整后直接与 θ_{Lo} 进行比较。

2GTI在VBMW上的效果评估。我们选择 uniCOIL 来研究 VBMW-2GTI 在检索 MS MARCO 开发集时的实用性。不选择 SPLADE++，因为测试查询的平均长度较长，对于此类查询，MaxScore 比 VBMW 更快。表7报告了在改变 k 时，使用 uniCOIL 进行段落检索时 VBMW-2GTI、VBMW-GTI 和 MaxScore-2GTI 的性能。每个条目的报告格式为 x， $y(z)$，其中 x 是 Dev 的 MRR@10，或 DL'19 和 DL'20 的 NDCG@10。 y 是以毫秒为单位的 MRT， z 是以毫秒为单位的 P_{99} 延迟。2GTI 使用快速设置，其中 $\alpha = 1$、 $\beta = 0.3$。对于 2GTI 和 GTI， $\gamma = 0.1$。结果表明，当 k 为 10 和 20 时，与 GTI 相比，2GTI 能够提升 VBMW 的相关性。对于 $k = 100$，相关性差异可以忽略不计。对于所有测试查询，MaxScore-2GTI 的平均速度仍然快于 VBMW-2GTI，而二者的相关性差异较小。下面我们将考察 VBMW-2GTI 是否能对部分查询发挥作用。

表8报告了这三种算法在段落开发集上的相关性和耗时，并根据查询长度以及查询是否包含停用词对查询进行了细分。也就是说，对于 uniCOIL，使用 $k = 10$、 $\alpha = 1$、 $\beta = 0.3$ 和 $\gamma = 0.1$。每个条目的报告格式与表7相同。结果表明，对于不包含停用词的短查询（$k \le 5$），VBMW-2GTI 比 MaxScore-2GTI 快得多；对于这类查询，VBMW-2GTI 的相关性优于 VBMW-GTI，同时与 MaxScore-2GTI 非常接近。上述结果表明，融合方法可以根据查询特征切换算法选择，从而取得良好效果；VBMW-2GTI 可用于一类查询。

参考文献

[1] Yang Bai、Xiaoguang Li、Gang Wang、Chao liang Zhang、Lifeng Shang、Jun Xu、Zhaowei Wang、Fangshan Wang 和 Qun Liu。2020。SparTerm：学习基于词项的稀疏表示以实现快速文本检索。arXiv abs/2010.00768（2020）。[2] Andrei Z. Broder、David Carmel、Michael Herscovici、Aya Soffer 和 Jason Zien。2003。使用两级检索过程进行高效查询评估。载于第 12 届 ACM 国际信息与知识管理会议论文集。426–434。[3] Daniel Fernando Campos、Tri Nguyen、Mir Rosenberg、Xia Song、Jianfeng Gao、Saurabh Tiwary、Rangan Majumder、Li Deng 和 Bhaskar Mitra。2016。MS MARCO：人工生成的机器阅读理解数据集。arXiv abs/1611.09268（2016）。[4] Kaushik Chakrabarti 、 Surajit Chaudhuri 和 Venkatesh Ganti。2011。压缩列表上 Top-k 处理的基于区间的剪枝。2011 年 IEEE 第 27 届国际数据工程会议（2011），709–720。[5] David R. Cheriton。2019。从 doc2query 到 docTTTTTquery。[6] coCondenser。2021。https://huggingface.co/Luyu/co-condenser-marco。（2021）。[7] Matt Crane、J. Shane Culpepper、Jimmy Lin、Joel Mackenzie 和 Andrew Trotman。2017。按文档处理与按分数处理查询评估的比较。载于第十届 ACM 网络搜索与数据挖掘国际会议论文集（英国剑桥）（WSDM '17）。ACM，纽约，美国，201–210。[8] Nick Craswell、Bhaskar Mitra、Emine Yilmaz、Daniel Fernando Campos 和 Ellen M. Voorhees。2020。TREC 2020 深度学习赛道概述。arXiv abs/2102.07662（2020）。[9] Zhuyun Dai 和 Jamie Callan。2020。用于第一阶段段落检索的上下文感知的词项加权。第 43 届 ACM 信息检索研究与发展国际会议论文集（2020）。[10] Constantinos Dimopoulos、Sergey Nepomnyachiy 和 Torsten Suel。2013。优化块最大索引的 Top-k 文档检索策略。载于 WSDM '13。[11] Shuai Ding 和 Torsten Suel。2011。使用块最大索引加速 Top-k 文档检索。载于第 34 届 ACM 信息检索研究与发展国际会议论文集。993–1002。[12] Thibault Formal、C. Lassance、Benjamin Piwowarski 和 Stéphane Clinchant。2021。SPLADE v2：用于信息检索的稀疏词法与扩展模型。arXiv abs/2109.10086（2021）。[13] Thibault Formal、Carlos Lassance、Benjamin Piwowarski 和 Stéphane Clinchant。2022。从蒸馏到硬负样本采样：提升稀疏神经信息检索模型的效果。第 45 届 ACM 信息检索研究与发展国际会议论文集（2022）。[14] Thibault Formal、Benjamin Piwowarski 和 Stéphane Clinchant。2021。SPLADE：用于第一阶段排序的稀疏词法与扩展模型。第 44 届 ACM 信息检索研究与发展国际会议论文集（2021）。[15] Luyu Gao、Zhuyun Dai 和 Jamie Callan。2021。COIL：在信息检索中结合上下文表示重新审视精确词法匹配。NAACL（2021）。[16] Luyu Gao、Zhuyun Dai、Tongfei Chen、Zhen Fan、Benjamin Van Durme 和 Jamie Callan。2021。用语义残差嵌入补充词法检索。ECIR 和 arxiv：2004 13969 [ ]。计算机科学·信息检索 Kalervo Järvelin 和 Jaana Kekäläinen。累积增益式信息检索技术评估。ACM 信息系统汇刊（TOIS）20，4（2002），422–446。[18] Karen Spärck Jones、Steve Walker 和 Stephen E. Robertson。2000。信息检索的概率模型：发展与比较实验。载于《信息处理与管理》。779–840。[19] Omar Khattab、Mohammad Hammoud 和 Tamer Elsayed。2020。兼得两全：更快、更稳健的 Top-k 文档检索。载于第 43 届 ACM 信息检索研究与发展国际会议论文集。1031–1040。[20] Carlos Lassance 和 Stéphane Clinchant。2022。SPLADE 模型的效率研究。载于第 45 届 ACM 信息检索研究与发展国际会议论文集。2220–2226。[21] Daniel Lemire 和 Leonid Boytsov。2015。通过向量化实现每秒解码数十亿个整数。Softw. Pract. Exp. 45，1（2015），1–29。[22] Hang Li、Shuai Wang、Shengyao Zhuang、Ahmed Mourad、Xueguang Ma、Jimmy Lin 和 G. Zuccon。2022。是否进行插值：伪相关反馈、稠密与稀疏检索器。SIGIR（2022）。[23] Jimmy Lin、Xueguang Ma、Sheng-Chieh Lin、Jheng-Hong Yang、Ronak Pradeep 和 Rodrigo Nogueira。2021。Pyserini：用于以稀疏与稠密表示开展可复现信息检索研究的 Python 工具包。载于第 44 届 ACM 信息检索研究与发展国际年会论文集（SIGIR 2021）。2356–2362。[24] Jimmy J. Lin 和 Xueguang Ma。2021。关于 DeepImpact、COIL 以及信息检索技术概念框架的简要札记。ArXiv abs/2106.14807（2021）。[25] Sheng-Chieh Lin、Jheng-Hong Yang 和 Jimmy J. Lin。2021。用于稠密检索的紧耦合教师知识蒸馏中的批内负例。载于 REPL4NLP。[26] Xueguang Ma、Kai Sun、Ronak Pradeep 和 Jimmy Lin。2021。稠密段落检索器的复现研究。arXiv: 2104 05740 [ ]。cs.CL Craig Macdonald、Nicola Tonellotto 和 Iadh Ounis。动态剪枝安全性对学习排序效果的影响。载于第 35 届 ACM 信息检索研究与发展国际会议论文集（美国俄勒冈州波特兰）（SIGIR '12）。计算机协会，纽约，美国，1051–1052。[28] Joel Mackenzie、J. Shane Culpepper、Roi Blanco、Matt Crane、Charles L. A. Clarke 和 Jimmy Lin。2018。早期检索中的查询驱动算法选择。载于第 11 届 ACM 网络搜索与数据挖掘国际会议论文集。396–404。[29] Joel Mackenzie、Andrew Trotman 和 Jimmy Lin。2021。学习型稀疏表示中的古怪权重，以及按分数处理查询评估的卷土重来。arXiv: 2110 11540 [ ]。cs.IR Antonio Mallia、Omar Khattab、Torsten Suel 和 Nicola Tonellotto。2021。为倒排索引学习段落影响分数。载于第 44 届 ACM 信息检索研究与发展国际会议论文集。1723–1727。[31] Antonio Mallia、Joel Mackenzie、Torsten Suel 和 Nicola Tonellotto。2022。通过引导遍历加速学习型稀疏检索。载于第 45 届 ACM 信息检索研究与发展国际年会论文集（SIGIR 2022）。1901–1905。[32] Antonio Mallia、Giuseppe Ottaviano、Elia Porciani、Nicola Tonellotto 和 Rossano Venturini。2017。采用可变大小块的快速 BlockMax WAND。载于第 40 届 ACM 信息检索研究与发展国际会议论文集。625–634。[33] Antonio Mallia、Michal Siedlaczek、Joel Mackenzie 和 Torsten Suel。2019。PISA：面向学术界的高性能索引与搜索。开源信息检索可复现性挑战论文集（2019）。[34] Antonio Mallia、Michal Siedlaczek 和 Torsten Suel。2019。索引压缩与 DAAT 查询处理方法的实验研究。载于第 41 届欧洲信息检索研究会议论文集，ECIR' 2019。353–368。[35] MiniLM-L-6-v2。2022。https://huggingface.co/cross-encoder/ms-marco-MiniLM- L-6-v2。（2022）。[36] Yifan Qiao、Yingrui Yang、Haixin Lin、Tianbo Xiong、Xiyue Wang 和 Tao Yang。2022。利用学习型稀疏表示进行文档检索的双重跳过引导。ArXiv abs/2204.11154（2022 年 4 月）。[37] Trevor Strohman 和 W. Bruce Croft。2007。主存中的高效文档检索。载于第 30 届 ACM 信息检索研究与发展国际会议论文集。175–182。[38] Nandan Thakur、Nils Reimers、Andreas Rücklé、Abhishek Srivastava 和 Iryna Gurevych。2021。BEIR：用于信息检索模型零样本评估的异构基准测试。载于第三十五届神经信息处理系统会议数据集与基准测试专题（第二轮）。https://openreview. net/forum?id=wCu6T5xFjeJ [39] Nicola Tonellotto、Craig Macdonald 和 Iadh Ounis。2013。通过选择性剪枝实现高效且有效的检索。载于第六届 ACM 网络搜索与数据挖掘国际会议论文集（WSDM '13）。ACM，63–72。[40] Nicola Tonellotto、Craig Macdonald、Iadh Ounis 等。2018。面向可扩展网络搜索的高效查询处理。《信息检索基础与趋势》12，4-5（2018），319–500。[41] Howard Turtle 和 James Flood。1995。查询评估：策略与优化。《信息处理与管理》31，6（1995），831–850。[42] Yingrui Yang、Yifan Qiao、Jinjin Shao、Xifeng Yan 和 Tao Yang。2022。面向高效关键词搜索的轻量级组合重排序与 BERT。WSDM（2022）。[43] Hamed Zamani、Mostafa Dehghani、William Bruce Croft、Erik G. Learned-Miller 和 J. Kamps。2018。从神经重排序到神经排序：为倒排索引学习稀疏表示。第 27 届 ACM 国际信息与知识管理会议论文集（2018）。