利用学习型索引提升嵌入式传感器设备上的时间序列索引性能

David Ding、Ivan Carvalho 和 Ramon Lawrence加拿
大不列颠哥伦比亚大学，基洛纳，不列颠哥伦比亚省，加拿大

关键词: 学习型索引、时间序列、数据库、传感器网络、嵌入式

摘要: 由于内存和 CPU 资源极其有限，在嵌入式传感器和物联网设备上高效查询数据颇具挑战。随着采集的数据量不断增加，在数据采集所在的边缘设备上处理、筛选和操作数据，对于提高效率并减少网络传输至关重要。现有的嵌入式索引结构无法适应数据分布及其特征。本文展示了如何应用能够生成紧凑数据摘要的学习型索引，从而显著提升查询性能和可预测性。基于线性近似的学习型索引可将查询输入/输出减少 50% 至 90%，并使查询吞吐量提高 2 至 5 倍，同时仅需数千字节的随机存取存储器。在多种时间序列数据集上的实验结果表明，学习型索引具有显著优势，其性能远超最先进的索引算法。

arXiv:2302.03085v1 [cs.DB] 6 Feb 2023

1 引言

嵌入式系统和物联网设备会收集海量数据，用于环境、工业和监测应用。这些时间序列数据通常会通过网络发送到云服务器进行处理。如今，人们越来越关注在采集数据的边缘设备上处理数据，以减少网络传输和功耗。

云服务器上有多种用于索引时间序列数据的系统（Wang 等人，2020；Yang 等人，2019）。这些系统无法适应嵌入式传感器设备内存有限、CPU 处理能力较低的情况。Microhash（Zeinalipour-Yazti等，2005）和时间序列顺序二进制索引（SBITS）（Fazackerley 等人，2021）等专用嵌入式索引，在内存占用极低的情况下展现出良好的索引性能。SBITS 支持按时间戳和数据值查询。按时间戳查询使用线性插值搜索，其性能优于以往嵌入式索引方法采用的二分搜索。

学习型索引是一种通过计算数据分布模型来高效预测记录位置的技术。已采用多种技术，包括 RadixSpline（Kipf等人，2020年）和分段几何模型（PGM）（Ferragina 和 Vinciguerra，2020）。这些索引在服务器应用中优于传统的基于树的索引。此前尚无针对嵌入式索引改造学习型索引的相关工作。

本研究的贡献是将学习型索引改造为适用于传感器设备采集和处理的时间序列数据。我们实现了基于 RadixSpline 和分段几何模型的学习型索引技术，并在实际传感器设备上使用真实世界数据集进行了评估。实验结果表明，与 SBITS 使用的二分搜索和线性插值相比，时间序列查询性能提升了两到五倍。只需少量索引内存，查询性能便能显著提升，这表明学习型索引对传感器领域具有重要价值。

2 背景

闪存上的索引具有独特特性，因此人们开发了许多索引算法来优化性能（Fevgas 等人，2020）。传感器设备上的索引通常基于闪存，除了服务器端索引所需的优化之外，还需要进行额外优化。数据结构必须适应闪存独特的性能特征，例如不同的读写时间以及先擦除后写入约束（即不能覆盖或进行原位写入）。常见的优化方法是尽量减少写入，并使用顺序写入而非随机写入。

传感器索引的一个关键考量是资源有限。设备拥有 4 KB 至 32 KB 的 SRAM 内存，处理器运行频率为

16 至 128 兆赫兹，闪存由 SD 卡或裸闪存芯片构成。索引结构必须尽量减少静态随机存取存储器的使用，才具有实用性。

面向嵌入式设备的数据处理系统，例如 Antelope（Tsiftes 和 Dunkels，2011）和 LittleD（Douglas 和 Lawrence，2014），支持设备端数据处理。通过为时间序列数据建立索引，可以提升查询性能。Antelope（Tsiftes 和 Dunkels，2011）支持按时间戳排序的内联索引。通过二分搜索，按时间戳查找记录。MicroHash（Zeinalipour-Yazti 等，2005）也将数据存储为排序后的时间序列数据文件，并利用二分搜索执行时间戳查询。MicroHash 使用范围分区，同时为数据值建立索引，以支持按数据值查询。PBFilter（Yin 和 Pucheral，2012）使用布隆过滤器汇总数据以建立索引。通过顺序写入数据文件和索引文件，PBFilter 的性能优于树和哈希技术。

SBITS（Fazackerley 等人，2021）是一种支持基于时间戳和数据值进行查询的顺序索引结构。时间戳查询通过线性插值高效完成。该方法根据时间戳以及对已存储数据记录的线性近似来预测记录的位置。鉴于传感器数据集通常以固定间隔采集，使用线性插值预测位置非常有效，且优于二分搜索。由于数据页按时间戳排序，因此只需根据起始时间戳、查询时间戳以及记录写入数据文件的速率（取决于采样率和每个数据块中的记录数）计算偏移量，即可通过 $O(1)$ 次读取检索任意时间戳的数据。通过为每个时段存储线性近似，可以扩展线性插值算法，使其支持多种不同的传感器采样率和时间段，但当前实现尚未支持此功能。SBITS 还通过使用可由用户自定义的位图索引，支持高效的数据值查询。先前的实验（Ould-Khessal 等，2022）表明，SBITS 的性能优于 MicroHash 和其他嵌入式索引技术。

鉴于传感器时间序列数据具有相对一致性，使用线性插值预测位置是有效的。然而，在真实世界数据集中，有时数据并不那么规则，无法轻易地用单个线性函数近似。采样频率变化、电源故障或传感器中断等事件可能导致数据集的规律性降低，索引性能也随之下降。因此，值得考虑采用更优的位置预测技术，例如学习型索引所提供的技术。

学习型索引利用机器学习模型超越传统索引，最初用于静态内存数据集（Kraska等人，2018年）。早期关于学习型索引的研究主要集中于对经验累积分布函数（eCDF）进行建模。这类索引的示例包括RMI（Kraska等人，2018年）、RadixSpline（Kipf等人，2020年）、PGM索引（Ferragina和Vinciguerra，2020年）以及PLEX（Stoian等人，2021年）。

这些学习型索引的核心思想，是利用经验累积分布函数与元素在有序数组中的位置之间的联系。令 $P(A \leq x)$ 表示小于键 x 的元素所占比例。那么，键 x 在包含 N 个元素的有序数组中的位置为 $pos = \lfloor N \cdot P(A \leq x) \rfloor$。由于难以获得 $P(A \leq x)$ 的精确公式，因此这些索引选择训练不同类型的机器学习模型来估计经验累积分布函数。

基于经验累积分布函数的学习型索引在静态内存使用场景中非常有效。Marcus 等人使用真实世界数据集，将 RMI、PGM 和 RadixSpline 与最先进的传统索引进行了基准测试（Marcus 等人，2020）。实验结果表明，在查找时间和内存占用方面，学习型索引始终优于传统索引，但构建时间除外。

尽管学习型索引在只读工作负载中非常有效，但要使索引能够处理更新仍是一项挑战。这对传感器使用场景尤为重要，因为传感器会持续采集数据并为其建立索引。索引必须能够处理持续追加的数据流。

ALEX（Ding 等人，2020）和 LIPP（Wu 等人，2021）等学习型索引支持更新。这些索引基于树结构，并利用模型搜索树。基于树的学习型索引不仅对经验累积分布函数进行近似，还着重利用模型将数据均匀划分，从而实现较小的树高。树高较小可带来更快的查找速度和更低的内存占用。

尽管可更新学习型索引取得了进展，但在混合读写场景中，它们并不总是优于传统索引。Wongkham 等人（2022）在其基准测试中，针对多种场景测试了 ALEX、LIPP 和 PGM 的一个可更新版本。他们的结论之一是，可更新学习型索引并不能保证空间效率。在更新操作下，ALEX 和 PGM 在内存消耗方面的优势会减弱，而 LIPP 的内存消耗则高于传统索引。这可能会限制学习型索引在传感器上的应用，因为传感器会持续更新，而且内存资源非常有限。

3 调整学习型索引

将学习型索引结构用于传感器时间序列数据，需要支持随着新值不断采集而持续增长的数据规模。因此，我们需要调整学习型索引，使其支持追加操作，同时保持较低的内存消耗。此外，由于记录在采集时会按时间戳存储在有序文件中，学习型索引只适用于按时间戳对传感器数据建立索引。学习型索引不能直接用于对记录中的数据（传感器值）建立索引，因为这些数据并未按顺序存储。

我们重点研究如何调整基于自底向上方法和经验累积分布函数的学习型索引。与需要预先获取全部数据、且难以直接用于时间序列场景的自顶向下建模方法（例如递归模型索引）不同，自底向上构建的索引（例如 RadixSpline）无需预先获取全部数据。我们调整并部署了两种索引结构：RadixSpline 和 PGM索引。

该索引方法基于 SBITS（Fazackerley 等，2021），具体步骤如下：

- 每条包含所采集传感器值的数据记录都会先存储在内存中的缓冲页里，直到该页写满。
- 已满的数据页会按顺序写入已排序数据文件。
- 数据页写入后，会创建一条索引记录，存储该数据页上最小记录的时间戳以及数据页索引。
- 索引算法必须在内存中维护其索引结构，并将大小限制在一定范围内（通常小于 1KB），还可以定期将索引页刷新到存储中以实现持久化。

通过时间戳进行查询时使用索引。通过数据值进行查询的方式与之前的 SBITS 实现保持不变，因为学习型索引不适用于未排序的数据值。

3.1 分段几何模型

分段几何模型索引（PGM）是一种学习型索引，通过分段线性近似（PLA）近似累积分布函数（CDF）（Ferragina 和 Vinciguerra，2020）。PGM 的核心是一个超参数 ε，用于控制线性近似的误差界。

原始PGM索引采用自底向上的递归方式构建。在第一层，PGM使用（O'Rourke，1981）提出的最优算法，对点集 $\{(x_i,i)\}_{i=0\ldots n}$ 构建分段线性近似，并且由（Elmeleegy 等人，2009）重新发现。对于后续层级，PGM采用相同的线性近似策略，将上一层的键 $\{(k_j,j)\}_{j=0\ldots s}$ 作为分段线性近似的点。当某一层恰好只有一条线时，递归终止。

我们通过创建算法 1 中所述的追加方法来改造 PGM。请注意，该方法事先不知道 PGM 的层数，并首先将一个键追加到最底层的线性近似中；如有必要，再将更新向上传播到更高层。我们实现中的另一项修改是采用 Elmeleegy 等人（2009）提出的 Swing Filter 算法。Swing Filter 无法产生最优的线条数，但处理一个点只需 $O(1)$ 的时间和内存。在传感器使用场景中，这是一个优势，因为最优的 Slide Filter 在最坏情况下需要 $O(h)$ 的内存，其中 h 是凸包中的点数，这可能无法装入内存。尽管我们对 PGM 进行了改造以支持追加操作，但其查找键的过程仍与原始 PGM 保持不变（见算法 2）。

算法 1： AppendPGMAdd(pgm,x)

```
L ← pgm.countLevels();
K ←x;
对于 i ← 0 到 L − 1 执行
    c ← pgm.levels[i].统计点数();
    pgm.levels[i].add(K); // 添加
        实现 Swing 过滤器
    如果 pgm.levels[i].countPoints() >c 则
        K ←pgm.levels[i].getLastKey();
    else
        返回;
/* 如果算法执行到这一步
    我们需要创建一个新的
    层级                          */
pgm.levels[L] = newPGMLevel();
firstK ← pgm.levels[L-1].getFirstKey();
lastK ← pgm.levels[L-1].getLastKey();
pgm.levels[L].add(firstK);
pgm.levels[L].add(lastK);
返回;
```

3.2 RadixSpline

RadixSpline 是一种学习型索引，通过线性样条和存储样条点的基数表来近似累积分布函数（Kipf等人，2020年）。决定 RadixSpline 形态的两个超参数是 ε（样条近似的误差界）和 r（基数表条目前缀的长度）。

算法 2： QueryAppendPGM(pgm, x)

```
如果x < pgm.levels[0].getFirstKey() 那么
    return 未找到;
L ← pgm.countLevels();
m← 0; // 模型的索引位置，位于 the
    第 i 层
对于i ← L − 1 到 1 执行
    a ← pgm.levels[i].getSlope(m);
    b ← pgm.levels[i].getIntercept(m);
    pos ← ⌊一个 · x + b⌋;
    lo ← 最大{0, 位置 − ε − 1};
    嗨 ← 最小值{pgm.levels[i].计数键() −
      1，位置 + ε + 1};
    m ← 满足 j 的最小值，使得 x ≥
      pgm.levels[i-1].getKey(j) 和
      lo ≤ j ≤ hi;
a ← pgm.levels[0].getSlope(m);
b ← pgm.levels[0].获取截距(m);
pos ← ⌊一个 · x + b⌋;
低位 ← 最大值{0, 位置 − ε − 1};
高位 ← 最小值{位置 + ε +
  1, pgm.levels[0].count键() − 1, };
/* 搜索是以下实现之一：
    二分搜索或线性搜索
    覆盖原始数组
    包含这些键                          */
返回Search(x, lo, hi);
```

RadixSpline 使用 (Neumann 和 Michel, 2008年) 提出的贪心算法，在经验累积分布函数上构建具有误差界的线性样条。构建完成后，样条点的前缀会被插入基数表中。基数表是一个大小为 2^r 的扁平数组，其中每个表项都映射到样条中的一个范围。

在 RadixSpline 中查询一个点分为三个步骤。第一步是在基数表中查找待搜索键的前缀，并找到对应的样条点。给定样条点后，使用线性插值计算一个大小为 2ε 的窄范围，键可能位于该范围内。最后一步是在该范围内通过线性搜索或二分搜索查找键。由于 2ε 是常数，这一操作很快。

我们对 RadixSpline 进行了改造，使其支持追加操作，具体做法是实现流式版本的 GreedySpline，该算法由 Neumann 和 Michel（2008）提出，如算法 3 所述。追加一个点后，我们会检查能否调整最后一个样条段，使其在 ε 范围内正确近似该点的位置。如果不能，我们就创建一个新的样条段来覆盖这个新点，并将新的样条点传播到基数表中（见算法 4）。

算法 3： RadixSplineAdd(rs,x)

```
c ← rs.样条.countPoints();
rs.spline.add(x); // 添加实现
    贪心样条
如果rs.spline.countPoints() >c 那么
    K ←rs.样条.getLastKey();
      RadixTableInsert(rs, K, c);// 参见
      算法 4
    返回;
返回;
```

算法 4: RadixTableInsert(rs, K, pos)

```
r ← rs.r; // 基数前缀大小
K ←K - rs.minKey;
nB ←mostSignificantBit(bitShiftRight(K,
    rs.shiftSize)); // 位数
    在表中容纳所需
Δ ← max(nB - r, 0); // 可用位数与
    之间的差值
    所需的位数
如果 Δ > 0 那么
    /* 新键触发表
        使用新的前缀大小重建
        以便容纳；我们合并
        旧的基数条目，同时考虑
        新的位移                        */
    rs.shiftSize ← rs.shiftSize + Δ;
    new表 ← allocate表( 2^r );
    对于 i ← 0 到 2^r 执行
        j ← bitShiftRight(i, Δ);
        new表[j] ← min(rs.表[i],
          newTable[j]);
    对于 i ← 2^(r−Δ) 到 2^r 执行
        new表[i] ← INT MAX;
    rs.table ←newTable;
// 更新基数表中的条目
    使用新的位置
T ←bitShiftRight(K, rs.prefixSize);
rs.表[T] ← min(rs.表[T], pos);
return;
```

4 实验结果

实验针对多个真实世界数据集评估了传感器时间序列索引性能。传感器硬件平台搭载基于 32 位 Microchip ARM®Cortex® M0+ 的 SAMD21 处理器，时钟频率为 48 兆赫兹，配备 256 KB 闪存程序存储器和 32 KB 静态随机存取存储器。该硬件板包含多种不同类型的内存，包括一张 SD 卡

和支持原地页级先擦除后写入的串行 NOR DataFlash[1] 。该平台代表了采用常用 32 位 ARM 处理器的嵌入式设备。串行 NOR DataFlash 用于测试无闪存转换层（FTL）的原始存储器性能。在原始存储器上进行测试可确保不会产生维护 FTL 操作的额外开销，并实现最高的读取性能。平台性能特征见表 1。

实验对四种索引进行了基准测试：

- 二**分搜索**：对已排序数据进行二分搜索。这是基准测试的基线方法，无需额外内存。
- **SBITS**：采用线性插值的 SBITS 需要 8 字节来维持线性近似。经过优化后，若预测误差较大，SBITS 默认改用二分搜索。
- **PGM**：支持追加操作且误差界为 $\varepsilon = 1$ 的 PGM索引修改版本。
- **RadixSpline**：支持追加操作的 RadixSpline 修改版本，误差界设为$\varepsilon = 1$，基数前缀大小为$r = 0$；该配置以最小的内存占用实现了最佳性能。

实验测量了四项核心指标：查询吞吐量、每次查询的 I/O 次数、内存消耗和插入吞吐量。这些指标与传感器的常见使用场景相关，例如按时间戳查询和摄取新数据。我们报告了三次独立运行所得指标的平均值。表 2 列出了评估所用的真实世界数据集。这些数据集涵盖多种传感器使用场景，包括环境监测、智能手表、GPS手机数据和化学物质浓度监测。环境数据集**sea** 和 **uwa** 最初来自（Zeinalipour-Yazti 等，2005），并已用于后续多项实验比较（Fazackerley 等人，2021；Ould-Khessal 等，2022）。我们还在图 1 中展示了这些数据集的经验累积分布函数。

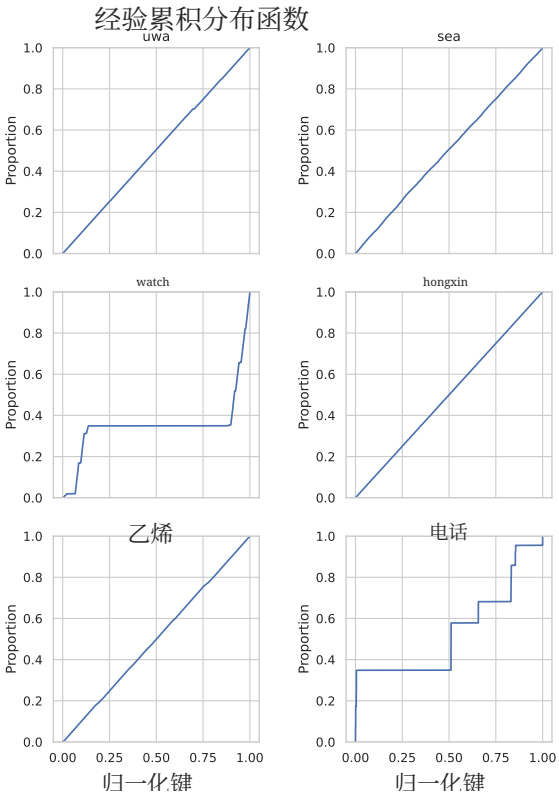


图 1：实验数据集的经验累积分布函数。PGM 和 RadixSpline 使用一组线性模型来将其近似为误差有界的 ε。

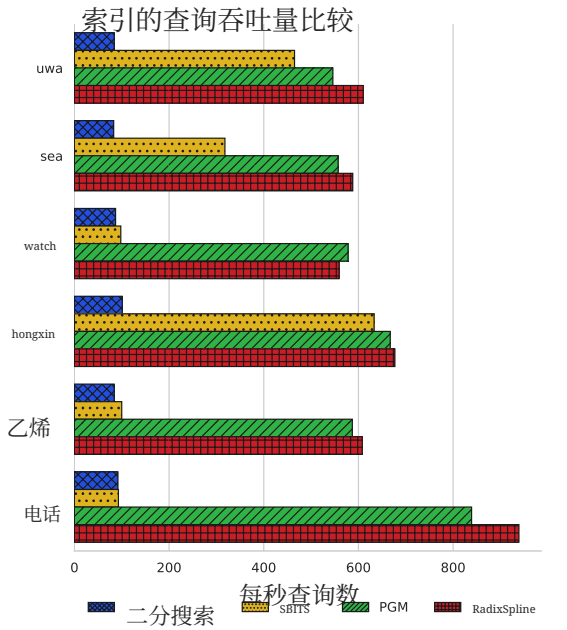


图 2：基准测试中各数据集索引的平均查询吞吐量。速率越高，结果越好。RadixSpline 和 PGM 的表现始终优于 SBITS 和二分搜索。

4.1 查询性能

在边缘设备上处理数据的一种常见用例是查询时间戳。我们测量了搜索时间戳的查询吞吐量。插入传感器数据后，在时间戳范围内查询了 10,000 个随机时间戳值。我们收集了执行所有查询所需的时间以及执行的输入/输出操作次数。

[1]https://www.dialog-semiconductor.com/products/ 内存/数据闪存-spi-内存

	读取（KB/s）		写入（KB/s）		
	Seq	**随机**	**Seq**	**随机**	**写读比**
M0+ SAMD21（数据闪存）	475	475	38	38	12.5

表 1：硬件性能特征

Name	点数	已用点数	传感器数据	来源
sea	100,000	100,000	温度、湿度、风、气压	西雅图-塔科马国际机场
uwa	500,000	500,000	温度、湿度、风、气压	ATG屋顶，华盛顿大学
hongxin	35,064	35,000	PM2.5、PM10、温度	（Zhang等，2017年）
乙烯	4,085,589	100,000	乙烯浓度	(Fonollosa等，2015年)
电话	18,354	18,000	智能手机X/Y/Z磁场	(Barsocchi等，2016年)
watch	2,865,713	100,000	智能手表 X/Y/Z 陀螺仪	(Stisen 等人，2015)

表 2：实验数据集

查询吞吐量结果见图 2。将学习型索引应用于传感器时间序列非常有效。在变化性较大的数据集上，PGM 和 RadixSpline 的表现始终优于 SBITS，提升显著。

对于**hongxin** 和 **uwa** 数据集，性能提升幅度较小：PGM 为 1.05x-1.20x，RadixSpline 为 1.06x-1.31x。这些数据集具有很强的线性特征，因此插值搜索达到了其最佳情况。由于 PGM 和 RadixSpline 使用分段线性近似来拟合数据集，它们的表现也与插值搜索一样出色。

然而，**watch**、**乙烯**和**电话** 等数据集对 SBITS 来说更具挑战性。其性能受到影响，吞吐量逐渐接近二分搜索基线。另一方面，学习型索引的优势进一步扩大，并且能够适应数据分布的变化。PGM 的性能提升幅度为 1.8x-8.9x，RadixSpline 为 1.8x-10x。这表明，学习型索引具有更灵活的模型，能够更好地描述数据分布。

所有数据集上，RadixSpline 与 PGM 的查询吞吐量差异都在 10% 以内。由于这两种方法都采用线性近似，差异取决于如何为这些线性近似建立索引。在这些实验中，RadixSpline 的基数表未分配任何空间，样条点仅通过二分搜索进行查找。由于点数很少，这种做法效果良好。使用大小为 $r = 4$ 和 $r = 8$ 的基数表进行性能测试后发现，查询性能并无提升，反而占用了宝贵的随机存取存储器。这是合理的，因为基数表只节省了少数几次比较

在内存中搜索样条点时，搜索时间主要取决于对闪存执行的 I/O 操作。PGM 会生成多级索引，因此需要更多空间，查询时间也稍长。总体而言，这两种方法都很有效，性能远胜于二分搜索或单次线性插

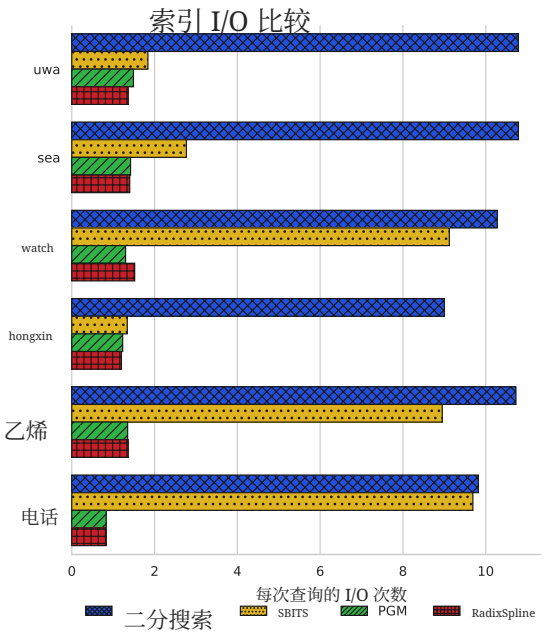


图 3：基准测试中每个数据集的各索引平均每次查询的 I/O 次数。数值越低越好。学习型索引显著减少了所需的 I/O 次数。

值。第 4.2 节将进一步讨论查询性能与内存空间之间的权衡。

查询的另一个相关指标是每次查询的 I/O 次数。执行的 I/O 操作次数主导着查询响应时间，在嵌入式系统中尤其重要，因为这类系统需要可预测的实时性能。图 3 展示了每次时间戳查询平均执行的 I/O 操作次数。二分搜索执行 $O(\log N)$ 次 I/O 操作，其 I/O 操作次数显著高于其他方法。SBITS 的单次线性插值在许多情况下都很有效，但对于 phone 这类数据集，单一线性近似的效果并不好，

因此，由于预测准确度较低，该算法经常退回到二分搜索。PGM 和 RadixSpline 在构造时都能保证误差界。使用 $\varepsilon = 1$ 时，任意查找最多执行 2 次 I/O 操作，平均次数通常在 1.3 到 1.5 次左右。这种可预测的性能是采用这些学习型索引的一大优势。

4.2 内存空间效率

对于学习型索引而言，内存占用较小至关重要，这样才能适用于嵌入式系统。由于随机存取存储器的限制，许多传统技术无法应用于嵌入式系统（Fazackerley 等人，2021），因此需要进行权衡内存与性能的改造。内存结果见表3。

内存消耗（KB）		
	PGM	**RadixSpline**
uwa	0.04	0.09
sea	1.88	0.96
watch	7.80	2.43
hongxin	0.10	0.07
乙烯	0.40	0.12
电话	1.09	0.25

表3：基准测试中各数据集的学习型索引内存消耗对比。

所有索引都能装入内存，最大内存用量为 7.80 KB。结果表明，RadixSpline 的内存消耗低于 PGM。这是意料之中的，因为样条和 PGM 的底层都包含一组非常相似的线性近似。两者的主要区别在于寻找所用线性近似的方法。RadixSpline 使用一个小型基数表为样条点建立索引。PGM 则采用递归方法构建额外的分段线性近似。由于 RadixSpline 的表大小由 r 参数决定，因此可以牺牲少许查询性能来减少内存用量。

对于我们选择的 $r = 0$，内存消耗的差异显著，而查询性能的差异可以忽略不计，因为性能主要取决于 I/O 操作的次数，而非内存中的比较次数。六个数据集中有五个，PGM 的内存用量是 RadixSpline 的两到四倍，例外的是最简单的数据集**uwa**。

通过调整误差界（ε），两种方法都可以减少内存占用，但代价是增加查询输入/输出次数并降低查询吞吐量。表 4 显示了索引大小（字节）、每次查询执行的 I/O 次数，以及在多个不同的 ε 值下，**sea** 数据集的查询吞吐量（次/秒）。将 ε 增至 2 或 3 后，索引大小显著减小。尽管每次查询的 I/O 次数有所增加，但始终不超过 ε。这使设计人员能够以可预测的方式，准确权衡空间占用与查询输入/输出次数之间的性能取舍。

4.3 插入性能

添加索引有助于提升查询性能，但也会影响插入时间。传感器持续接收数据至关重要，因此需要监控插入吞吐量，确保其符合传感器所需的采样率。对于插入性能，每个数据集使用的N 条记录均以硬件允许的最高速率插入。插入性能主要受将数据页写入存储时所需的 I/O 操作影响，但索引构建时间可能会带来一些额外开销。图4中的插入速率展示了各数据集和索引在硬件上所能达到的最高速率。

插入性能的基准是二分搜索情形，其中仅进行简单的数据记录追加，不产生索引开销。平均吞吐量为每秒 1967 次插入。由于传入的时间戳严格递增，且二分搜索不会存储任何有助于搜索的额外信息，因此这代表了插入性能的上限。

SBITS 和 RadixSpline 是插入吞吐量最高的索引，每秒插入次数分别为 1966 和 1965。它们的表现几乎与完全不使用索引的情况相当。SBITS 的结果与先前的实验（Ould-Khessal 等，2022）一致，表明维护索引最新状态所需的 I/O 操作极少。最令人意外的或许是 RadixSpline 的结果，因为它需要计算样条点并更新基数表。我们的基准测试显示，这些操作的开销很小。

PGM 每秒支持 1951 次插入，带来的开销不到 1%。PGM 的插入速率略低，因为部分插入操作会触发 PGM 多个层级的变化，而 RadixSpline 在样条和基数表中最多只会触发一次变化。PGM 的插入吞吐量仍然具有竞争力，胜过 B树等传统嵌入式索引（Ould-Khessal 等人，2022）。

	RadixSpline			PGM		
Test	索引大小	查询输入/输出	吞吐量 索引大小		查询输入/输出	吞吐量
sea $\varepsilon = 1$	932	1.40	588	1920	1.42	557
sea $\varepsilon = 2$	436	1.85	461	856	1.91	440
sea $\varepsilon = 3$	260	2.23	386	664	2.26	376
SEA $\varepsilon = 5$	212	3.12	281	496	3.41	256
SEA $\varepsilon = 10$	132	5.37	166	232	4.94	179

表 4：在 不同误差界下的索引大小（字节）、每次时间戳查询的 I/O 操作数和查询吞吐量（次查询/秒） ε

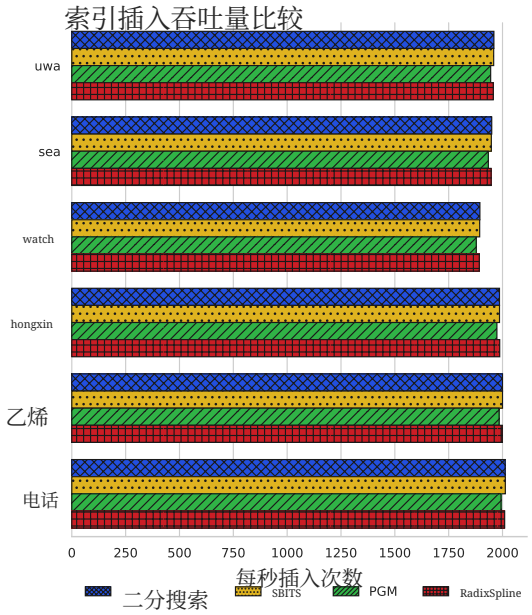


图4：基准测试中各数据集的索引平均插入吞吐量。速率越高，结果越好。PGM 的插入吞吐量略低，但总体而言，学习型索引仍具有竞争力。

4.4 结果讨论

总体而言，实验结果表明，使用针对嵌入式时间序列数据进行适配的学习型索引具有显著优势。最显著的优势是时间戳查询性能可预测且有界。通过指定误差界 (ε)，每次查询的 I/O 次数上限为 $2\varepsilon + 2$。其查询性能显著优于 SBITS 线性插值搜索或二分搜索。就插入吞吐量而言，索引带来的开销极小。为应对内存有限的问题，可以通过增大误差界来缩小索引大小。在所测试的实验中，索引大小通常不到几个 KB。索引算法能够支持实验硬件以每秒近 2000 条记录的速度实时采集传感器数据，这一速度显著高于大多数应用的采集速率。

5 结论

本研究针对传感器时间序列数据改造并应用了学习型索引。实验表明，RadixSpline 和 PGM 适合应对嵌入式系统在其独特约束下为数据量庞大的数据建立索引这一挑战。本研究搭建了嵌入式系统索引与学习型索引最初面向大型内存数据库的应用之间的桥梁。只要进行适当的改造，就可以重用这些为查询大型静态数据集而设计的结构，并将其应用于内存极小且频繁追加数据的嵌入式系统。

未来的工作将扩展面向传感器的学习型索引，使其能够自动处理超参数并减少调优需求。我们设想，学习型索引能够自动调整参数，以适应嵌入式系统中有限的资源，具体而言，通过调整误差界来确保索引能够容纳于可用内存中。

参考文献

Barsocchi, P., Crivello, A., Rosa, D. L., and Palumbo, F.（2016）。一种用于基于无线局域网和地磁场指纹识别的室内定位方法的多源多变量数据集。载于*2016*年室内定位与室内导航国际会议（*IPIN*），第1–8页。IEEE。

Ding, J., Minhas, U. F., Yu, J., Wang, C., Do, J., Li, Y., Zhang, H., Chandramouli, B., Gehrke, J., Kossmann, D., Lomet, L., and Kraska, T.（2020）。ALEX：一种可更新的自适应学习型索引。载于*2020*年*ACM SIGMOD*数据管理国际会议论文集，第969–984页。ACM。

Douglas, G. and Lawrence, R.（2014）。LittleD：面向传感器节点和嵌入式应用的SQL数据库。载于应用计算研讨会，第827–832页。Elmeleegy, H., Elmagarmid, A. K., Cecchet, E., Aref, W. G., and Zwaenepoel, W.（2009）。具有精度保证的数值流在线分段线性近似。*Proc. VLDB Endow.*，2(1):145–156。Fazackerley, S., Ould-Khessal, N., and Lawrence, R.（2021）。面向时间序列数据的高效闪存索引

用于内存受限的嵌入式传感器设备。载于第十届传感器网络国际会议（*SENSORNETS 2021*）论文集，第92–99页。SCITEPRESS。Ferragina, P. and Vinciguerra, G.（2020）。PGM-index：一种具有可证明最坏情况界限的全动态压缩学习型索引。*PVLDB*，13(8):1162–1175。Fevgas, A., Akritidis, L., Bozanis, P., and Manolopoulos, Y.（2020）。闪存存储设备中的索引：挑战、当前方法与未来趋势综述。*VLDBJ.*，29(1):273–311。Fonollosa, J., Sheik, S., Huerta, R., and Marco, S.（2015）。储层计算补偿了化学传感器阵列在连续监测中对快速变化气体浓度的缓慢响应。传感器与执行器*B*：化学，215:618–629。Kipf, A., Marcus, R., van Renen, A., Stoian, M., Kemper, A., Kraska, T., and Neumann, T.（2020）。RadixSpline：单遍学习型索引。载于利用人工智能技术进行数据管理第三届国际研讨会论文集，第1–5页。Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N.（2018）。学习型索引结构的价值。载于*2018*年*SIGMOD*数据管理国际会议论文集，第489–504页。Marcus, R., Kipf, A., van Renen, A., Stoian, M., Misra, S., Kemper, A., Neumann, T., and Kraska, T.（2020）。学习型索引基准测试。*Proc. VLDBEndow.*，14(1):1–13。Neumann, T. and Michel, S.（2008）。具有误差保证的平滑插值直方图。载于计算机科学讲义，第126–138页。施普林格·柏林·海德堡。O'Rourke, J.（1981）。一种在数据范围之间拟合直线的在线算法。*ACM*通讯，24(9):574–578。Ould-Khessal, N., Fazackerley, S., and Lawrence, R.（2022）。使用位图、分区和树评估嵌入式时间序列索引的性能。载于*SENSORNETS 2021*特邀与修订论文，第1674卷传感器网络，第125–151页。施普林格。Stisen, A., Blunck, H., Bhattacharya, S., Prentow, T. S., Kjærgaard, M. B., Dey, A., Sonne, T., 和 Jensen, M. M.（2015）。智能设备各不相同：评估并缓解移动感知异质性，以实现活动识别。载于第*13*届*ACM*嵌入式网络传感器系统会议论文集，SenSys '15，第127–140页，美国纽约。ACM。Stoian, M., Kipf, A., Marcus, R., 和 Kraska, T.（2021）。迈向实用学习型索引。*https://arxiv.org/abs/2108.05117*。Tsiftes, N. 和 Dunkels, A.（2011）。每个传感器中都有一个数据库。载于第*9*届*ACM*嵌入式网络传感器系统会议论文集，SenSys '11，第316–332页，美国纽约州纽约市。ACM。Wang, C., Huang, X., Qiao, J., Jiang, T., Rui, L., Zhang, J., Kang, R., Feinauer, J., Mcgrail, K., Wang, P., Luo, D., Yuan, J., Wang, J., 和 Sun, J.（2020）。Apache IoTDB：面向物联网的时间序列数据库。*Proc.VLDBEndow.*，13(12):2901–2904。Wongkham, C., Lu, B., Liu, C., Zhong, Z., Lo, E., 和 Wang, T.（2022）。可更新学习型索引是否已准备就绪？*VLDB*基金会会刊，15(11):3004–3017。Wu, J., Zhang, Y., Chen, S., Wang, J., Chen, Y., 和 Xing, C.（2021）。具有精确位置的可更新学习型索引。*Proc. VLDB Endow.*，14(8):1276–1288。Yang, Y., Cao, Q., 和 Jiang, H.（2019）。EdgeDB：面向边缘计算的高效时间序列数据库。*IEEEAccess*，7:142295–142307。Yin, S. 和 Pucheral, P.（2012）。PBFilter：一种适用于嵌入式系统的基于闪存的索引方案。信息系统，37(7):634–653。Zeinalipour-Yazti, D., Lin, S., Kalogeraki, V., Gunopulos, D., 和 Najjar, W.（2005）。MicroHash：一种适用于基于闪存的传感器设备的高效索引结构。载于*FAST '05*文件与存储技术会议论文集，第31–43页。USENIX协会。Zhang, S., Guo, B., Dong, A., He, J., Xu, Z., 和 Chen, S. X.（2017）。北京空气质量改善的警示故事。英国皇家学会会刊*A*辑：数学、物理与工程科学，473(2205):20170457。