The Case for Learned Index Structures

Tim Kraska	Alex Beutel	Ed H. Chi	Jeffrey Dean	Neoklis Polyzotis	Haizhao He
MIT	Google	Google	Google	Google	Hebut

日期：2023 年 1 月 5 日

摘　　要

传统的索引模型可以分为以下 3 类：B 树（B-Trees）：可以视为一个模型，它将一个键值 Key 映射到一个有序数组中的一条记录的位置。通常用于范围查询（例如检索特定时间范围内的所有记录）。哈希映射（Hash-maps）：可以视为一个模型，它将一个键值 Key 映射到一个无序数组中的一条记录的位置。通常用于单键值查找。比特映射索引（BitMap-Index）：可以视为一个模型，用于检查一条数据记录是否存在。（例如：Bloom filters）在本文中，假设所有现存的索引结构都可以替换为其他类型的模型，包括深度学习模型，我们称之为学习索引（learned index）关键思想：一个模型可以学习排序顺序或者查询键值的结构，并且利用这类信息来高效预测记录的位置或者判断记录是否存在。文章从理论上分析了在什么情况下，学习索引的表现优于传统索引，以及设计学习索引结构时的一些主要挑战。文章初步结果显示，在几个真实数据集上，基于神经网络的学习索引在速度上要比经过缓存优化的 B 树快 70%，并且在内存方面要节省一个数量级。更重要的是，作者认为这种利用学习模型替换数据管理系统中的核心组件的想法对于未来的系统设计具有潜在的长远影响。

关键词： 学习索引, CDF 函数

1　导论

传统索引结构的缺点：没有对数据本身的分布作出任何假设，并且也没有利用现实世界中普遍存在的一些更常见模式的优势。在知道确切的数据分布的前提下，我们几乎可以对任何索引结构进行高度优化。当然，在现实世界中大部分情况下，数据不会完美服从某个已知分布，而且针对每个用例构建特定的解决方案会导致巨大的工程量。但是，作者认为机器学习可以让我们以较低的工程成本自动合成所谓“学习索引”的特殊索引结构。

本文作者探索了（包括神经网络在内的）学习索引可以用于提升、甚至替换传统索引（从 B 树到布隆过滤器）。在语义学方面，索引在很大程度上已经可以被视为学习模型了。这使得我们用其他机器学习模型对其进行替换时比预期的更容易。例如：B 树可以视为一个模型，输入为 key，预测为一个有序集合中的某条数据的位置；而布隆过滤器则可以视为一个二分类模型，预测一个集合中的某个 key 是否存在。但是，它们和真正的学习模型还存在一些微妙的但非常重要的差异，例如：一个布隆过滤器可以只有假阳例，而没有假阴例。

在性能方面，由于每个 CPU 都具有强大的 SIMD 功能，可以合理推测越来越多的设备将拥有图形处理单元（GPU）或张量处理单元（TPU），并且功能将越来越强大。因为对于神经网络使用的（并行）数学运算受限集的扩展要比通用指令集的扩展容易得多。因此，执行神经网络的

高成本在未来实际上可以忽略不计。很重要的一点是，作者并不主张使用学习索引来完全取代传统的索引结构。相反，本文主要贡献在于提出了一种建立索引的新方法，并对其进行了评估。它作为现有工作的补充，为这一领域开辟了一个新的研究方向。虽然本文专注于分析只读工作负载，但是作者也简单概述了如何将其扩展到涉及频繁写入工作负载的任务。此外，作者还简要概述如何使用相同的原则来替换数据库及其他组件的操作，包括排序和联表 (join)。这些可能会在未来引领一种全新的数据库的开发方式。

2 范围索引

范围索引结构（例如：B 树）可以视为一种模型：给定一个键（key），“预测”一个基于 key 的有序集合中某个值的位置。如图 1 中的（a）所示，B 树提供了一种从查找键（look-up keys）到存储记录（records）的有序数组内某个位置（position）的映射，并且保证该位置的那条记录对应的 key 是第一个等于或者大于查找键的 key。注意，必须对数据进行排序以允许范围请求。相同的概念也适用于二级索引，此时数据为 key-记录指针对 $< key, record_pointer >$ 的列表。

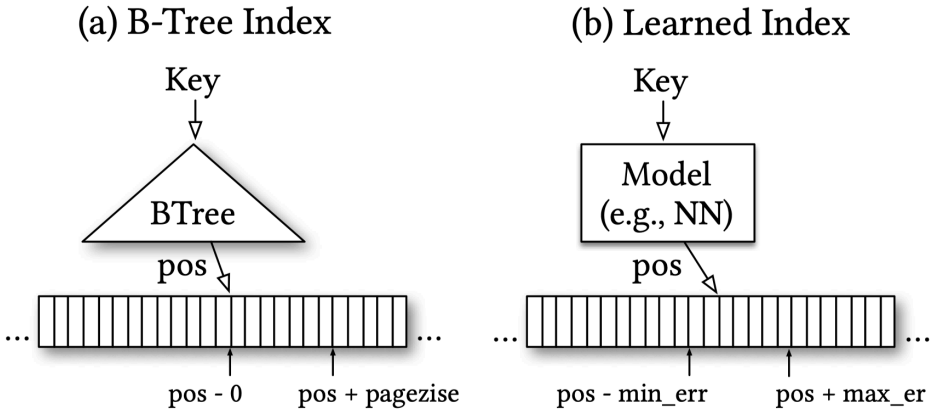

图 1: 为什么 B 树可以视为模型

出于效率原因，通常不会对已排序记录的每个单独的 key 进行索引，而是对每隔 n 条记录对应的 key 进行索引，即每个页（page）的第一个 key。这里仅仅假设固定长度记录，以及在一块连续内存区域上逻辑分页（logical paging）的情况，即一个单独的数组，而非分散在不同内存区域上的物理页。仅仅对每个 page 的第一个 key 进行索引，可以在没有明显性能损失的前提下，显著减少必须存储的 keys 的数量。因此，B 树可以视为一个回归树模型：它将一个 key 映射到一个位置的最小和最大误差之间（最小误差为 0，最大误差为一个 page 的大小），这可以保证只要这个 key 存在，那么一定可以在该范围内找到它。按照这种思路，我们可以用其他机器学习模型（包括神经网络）替代 B 树索引，只要其能够提供类似的有关最小误差和最大误差的有力保证。

实际上，为其他 ML 模型提供相同错误保证这件事要比看起来简单。首先，B 树仅为存储的 keys，而非所有可能的 keys，提供这种关于最小最大误差的有力保证。对于新数据，B 树需要进行重新平衡，或者对应于机器学习中的重新训练，来提供相同的误差保证。也就是说，对于单调模型，我们唯一要做的就是利用模型对每一个 key 进行预测，并且记下对于一个位置的最差

的过高和过低的预测，来计算最小和最大误差。其次，更重要的一点是，我们甚至不需要强的误差边界。因为为了支持范围请求，数据总是会进行排序，所以，任何误差都可以通过在预测位置附近进行局部搜索来纠正。因此，甚至可以扩展到非单调模型。所以，我们可以用任何其他类型的回归模型，例如线性回归或者神经网络，来替代 B 树。

要实现这种替换还存在一些其他挑战。例如，B 树对于插入和查找都有一个有界代价，并且能够很好地利用缓存（cache）的优势。此外，B 树还可以将 keys 映射到那些分散在非连续内存或者磁盘中的页（pages）。

同时，使用其他模型替代索引可以带来极大的好处。最主要的一点就是这种做法有潜力将时间复杂度为 $O(\log n)$ 的 B 树查找转换为一个常数时间操作 $O(1)$。机器学习（尤其是神经网络）的优势在于它们能够学习各种各样的数据分布/混合和其他数据特征和模式。显然，挑战在于平衡模型的复杂度与准确性。

2.1 我们可以接受的模型复杂度

为了更好地理解模型的复杂度，我们需要知道在和遍历 B 树所需的相同时间内，模型可以执行的操作次数，以及学习索引需要达到什么样的精度才能在查找效率上超过 B 树。

考虑一个索引 $100M$ 记录、page 大小为 100 的 B 树。我们可以将 B 树的每一个结点视为一种空间划分方式，以减小“误差”、缩小范围，从而找到数据。所以，对于一个 page 大小为 100 的 B 树，每个结点的精度增益（precision gain）都是 $1/100$，总共需要遍历 $\log_{100} N$ 个结点。因此，第一个结点将查找空间从 $100M$ 缩小到 $100M/100 = 1M$，第二个结点从 $1M$ 缩小到 $1M/100 = 10k$，以此类推，直到找到记录。现在，采用二分查找对一个 B 树的单个 page 进行遍历大约需要 50 个时钟周期，并且非常难以实现并行化。相比之下，现在 CPU 可以在每个周期执行 8-16 次 SIMD 操作。因此，只要一个模型在每 $50 * 8 = 400$ 次算术运算的精度增益超过 $1/100$，它就可以在查找速度上超过 B 树。注意，这种计算方法仍然是假定所有的 B 树 pages 都在缓存中。单个的缓存缺失（cache-miss）会消耗 50-100 个额外的时钟周期，因此可以允许更复杂一些的模型。

此外，机器学习允许在相同的时间内运行更复杂的模型，并且将计算任务从 CPU 转移到 GPU/TPU 上。尽管作者认为 GPU/TPU 是实践中采用学习索引的主要原因之一，但本文仍然将重点放在性能有限的 CPU 上，以便在不考虑硬件因素的前提下，更好地研究通过机器学习替换和增强索引的影响。

2.2 范围索引模型是 CDF 模型

如前所述，索引是一个模型，它接受一个 key 作为输入并预测某条记录的位置。虽然对于单点查询，记录的顺序并不重要，但是对于范围查询，必须根据 look-up key 对数据进行排序，使得所有数据项都在一个范围内。这导致了一个有趣的观察：对于一个通过给定 key 来预测有序数组中的某个位置的模型，它可以有效近似为累积分布函数（CDF）。我们可以对数据的 CDF 建模来预测位置：

$$p = F(\text{Key}) * N$$

其中, p 是位置估计 $F(Key)$; $F(\cdot)$ 是数据的估计的 CDF, 用来估计一个 key 小于或者等于 look-up key 的似然，即 $P(X \leq Key)$ ； N 是 key 的总数量。

这带来了一些新方向：

1. 这意味着从字面上看，索引需要学习数据分布。例如：B 树通过构建回归树来“学习”数据分布；而线性回归模型则通过最小化线性函数的（平方）误差来学习数据分布。
2. 估计数据集的分布是一个被广泛研究过的问题，学习索引可以从之前数十年的研究中受益。
3. 学习 CDF 在优化其他类型的索引结构和潜在算法时，也有着非常重要的作用。
4. 已有大量关于理论 CDF 和经验 CDF 近似的研究，这为理解学习索引的好处提供了理论支撑。

2.3 朴素学习索引（Naive Learned Index）

为了更好地理解用学习索引取代传统 B 树有哪些技术要求，我们使用了 $200M$ 的 Web 服务器日志记录，目标是使用 Tensorflow 在时间戳上建立一个二级索引：

- 架构：2 层的全连接神经网络，每层 32 个神经元
- 激活函数：ReLU
- 输入特征：时间戳
- 输出标签：有序数组中的位置

之后，使用 Tensorflow 和 Python 作为前端，随机选择一个 key，测量查找时间（运行多次取平均值，除去开始的几次）。在这种情况下，实现了每秒大约 1250 次预测，即，在不包括搜索时间（从预测位置到真实位置所花的查找时间）的情况下，使用 Tensorflow 执行模型需要大约 80,000 纳秒（ns）。相比之下，B 树遍历同样数据只需要大约 300 ns，而二分查找则需要大约 900 ns。其原因是多方面的：

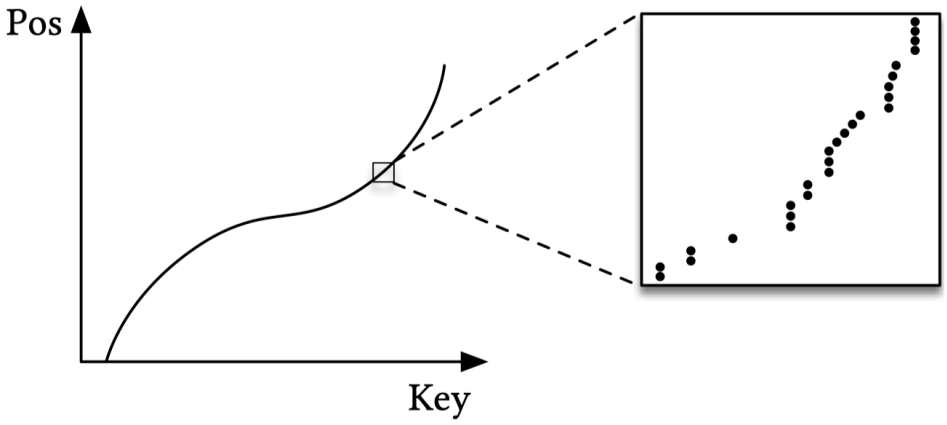

图 2: 范围索引是 CDF 模型

1. Tensorflow 主要是用来运行大型模型的，而非小模型，因此会产生明显的调用开销，尤其是使用 Python 作为前端时。
2. B 树，或者一般意义上的决策树，使用少量的操作就很容易过拟合数据，因为它们使用简单的 if 语句递归地分割空间。相比之下，其他模型可以明显更有效地近似 CDF 的总体形

状，但在单个数据实例级别的精确定位上存在一些问题。从图 2 中可以看到，宏观上，CDF 函数看起来非常平滑和规律。但是，如果放大到单条记录的层面，就能看到越来越多的不规律之处; 这是一个众所周知的统计效应。因此，像神经网络、多项式回归等模型可能会以较高的 CPU 和空间效率从整个数据集缩小到仅包含数千条记录的范围，但是，单个神经网络通常需要更多的空间和 CPU 时间来完成“最后一公里”：将误差从数千条缩小到数百条的范围。

3. B 树具有极高的缓存效率和操作效率，因为它们总是将顶层结点保留在缓存中，并且在需要时访问其他 pages。相比之下，标准的神经网络在计算预测时需要知道所有的权重，这会带来很大的乘法运算开销。

3 RM 索引（Recursive Model Index）

为了克服这些挑战，并探索模型作为索引替代或者优化的潜力，作者开发了学习索引框架（LIF）、递归模型索引（RMI）和基于标准误的搜索策略。出于简洁性和灵活性的考虑，作者将主要关注简单的全连接神经网络，但是，作者也相信其他类型的模型可能也会带来一些额外的好处。

3.1 学习索引框架（LIF）

LIF 可以被视为一个索引合成系统；给定一个索引规范，LIF 将生成不同的索引配置，并且对其进行优化和自动化测试。尽管 LIF 可以学习简单模型（例如：线性回归模型），但其依赖的 Tensorflow 主要是面向更加复杂模型的（例如：神经网络）。但是，它永远不会使用 Tensorflow 进行推断。相反，给定一个训练好的 Tensorflow 模型，LIF 会自动从模型中提取所有权重，并根据模型规范在 C++ 中生成高效的索引结构。我们的代码生成专注于小型模型，并且移除了 Tensorflow 在管理大模型所需的所有不必要开销。我们参考了 [25] 的思路，其展示了如何避免在 sprak 运行时的不必要开销。结果，我们可以在 30 ns 的数量级上执行简单模型。

然而，必须指出的是 LIF 仍然是一个实验性的框架，并且被工具化用于快速评估不同的索引配置（例如：机器学习模型、page 大小、搜索策略等），这引入了以加法计数器、虚拟函数调用等形式的额外开销。另外，除了编译器完成向量化之外，我们并没有使用特殊的 SIMD 指令集。尽管这些低效的做法并不影响我们的评估因为我们在框架中确保了公平的比较方式，但是在生产设定下或者将其与其他实现方式下的报告中的性能数字进行对比时，则必须对这种低效性进行考虑/避免。

3.2 递归模型索引

正如在 2.3 中提到的，构建一个替代 B 树的学习模型的关键挑战在于最后一英里搜索的准确性。例如，利用单个模型将预测误差从 $100M$ 条记录减小到几百条的数量级通常是非常困难的。同时，将误差从 $100M$ 减小到 $10k$ 则要简单得多（例如，通过利用一个模型来替换 B 树中的前两层来达到 $100 * 100 = 10000$ 的精度增益），哪怕是采用非常简单的模型。类似地，将误差从 $10k$ 减小到 100 同样是一个更加简单的问题，因为模型只需要关注整个数据集的一个子集。

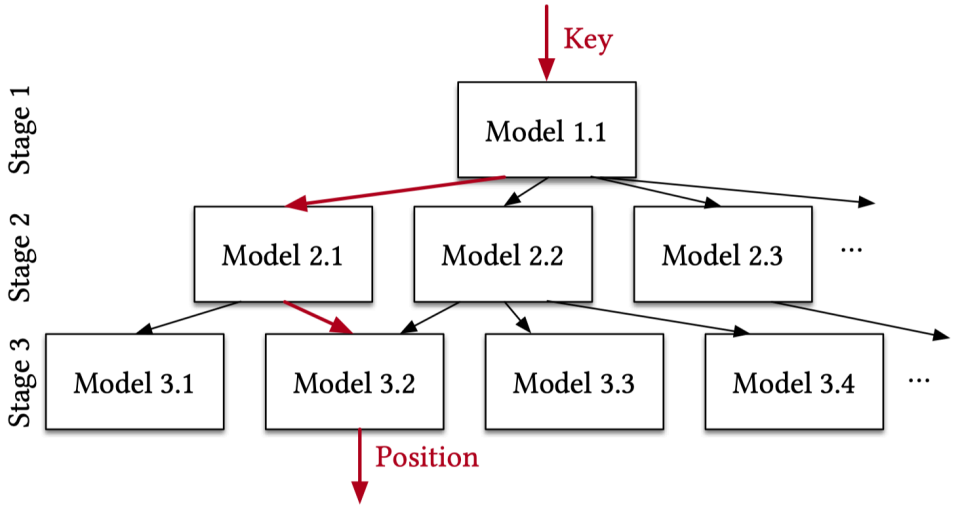

图 3: 阶段模型

基于上述观察，作者提出了图 3 所示的递归回归模型。也就是说，作者构建了一个分级模型，在每个阶段，模型将 key 作为输入，并基于它选择下一个模型，直到最终阶段预测出位置。更为正式的表述是，对于模型 $f(x)$，其中 x 是 key，$y \in [0, N)$ 位置，假设在阶段 l 有 M_l 个模型。我们在阶段 0 训练模型 $f_0(x) \approx y$。因此，阶段 l 的模型 k 可以表示为 f_l^k，其训练采用的损失函数为：

$$L_\ell = \sum_{(x,y)} (f_\ell^{(\lfloor M_\ell f_{\ell-1}(x)/N \rfloor)}(x) - y)^2 \qquad L_0 = \sum_{(x,y)} (f_0(x) - y)^2$$

注意，这里表示 $f_{l-1}(x)$ 将递归执行 $f_{\ell-1}(x) = f_{\ell-1}^{(\lfloor M_{\ell-1} f_{\ell-2}(x)/N \rfloor)}(x)$。总的来说，我们利用损失函数 L_ℓ 对每个阶段进行迭代训练从而构建出完整的模型。一种理解这些不同模型的方法是，对于给定的 key，每个模型都给出了一个包含特定误差的关于记录位置的预测，并且预测结果被用来选择下一个模型，而下一个模型需要对一个特定范围的 key 空间给出一个具有更小误差的更好的预测。但是，递归模型索引不需要是树，如图 3 所示，某个阶段的不同模型可能会选择下一个阶段的同一个模型。并且，每个模型并不需要像 B 树一样包含相同数量的记录（例如：一个 page 大小为 100 的 B 树将包含 100 条或者更少的数据）。最后，取决于所使用的模型，不同阶段之间的预测不一定需要解释为位置估计，而是应当被视为挑选一个对于特定的 keys 有更好了解的专家。

该模型架构具有诸多好处：

1. 它将模型大小与复杂度从执行开销中分离出来。
2. 它利用了其易于学习数据分布的总体形状的事实。
3. 它有效地将空间划分成了更小的子范围（就像 B 树一样），这使得以更少的操作达到所需的“最后一英里”准确性更容易。
4. 在各阶段之间不需要搜索过程。例如，模型 1.1 的输出被直接用于选择下个阶段的模型。这不仅减少了管理数据结构的指令数量，而且还允许将整个索引表示为稀疏矩阵乘法以扩展到 TPU/GPU 上。

3.3 混合索引

递归模型索引的另一个优势在于我们可以构建混合模型。例如：考虑到在顶层一个小的 ReLU 神经网络可能是最佳选择因为它们通常可以学习各种复杂的数据分布，而模型分层结构中的底层模型可能是数以千计的简单线性回归模型，因为它们的空间和时间成本都较低。并且，如果数据非常难以学习，我们甚至可以在底层阶段使用传统的 B 树。

本文主要关注两种模型：

- 简单神经网络：具有 0-2 个全连接隐藏层，ReLU 作为激活函数，每层宽度多达 32 个神经元。
- B 树（即决策树）

Algorithm 1: Hybrid End-To-End Training

```
   Input: int threshold, int stages[], NN_complexity
   Data: record data[], Model index[][]
   Result: trained index
1  M = stages.size;
2  tmp_records[][];
3  tmp_records[1][1] = all_data;
4  for i ← 1 to M do
5      for j ← 1 to stages[i] do
6          index[i][j] = new NN trained on tmp_records[i][j];
7          if i < M then
8              for r ∈ tmp_records[i][j] do
9                  p = index[i][j](r.key) / stages[i + 1];
10                 tmp_records[i + 1][p].add(r);
11 for j ← 1 to index[M].size do
12     index[M][j].calc_err(tmp_records[M][j]);
13     if index[M][j].max_abs_err > threshold then
14         index[M][j] = new B-Tree trained on tmp_records[M][j];
15 return index;
```

注意，一个具有 0 个隐藏层的神经网络等价于线性回归。给定一个索引配置，它是一个 size 的数组，指定了阶段数量和每个阶段的模型数量。混合索引的端到端训练按照算法 1 完成：

从整个数据集开始（第 3 行），它训练第一个顶层结点模型。基于这个顶层结点模型的预测结果，它随后选择下个阶段的模型（第 9-10 行），并且添加所有落在该模型的 keys（第 10 行）。最后，在混合索引的情况下，如果绝对最小/最大误差超过了预定义的阈值（第 11-14 行），则通过用 B 树替换 NN 模型来优化索引。

请注意，我们在最后阶段为每个模型存储标准误差和最小误差和最大误差。这样做的好处是，我们可以根据每个 key 所使用的模型单独地限制搜索空间。目前，我们通过简单的网格搜索来调整模型的各种参数（即，阶段数量、每个模型的隐藏层数等等）。然而，还有很多潜在的优化技术（从 ML 自动调参到采样技术等）可以加速训练过程。

注意，混合模型允许我们将学习索引中最差的样例性能限定为和 B 树性能相当。即，对于非常难以学习的数据分布，所有模型都会被自动替换为 B 树，使得其实际上成为一个完整的 B 树。

3.4 搜索策略和单调性

范围索引通常实现了一个上界 $upper\_bound(key)$ / 下界 $lower\_bound(key)$ 接口，用来查找有序数组内等于或者高于/低于 look-up key 的第一个 key 的位置，这样可以有效支持范围请求。

对于学习范围索引，我们需要基于预测来找到第一个高于/低于 look-up key 的 key。尽管经历了各种尝试，但是研究表明，对于小数据量而言，二分查找或者扫描通常都是查找一个有序数组内的 key 的最快策略。然而，学习索引在这里还有一个优势：模型实际上预测的是 key 的位置，而非范围（page）。这里，我们讨论两种利用了这一信息的简单搜索策略：

模型偏置搜索（Model Biased Search）：我们默认的搜索策略，和传统二分查找的唯一区别是将模型的预测值设为第一个中点（middle point）。

偏置四分查找（Biased Quaternary Search）：四分查找将一个分裂点替换为三个分裂点，这种情况下，如果数据不在缓存中，则硬件可以一次预取所有三个数据点以获得更好的性能。在我们的实现中，我们将四分查找的初始三个中间点定义为 $pos - \sigma, pos, pos + \sigma$。也就是说，我们猜测大多数预测都是准确的，并且首先将注意力集中在位置估计上，然后继续进行传统的四元搜索。

对于我们所有的实验，我们使用最小误差和最大误差作为所有技术的搜索范围。也就是说，我们为每个 key 执行 RMI 模型，并存储每个最后阶段模型的最坏的过度预测和不足预测。尽管此技术保证找到所有现有密钥，但是对于不存在的密钥，如果 RMI 模型不是单调的，则它可能会返回错误的上界或下界。为了克服这个问题，一种选择是强制我们的 RMI 模型是单调的，正如在机器学习中已经研究的那样【41，71】。

另外，对于非单调模型，我们可以自动调整搜索区域。也就是说，如果找到的上界（下界）key 位于由最小和最大误差定义的搜索区域的边界上，则我们将逐步调整搜索区域。然而，另一种可能性是使用指数搜索技术。假设误差呈正态分布，则这些技术平均而言应与替代搜索策略一样好用，而无需存储任何最小和最大误差。

3.5 索引字符串

我们主要专注于对实值 key 进行索引，但是许多数据库都依赖于索引字符串，幸运的是，已有大量的机器学习研究都专注于对字符串建模。和之前一样，我们需要设计一个高效而富有表现力的字符串模型。做好字符串处理会带来许多独特的挑战。

第一个设计考虑是如何将字符串转换为模型特征，通常称为 token 化。出于简单和效率的考虑，我们将一个长度为 n 的字符串视为一个特征向量 $\mathbf{x} \in \mathbb{R}^n$，其中 x_i 是 ASCII 码的十进制值（或者 Unicode 的十进制值，具体取决于字符串）。并且，如果所有的输入的大小都相等的情况下，大部分的机器学习模型的效率更高。因此，我们将设置一个最大输入长度 N。由于数据是按照字典顺序排序的，我们可以在 token 化之前将 keys 截断到长度 N 。对于长度为 $n < N$ 的字符串，我们设置 $x_i = 0$，对于 $i > n$。

考虑到效率，我们通常采取和实值输入类似的建模方法。我们学习了一个相对较小的前馈神经网络的层次结构。其中一个区别是输入不再是一个单独的实值 x，而是一个向量 x。随着输入长度 N 的增加，线性模型 $w \cdot x + b$ 的乘法和加法操作数也呈线性增长。即使是只包含单个宽度为 h 的隐藏层的前馈神经网络，其乘法和加法操作数也会扩大 $O(hN)$。

最终，我们相信未来会有大量研究可以优化对于字符串 keys 的学习索引。例如，我们可以很容易想到其他的 token 化算法。在自然语言处理领域，已有大量关于字符串 token 化的研究可以将字符串拆分成更加有用的字段以用于机器学习模型，例如，机器翻译中的 wordpieces 技术

【70】。并且，我们还可以将后缀树（suffix-trees）和学习索引结合，或者探索更加复杂的模型层次架构（例如，递归或者卷积神经网络）。

3.6 训练

尽管训练（即加载）时间并非本文的关注点，但是必须指出，我们的所有模型，包括浅层神经网络或者甚至线性/多变量回归模型，训练过程都相当快。考虑到简单的神经网络可以利用随机梯度下降进行高效训练，并且在随机数据上可以在一到几次传递内收敛，而对于线性多变量模型（例如：0 层的神经网络）存在一个闭合解，并且在排序数据上可以在单词传递内完成训练。因此，用 $200M$ 记录训练一个简单的 RMI 索引可以在几秒内完成（当然，具体还取决于超参数的自动调整次数）；神经网络根据模型复杂度在每个模型上的训练时间约数分钟。并且请注意，在所有数据集上训练顶层模型通常是不必要的，因为这些模型的收敛时间甚至经常少于对整个随机数据进行一次扫描。这部分是因为我们使用的是简单模型，并且不太关注精度的最后几位，因为它对索引性能影响很小。最后，关于改进学习时间的研究【27，72】也适用于我们的情况，我们期望未来会有很多研究关注这个方向。

3.7 结果

我们在几个真实数据集和人造数据集上，对学习范围索引在空间和速度方面，同其他读取优化索引结构进行了对比评估。

3.7.1 整数数据集

作为第一个实验，我们在三个不同的整数数据集上，对一个基于 2 阶段 RMI 模型的学习索引在不同的第二阶段大小（$10k$, $50k$,$100k$ 和 $200k$）下，和具有不同 page 大小的读取优化 B 树进行了对比。这三个数据集中有两个来自真实世界的数据：（1）Web 日志和（2）地图，以及最后一个（3）人工合成数据集（对数正态）。其中，Web 日志包含 $200M$ 的一个主流大学网站数年来的每条请求的日志记录。这个数据集对于学习索引来说几乎可以说是最坏情况了，因为它包含了由于班级计划、周末、假期、午餐时间、部门活动、学期假期等因素导致的非常复杂的时间模式，而这些都非常难以学习。对于地图数据集，我们索引了全球约 $200M$ 用户维护特征（例如：道路、博物馆、咖啡店）的经度。不出所料，位置的经度相对线性，并且和 Web 日志数据集相比，不规律性更小。最后，为了测试索引在重尾分布上的表现，我们生成了一个 $190M$ 的唯一值组成的人工数据集，其中每个值都采样自一个 $\mu = 0, \sigma = 2$ 的对数正态分布。这些值被放大为高达 $1B$ 的整数。该数据显然是高度非线性的，这使得神经网络更难学习到 CDF。对于所有的 B 树实验，我们使用 64 位的 keys 和 64 位的有效负载/值。

作为我们的 baseline，我们使用了类似于 stx::btree 的生产质量 B 树实现，但具有进一步的缓存行优化、密集页面（即填充系数为 100%）和非常有竞争力的性能。为了调整 2 阶段学习索引，我们在神经网络上使用了简单的网格搜索，该网络具有 0 到 2 个隐藏层，层宽度范围从 4 到 32 个节点。总的来说，我们发现第一阶段采用简单（0 个隐藏层）到半复杂（2 个隐藏层以及 8 或 16 层宽）模型的效果最好。对于第二阶段，简单的线性模型具有最佳性能。这不足为奇，因

为对于最后一英里，执行复杂模型通常不值得，并且线性模型可以以最优方式学习。

学习索引 vs B 树表现

		Map Data			Web Data			Log-Normal Data		
Type	**Config**	Size (MB)	Lookup (ns)	Model (ns)	Size (MB)	Lookup (ns)	Model (ns)	Size (MB)	Lookup (ns)	Model (ns)
Btree	page size: 32	52.45 (4.00x)	274 (0.97x)	198 (72.3%)	51.93 (4.00x)	276 (0.94x)	201 (72.7%)	49.83 (4.00x)	274 (0.96x)	198 (72.1%)
	page size: 64	26.23 (2.00x)	277 (0.96x)	172 (62.0%)	25.97 (2.00x)	274 (0.95x)	171 (62.4%)	24.92 (2.00x)	274 (0.96x)	169 (61.7%)
	page size: 128	13.11 (1.00x)	265 (1.00x)	134 (50.8%)	12.98 (1.00x)	260 (1.00x)	132 (50.8%)	12.46 (1.00x)	263 (1.00x)	131 (50.0%)
	page size: 256	6.56 (0.50x)	267 (0.99x)	114 (42.7%)	6.49 (0.50x)	266 (0.98x)	114 (42.9%)	6.23 (0.50x)	271 (0.97x)	117 (43.2%)
	page size: 512	3.28 (0.25x)	286 (0.93x)	101 (35.3%)	3.25 (0.25x)	291 (0.89x)	100 (34.3%)	3.11 (0.25x)	293 (0.90x)	101 (34.5%)
Learned Index	2nd stage models: 10k	0.15 (0.01x)	98 (2.70x)	31 (31.6%)	0.15 (0.01x)	222 (1.17x)	29 (13.1%)	0.15 (0.01x)	178 (1.47x)	26 (14.6%)
	2nd stage models: 50k	0.76 (0.06x)	85 (3.11x)	39 (45.9%)	0.76 (0.06x)	162 (1.60x)	36 (22.2%)	0.76 (0.06x)	162 (1.62x)	35 (21.6%)
	2nd stage models: 100k	1.53 (0.12x)	82 (3.21x)	41 (50.2%)	1.53 (0.12x)	144 (1.81x)	39 (26.9%)	1.53 (0.12x)	152 (1.73x)	36 (23.7%)
	2nd stage models: 200k	3.05 (0.23x)	86 (3.08x)	50 (58.1%)	3.05 (0.24x)	126 (2.07x)	41 (32.5%)	3.05 (0.24x)	146 (1.79x)	40 (27.6%)

图 4: 学习索引 vs B 树

主要结果如图 4 所示。请注意，B 树的 page 大小表明了每个 page 的 keys 的数量，而不是字节大小（实际上更大）。在主要指标中，数据大小单位为 MB，总查找时间单位为纳秒，模型执行时间（无论是 B 树的遍历还是机器学习模型）单位也为纳秒，括号内是对比总时间的百分比。此外，在 size 和 lookup 两列，我们还以一个 page 大小为 128 的 B 树为基准，对比了加速和空间节省两个指标（括号中的数据）。我们选择一个 page 大小为 128 的 B 树作为固定参照点，是因为它提供了所有 B 树配置中的最佳 lookup 表现（请注意，通过完全不使用索引，总是可以很容易节省空间）。在 lookup 和 size 两列中的颜色编码表明了该索引相对参照点有多快/慢（更大/更小）。

可以看到，学习索引超过了几乎所有配置的 B 树索引，查找时间要快上 1.5 到 3 倍，并且空间方面要节省两个数量级。当然，通过解压缩，B 树可以在 CPU 时间上进一步压缩。但是，这些优化大多数都是正交的，并且同样（如果不是更加）适用于神经网络。例如，神经网络的压缩可以通过使用 4 位或 8 位整数而不是 32 位或 64 位浮点值来表示模型参数（此过程称为量子化）。这种压缩级别可以为学习索引带来额外增益。

毫不奇怪，第二阶段的大小会对索引大小和查找性能产生重大影响。就第 2.1 节中的分析而言，在第二阶段使用 10,000 个或更多模型尤其令人印象深刻，因为这表明我们的第一阶段模型可以比 B 树中的单个节点产生更大的精度跳跃。最后，对于这些数据集，我们并没有给出混合模型或者除了二分查找之外的其他搜索技术，因为它们没有带来明显的好处。

学习索引 vs 替代的 Baselines 除了从细节方面将学习索引和读取优化 B 树进行评估比较之外，我们还将学习索引和其他替代的 baselines 进行了对比，包括第三方实现。接下来，我们讨论一些替代 baselines，并且在适当情况下将其与学习索引进行对比：

直方图： B 树近似了数据分布的 CDF。一个明显的问题是直方图是否可以被当做一个 CDF 模型使用。原则上来说，答案是可以，但是为了保证数据的快速访问，直方图必须是 CDF 的一个低误差近似。通常这要求划分区间的数量很多，这将导致直方图自身的搜索成本过高。如果划分区间具有不同的区间边界以有效地处理数据倾斜，使得只有少数几个划分区间内的样本为空或太满，则尤其如此。解决此问题的一个明显办法是生成一个 B 树，因此我们将不再对直方图进行讨论。

查找表： 一个替代 B 树的简单方案是（分层）查找表。查找表通常具有固定的大小和结构（例如：64 个 slots，其中每个 slot 都指向另外 64 个 slots 等等）。查找表的优点在于，由于其固定大小，可以使用 AVX 指令对其进行高度优化。我们与一个三阶段查找表进行了对比，该表是

通过将每个第 64 个 key 放入一个包含 padding 的数组中而构成的，以使其成为 64 的倍数。然后我们在不包含 padding 的情况下，对该数组再重复一次该过程，这样一共创建两个数组。要查找一个 key 时，我们在顶层的表上使用二分查找，然后对第二张表和数据本身使用 AVX 优化的无分支扫描【14】。与其他替代方案相比（例如，使用顶层扫描或在第二个数组或数据上进行二分查找），该配置可以获得最快的查找时间。

FAST： FAST【44】是经过高度 SIMD 优化的数据结构。我们使用【47】中的代码作为对比。但是，应注意的是，FAST 始终要求以 2 的幂分配内存以使用无分支 SIMD 指令，这可能导致索引大得多。

固定大小的 B 树和插值搜索：最后，根据最近的博客文章【1】的建议，我们创建了具有插值搜索的固定高度的 B 树。设置 B 树的高度，使树的总大小为 $1.5MB$ ，类似于我们学习的模型。

没有额外开销的学习索引：对于我们的学习索引，我们使用了两阶段的 RMI 索引，顶层是多变量线性回归模型，底层是简单线性模型。对于顶层模型，我们使用简单的自动特征工程，自动创建和筛选 $key, \log(key), key^2$ 等形式的特征。多变量线性回归是一种替代神经网络的有趣方法，因为它特别适合仅用少量操作来拟合非线性模式。并且，为了确保比较的公平性，我们在基准框架之外实现了该学习索引。

	Lookup Table w/ AVX search	FAST	Fixe-Size Btree w/ interpol. search	Multivariate Learned Index
Time	199 ns	189 ns	280 ns	105 ns
Size	16.3 MB	1024 MB	1.5 MB	1.5 MB

图 5: 替代的 Baselines

为了进行比较，我们使用了带有 8 字节指针有效载荷的对数正态数据。结果如图 5 所示。可以看到，在公平条件下的数据集上，学习索引可提供最佳的整体性能，同时节省大量内存。应该注意的是，由于对齐要求，FAST 索引很大。

尽管结果令人鼓舞，但我们绝不认为学习索引在大小或速度方面始终是最佳选择。相反，学习的索引提供了一种思考索引的新方法，还需要进行更多的研究才能充分理解其含义。

3.7.2 字符串数据集

我们还在一个 $10M$ 的大型 Web 索引的非连续文档 id 上创建了一个二级索引，它被用于 Google 的一项真实产品的一部分中，以测试学习索引在字符串上的表现。基于字符串的文档 id 数据集的结果见图 6，其中还包含混合模型。此外，表中还包含了最佳模型，它是一个采用四分查找的非混合 RMI 模型索引，称为 “Learned QS”（表中最后一行）。所有的 RMI 模型在第二阶段都使用了 10,000 个模型，对于混合索引，我们使用 128 和 64 这两个阈值，作为模型在被 B 树替换之前的最大容许绝对误差。

可以看到，针对字符串，学习索引相对于 B 树的加速效果并不是那么明显。部分原因是模型执行成本相对较高，这点可以通过引入 GPU/TPU 解决。此外，搜索字符串的成本要高得多，因此更高的精度通常会有所回报；这也是采用混合索引（通过使用 B 树来替换性能不佳的模型）

	Config	Size(MB)	Lookup (ns)	Model (ns)
Btree	page size: 32	13.11 (4.00x)	1247 (1.03x)	643 (52%)
	page size: 64	6.56 (2.00x)	1280 (1.01x)	500 (39%)
	page size: 128	3.28 (1.00x)	1288 (1.00x)	377 (29%)
	page size: 256	1.64 (0.50x)	1398 (0.92x)	330 (24%)
Learned Index	1 hidden layer	1.22 (0.37x)	1605 (0.80x)	503 (31%)
	2 hidden layers	2.26 (0.69x)	1660 (0.78x)	598 (36%)
Hybrid Index	t=128, 1 hidden layer	1.67 (0.51x)	1397 (0.92x)	472 (34%)
	t=128, 2 hidden layers	2.33 (0.71x)	1620 (0.80x)	591 (36%)
	t= 64, 1 hidden layer	2.50 (0.76x)	1220 (1.06x)	440 (36%)
	t= 64, 2 hidden layers	2.79 (0.85x)	1447 (0.89x)	556 (38%)
Learned QS	1 hidden layer	1.22 (0.37x)	1155 (1.12x)	496 (43%)

图 6: 字符串数据：学习索引 vs B 树

有助于提高性能的原因。

由于搜索成本，不同的搜索策略会产生更大的差异。例如，具有 1 个隐藏层和偏置二分查找的神经网络的搜索时间为 $1102ns$，如图 6 所示。相比之下，具有相同模型的偏置四分查找仅需 $658ns$，这是一个显着的提升。偏置查找和四分查找之所以表现更好，是因为它们考虑了模型误差。

4 结论和未来工作

我们已经表明，通过利用被索引的数据的分布，学习的索引可以提供显著的好处。这为许多有趣的研究问题打开了大门。其他 ML 模型：虽然我们的重点是线性模型和混合专家的神经网络，但还有许多其他 ML 模型类型以及将它们与传统数据结构结合的方法，值得探索。

多维索引：可以说，学习索引思想最令人兴奋的研究方向是将其扩展到多维索引。模型，尤其是神经网络，非常擅长捕捉复杂的高维关系。理想情况下，该模型能够估计由任意属性组合过滤的所有记录的位置。

除了索引：学习过的算法可能令人惊讶，CDF 模型还可以加速排序和联接，而不仅仅是索引。例如，加速排序的基本思想是使用现有的 CDF 模型 F 将记录大致按排序顺序排列，然后校正几乎完全排序的数据，例如使用插入排序。

最后，正如本文多次提到的，GPU/TPU 将使学习索引的思想更加有价值。同时，GPU/TPU 也有其自身的挑战，最重要的是高调用延迟。虽然可以合理地假设，由于前面所示的异常压缩比，所有学习到的索引可能都适合 GPU/TPU，但在它们上调用任何操作仍然需要 2-3 微秒。同时，机器学习加速器与 CPU 的集成变得越来越好 [4,6]，使用批处理请求等技术，调用的成本可以摊销，因此我们不认为调用延迟是一个真正的障碍。

总之，我们已经证明了机器学习的模型有潜力提供比最先进的指数更大的好处，我们相信这是未来研究的一个富有成效的方向。