A Tutorial on Learned Multi-dimensional Indexes

SIGSPATIAL 2020

Presenters: Abdullah-Al-Mamun, Hao Wu, Walid G. Aref
Purdue University
West Lafayette, Indiana, USA

Website of Tutorial: https://www.cs.purdue.edu/homes/aref/learned-indexes-tutorial.html
Outline of the Tutorial

• Introduction and Taxonomy
• Indexing the Learned Models vs. Learning the Indexes
• Static vs. Dynamic Learned Indexes
• Fixed vs. Dynamic Data Layout
• Learned One-Dimensional Indexes
• Learned Multidimensional Indexes
• Open Problems for Future Research
Outline of the Tutorial

• **Introduction and Taxonomy**
  • Indexing the Learned Models vs. Learning the Indexes
  • Static vs. Dynamic Learned Indexes
  • Fixed vs. Dynamic Data Layout
• Learned One-Dimensional Indexes
• Learned Multidimensional Indexes
• Open Problems for Future Research
Introduction: ML and DB

• Machine Learning (ML) has been successful in many application domains
• Two recent trends of research in the area of Database Systems (DB):

  • **Machine Learning for Database Systems (ML for DB)**
    • Replace core components of a Database System (e.g., query optimizer, Indexes, DB administration) with Machine Learning techniques
      • Achieve better performance
      • Less space requirement

  • **Database Systems for Machine Learning (DB for ML)**
    • Extend database system techniques to support efficient ML workloads
Introduction: Database Indexing

• Database Index: Provide efficient access to data
• Popular index structure is: B+-tree
• Given search key, B+-tree identifies the storage location of the tuple that contains the search key
• Can view the B+-tree as a function: B+-tree(key) that takes a key as input and returns the order of key’s tuple inside the table
Can one use ML techniques to guide data indexing?  
Can we learn the function:  
  • B+-tree(key) → Location of tuple in table?  
Can we replace the B+-tree with an ML model?  
  • “Index as a model”  
  • ML_Model (key) predicts the storage location of the key  
  • Searching executes potentially in O(1) time  
  • → “Learned Index”
Introduction: Initial Results

Initial performance results (approximate) of a learned index

Promising → Faster lookup time and smaller storage

- Fast Lookup Table
- Fixed Sized Read Optimized B-Tree with Interpolation Search
- Learned Index

Size (MB) vs. Lookup-Time (ns)

Larger

256

Smaller

0,0

Faster

Slower

350

Taxonomy of Learned Indexes

LEARNED INDEX

Mutable

Immutable

Learning the Index

Indexing Learned Models

Benchmarking

Fixed Data Layout

Dynamic Data Layout

SpatioTemporal

Sequential

One-d

Multi-d

One-d

Multi-d

One-d

Multi-d

ALEX[4]
MADEX[15]
Fitting-tree[10]
Doraemon[38]
IFB-Tree[14]
LISA[24]
RSMI[33]

SageDB[21]
ZM-index[41]
ML-index[3]
Learned BF[27]
SIndex[42]
PGM[7]
ASLM[25]

Flood[31]
Tsunami[5]
Case[32]
Qd-tree[45]

Hands-off[16]

Handwritten Trie[22]

SOSD[19]

Benchmark[28]


LISA[24] RSMI[33]

SOSD[19] Benchmark[28]
Outline of the Tutorial

• Introduction and Taxonomy
• Indexing the Learned Models vs. Learning the Indexes
  • Static vs. Dynamic Learned Indexes
  • Fixed vs. Dynamic Data Layout
• Learned One-Dimensional Indexes
• Learned Multidimensional Indexes
• Open Problems for Future Research
Machine Learning and data indexing interact in two possible ways:

1. **Indexing the Learned Models**
   - Given a collection of ML models, e.g., object recognition models (Cats, Dogs, Trains, etc.), and an input object, say o
   - Identify the class of o (cat vs. dog, etc.)
   - Instead of executing all models and identifying which has the highest matching score
   - Can we index the learned models to speed up the matching process?

2. **Learning the Index**
   - Given a key value, say k, and an ordered array of key values
   - Build an ML-based model that helps predict the location of k in the ordered array
Taxonomy of Learned Indexes

- **LEARNED INDEX**
  - Learning the Index
  - Indexing Learned Models
  - Benchmarking

- **Immutable**
  - Fixed Data Layout
  - Dynamic Data Layout

- **Mutable**
  - One-d
  - Multi-d

- **One-d**
  - RMI
  - CDF Shop
  - Pavo
  - SageDB
  - ZM-index
  - ML-index
  - Learning BF
  - Flood
  - Tsunami
  - Case
  - Qd-tree

- **Multi-d**
  - ALEX
  - MADEX
  - Acc
  - B+ Tree
  - Slindex
  - PGM
  - ASLM
  - Fitting-tree
  - Doraemon
  - XIndex
  - BF-Sandwich
  - IFB-Tree
  - Hybrid-O
  - Drift Model

- **Dynamic Data Layout**
  - Hands-off
  - SoftFunctional

- **SpatioTemporal**
  - Trie with HMM
  - R-Tree with HMM

- **Sequential**
  - Music Retrieval

- **Benchmarking**
  - SOSD
  - Benchmark
• Collection of spatiotemporal sequences, e.g., heart pulse rates, stock market trends over time, handwritten drawing on a tablet, object movement trajectory
  • Index shown in the context of handwritten text

• Divide each spatiotemporal sequence into basic alphabet symbols

• Because of the variability, there is a need for training to recognize similar, but not exactly the same, patterns

• Model each alphabet symbol in the spatiotemporal sequence using *local spatiotemporal features* along the trajectory of the sequence
  • Time
  • Velocity
  • Direction
  • Acceleration
  • Aspect ratio, . . .

• Left-to-right Hidden Markov Models are suitable for representing spatio-temporal sequences

• Instead of building a Hidden Markov Model for each entire sequence, we build an HMM for each alphabet symbol in the spatiotemporal sequence

• Need to segment each spatio-temporal sequence into alphabet symbols

• Train the left-to-right Hidden Markov Model using multiple samples of the alphabet symbols

• Construct a trie structure over the learned alphabet symbols

The Handwritten Trie: Indexing Electronic Ink

Hidden Markov Model

Each HMM is constructed to represent a pictogram class (accept a specific pictogram with high probability)
- **Left-to-right HMM**: Useful for modeling cursive handwritten text
- **Input**: Pictogram
- **Output**: Matching Probability

Traverse Nodes (HMM Models)

Choose Nodes with highest probability

choose combination of nodes with highest probability

Input: “be”

Output: “be”
Indexing the Learned Models: Summary

• The Handwritten Trie: Indexing Electronic Ink is one of the earlier works to index the models.

• Another early work about indexing the models using R-Tree-like structure in the context of indexing HMMs for music retrieval can be found in [ISMIR’02]

The Case for Learned Index Structures [SIGMOD'18]

Introduction

• Introduced the idea that “Indexes are models”
• Replace traditional database indexes by learned models
• Approximate the Cumulative Distribution Function (CDF) of the underlying (sorted) data
• Proposed Recursive Model Index (RMI), a multi-stage ML model
• Combine simpler ML models
  • The first stage model will make an initial prediction of the CDF for a specific key
  • The next stage models will be selected to refine this initial prediction
• Proposed Learned Index Structures: Range Index, Point Index, and Existence Index

The Case for Learned Index Structures  [SIGMOD’18]

Mechanism

Recursive Model Index

Range Index

Model 2.1  Model 2.2  Model 2.3

Model 3.1  Model 3.1  Model 3.1  Model 3.1

Position

Point Index

Learned Hash-map

Existence Index

Key 1  Key 2  Key 3

Model  Model  Model

Bloom Filter with Model-Hashes

Key

Model

Bloom Filter

No

Yes

Bloom Filter as Classification Problem
Limitations:

- Focus on in-memory read-only workloads
- The structure of RMI is static
- Does not support updates (e.g., insertion, deletion)

Many follow up works extend on this paper
### Introduction

<table>
<thead>
<tr>
<th>Target</th>
<th>2-stage Recursive Model Indexes (RMIs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Idea</td>
<td>• Build a tool to explore and optimize RMIs</td>
</tr>
<tr>
<td>Objectives</td>
<td>• Test RMI configurations</td>
</tr>
<tr>
<td></td>
<td>• Explore automatic optimization</td>
</tr>
</tbody>
</table>

CDFShop: Exploring and Optimizing Learned Index Structures [SIGMOD’20]

Introduction

1. Explore RMIs
   - Allow users to test and explore RMI with different configurations:
     - visualize how RMI approximates the CDF of data distribution
     - measure lookup performance
     - explore size VS time performance compared with binary search and B-tree

2. Automatic Optimization
   - Allow users to visualize steps of optimization process

2-stage Recursive Model Index

Key Input

Root Model

Model 1.1
Model 1.2

Hyperparameters
- Choice of model
- Branching factor: number of branches

Benchmarks
- Inference Time (Time needed to predict the index of a key)
- Size
- Training Error (Error Range)
Taxonomy of Learned Indexes

LEARNED INDEX

- Learning the Index
  - LEARNED INDEX
    - Indexing Learned Models
      - Benchmarking

- Immutable
  - Fixed Data Layout
    - One-d
      - RMI[22]
    - Multi-d
      - SageDB[21]
        - ZM-index[41]
        - ML-index[3]
        - Learned BF[27]
        - Flood[31]
        - Tsunami[5]
        - Case[32]
        - Qd-tree[45]
        - ALEX[4]
        - MADEX[15]
        - AccB+Tree[26]
        - SIndex[42]
        - PGM[7]
        - ASLM[25]
        - Fiting-tree[10]
        - Doraemon[38]
        - XIndex[39]
        - BF-Sandwich[30]
        - ADEL[23]
        - IFB-Tree[14]
        - Hybrid-O[34]
        - Drift Model[13]

- Mutable
  - Dynamic Data Layout
    - One-d
      - Hands-off[16]
    - Multi-d
      - LISA[24]
      - RSMI[33]
      - Handwritten Trie[22]
      - R-Tree with HMM
      - Trie with HMM
      - SOSD[19]
      - Benchmark[28]

- SpatioTemporal
  - Sequential

- Music Retrieval[46]
### Pavo: A RNN-Based Learned Inverted Index [ACCESS’18]

<table>
<thead>
<tr>
<th>Target</th>
<th>Inverted index (learned index for inverted data)</th>
</tr>
</thead>
</table>
| Objectives | • adaptive to various data distributions of inverted data  
• lower collision rate  
• higher space utilization rate |
| Core Idea | • RNN Based Learned Index  
• 4 Stages  
• Supervised & Unsupervised Models |

Pavo: A RNN-Based Learned Inverted Index [ACCESS’18]
Outline of the Tutorial

• Introduction and Taxonomy
• Indexing the Learned Models vs. Learning the Indexes
  • Static vs. Dynamic Learned Indexes
  • Fixed vs. Dynamic Data Layout
• Learned One-Dimensional Indexes
• Learned Multidimensional Indexes
• Open Problems for Future Research
Given a Learned Index, can we support updates?
Why are updates hard?
- The learned index takes significant time to train
- New data will require retraining because it changes the order of the data

Classify learned indexes based on the ability to support updates:

1. **Immutable Learned Index:**
   - Does not support inserts, updates, or deletes

2. **Mutable Learned Index:**
   - Supports inserts, updates or deletes
### Introduction

**Target**

Learned Index (Stable & Dynamic Workload)

**Objectives**

Implement dynamic, updatable learned index to handle dynamic workload

**Core Idea**

- Adaptive RMI as Model Hierarchy
- Linear Regression Model as Node
- Gapped Array or Packed Memory Array as Node Layout

---

ALEX: An Updatable Adaptive Learned Index [SIGMOD’20]

Mechanism

Key Input

Root Model

Model 1.1

Model 1.2

Model 2.1

Model 2.2

Model 3.1

Model 3.2

Adaptive Recursive Model Index

Node Layout

Packed Memory Array

Gapped Array

Exponential Search
ALEX: An Updatable Adaptive Learned Index [SIGMOD’20]

**Node Layout**

1. **Gapped Array (GA)**
   - Packed Memory Array
   - Gapped Array
   - Exponential Search

2. **Packed Memory Array (PMA)**
   - Binary Tree Representation of PMA

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>8</th>
<th>11</th>
</tr>
</thead>
</table>
• Limitations:
  • Adversarial workload when data is skewed
  • ALEX does not support duplicate keys of secondary indexes
  • Requires concurrency control to handle updates with concurrent lookups
  • How to check sorted order during insertion in gapped array (linear search?)
Outline of the Tutorial

• Introduction and Taxonomy
• Indexing the Learned Models vs. Learning the Indexes
• Static vs. Dynamic Learned Indexes
• **Fixed vs. Dynamic Data Layout**
  • Learned One-Dimensional Indexes
  • Learned Multidimensional Indexes
• Open Problems for Future Research
There are two types of data layouts for learned indexes:

1. **Fixed Data Layout:**
   - The layout of the data and the structure of the index is fixed before building the learned index

2. **Dynamic Data Layout:**
   - The layout of the data is arranged and can be modified by the ML models while building and updating the learned index
<table>
<thead>
<tr>
<th>Target</th>
<th>B-tree</th>
</tr>
</thead>
</table>
| Core Idea | • Interpolation-Friendly B-tree:  
1. Interpolation  
2. Error Window |
| Objectives | • Enhance the performance of B-tree without changing too much of the data structure |

Interpolation-friendly B-trees: Bridging the Gap Between Algorithms and Learned Indexes [EDBT’19]

Interpolation-Friendly B-tree

- Similar structure to B-tree
- Guaranteed performance as B-tree
- Ideas from learned index (intra-node interpolation, error bounds)

Intra-nodes Interpolation

\[ \hat{p}_q = \left[ \frac{\nu_q - \nu_i}{\nu_{i+1} - \nu_i} \times \text{node\_size} \right] \]

- \(\nu_q\): queried key
- \(\nu_i, \nu_{i+1}\): two keys in the B-tree that \(\nu_q\) is in between
- \(\hat{p}_q\): interpolated index of \(\nu_q\)

Global* Error Window \(\Delta\)

\[ [\hat{p}_q - \Delta, \hat{p}_q + \Delta] \]

- A B-tree node is “Interpolation-Friendly” if any key can be found within error window

* Setting error window global to avoid excessive memory footprint and to exploit the SIMD capability
## Considerations for Handling Updates in Learned Index Structures [AIDM’19]

<table>
<thead>
<tr>
<th><strong>Target</strong></th>
<th>Learned Index Structures for Updatable Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objectives</strong></td>
<td>Reduce Prediction Error Caused by Updates</td>
</tr>
<tr>
<td><strong>Core Idea</strong></td>
<td>Instead of retraining the learned index, model the drift due to insertion and calibrate the prediction by learned index</td>
</tr>
</tbody>
</table>

Considerations for Handling Updates in Learned Index Structures [AIDM’19]

Mechanism

Segmentation

Drift Estimation & Correction

Key

Model

Pos

... Segment i-1  Segment i  Segment i+1 ...

... Pos ...

Reference Point

Reference Point

... Segment i-1  Segment i  Segment i+1 ...

... Pos ...

Insertions

P + drift estimated from interpolation
Taxonomy of Learned Indexes
<table>
<thead>
<tr>
<th>Target</th>
<th>Dynamic Workloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objectives</td>
<td>Improve query performance for skew queries and speed up model re-training with data distribution shift</td>
</tr>
</tbody>
</table>
| Core Idea    | ● Incorporate query distribution  
                   ● Reuse pre-trained model |

Learned Indexes for Dynamic Workloads

Mechanism

- Workload
- Dataset

Training Set Generator

Analyzer

Model Cache

Similarity

Auto Tuner

Fine Tuner

Finalizer

Incorporate Read Access Pattern

Reuse Pre-trained Models
Taxonomy of Learned Indexes

LEARNED INDEX

- Learning the Index
- Indexing Learned Models
- Benchmarking

Immutable
- One-d
- Fixed Data Layout

Multi-d
- Dynamic Data Layout

Mutable
- One-d
- Dynamic Data Layout

Multi-d
- SpatioTemporal
- Sequential

- One-d
- Hands-off

- Multi-d
- SoftFunctional

One-d
- Hands-off

Multi-d
- Triage with HMM

Dynamic Data Layout
- Trie with HMM

One-d
- SOSD

Multi-d
- RSMI

- One-d
- Learned BF

Multi-d
- R-Tree with HMM

ALEX
- MADEX
- Acc
- B+ Tree
- Slindex
- PGM
- ASLM
- Fiting-tree
- Doraemon
- XIndex
- BF-Sandwich
- AIDEL
- IFB-Tree
- Hybrid-O
- Drift Model
- Hands-off
- SoftFunctional
- LISA
- RSMI
- SOSD
- Benchmark

Benchmarking
- AIDEL
- ASLM
- Hybrid-O
- XIndex
- SIndex
- BF-Sandwich
- AIDEL

Music Retrieval
- RSMI
- SOSD

SpatioTemporal
- SOSD

Sequential
- SOSD

SoftFunctional
- SOSD

Learning the Index
- SOSD

Indexing Learned Models
- SOSD

- Benchmarking
- SOSD

Mutable
- SOSD

Dynamic Data Layout
- SOSD

Fixed Data Layout
- SOSD

One-d
- SOSD

Multi-d
- SOSD

One-d
- SOSD

Multi-d
- SOSD

- Benchmarking
- SOSD

- Learning the Index
- SOSD

Indexing Learned Models
- SOSD

Benchmarking
- SOSD

Outline of the Tutorial

• Introduction and Taxonomy
• Indexing the Learned Models vs. Learning the Indexes
• Static vs. Dynamic Learned Indexes
• Fixed vs. Dynamic Data Layout
• **Learned One-Dimensional Indexes**
• Learned Multidimensional Indexes
• Open Problems for Future Research
Outline of the Tutorial

• Introduction and Taxonomy
• Indexing the Learned Models vs. Learning the Indexes
• Static vs. Dynamic Learned Indexes
• Fixed vs. Dynamic Data Layout
• Learned One-Dimensional Indexes
• Learned Multidimensional Indexes
• Open Problems for Future Research
Taxonomy of Learned Indexes

LEARNED INDEX

- Learning the Index
- Indexing Learned Models
- Benchmarking

.Immutable

.Fixed Data Layout

.Dynamic Data Layout

.Mutable

-One-d
- Multi-d

- One-d RMI[22], CDF Shop[29], Pavo[44], RS[20]
- Multi-d SageDB[21], ZM-index[41], ML-index[3], Learned BF[27]
- One-d ALEX[4], MADEX[15], Acc B+ Tree[26], Slindex[42], PGM[7], ASLM[25]
- One-d LISA[24], RSMI[33]
- Multi-d Trie with HMM

-One-d SOSD[19], Benchmark[28]
- Multi-d Handwritten Trie[22], Music Retrieval[46]

SpatioTemporal

Sequential

Trie with HMM

R-Tree with HMM
• One of the earliest distribution-aware spatial indexes can be found in:

• Can ML models replace and act in place of a multi-dimensional index?
  • Yes, ML models can act in place of a multi-dimensional index, e.g., R-Tree
Challenges for Learned Multidimensional Indexes

Introduction

- **Sorting/ordering of multi-dimensional data:**
  - No obvious sort order for multi-dimensional data

- **Error correction mechanism in case of misprediction:**
  - Difficult to define an error correction mechanism in case of mispredictions

- **Choice of the ML model:**
  - Which ML models to choose?

- **Layout of the data:**
  - How to store the data?
  - Affect range query time and model accuracy
• Proposal:
  • **Step-1**: Project the multi-dimensional data points into one-dimensional space
    • Successively sorting and partitioning points along a sequence of dimensions into equally-sized cells
      • Produces a layout that is efficient to compute and learnable
    • Comparing with Z-order which is difficult to learn
  • **Step-2**: Uses a trained CDF model (e.g., RMI) to predict the physical location of the point
• Initial Result:

• R-Tree vs. Learned Multi-dimensional Index on TPC-H data

Result on average query time

Result on index size

Taxonomy of Learned Indexes

Learning the Index

Immutable

One-d

Multi-d

Flexible Data Layout or Arranged by Model

Data Type

Fixed Data Layout

ALEX[4]

SageDB[21]

ZM-index[41]

ML-index[3]

Learned BF[27]

ALSA[15]

Acc B+ Tree[26]

SIndex[42]

PGM[7]

ASLM[25]

Fitting-tree[10]

Doraemon[38]

XIndex[39]

BF-Sandwich[30]

AIDEL[23]

IFB-Tree[14]

Hybrid-O[34]

Drift Model[13]

Hands-off[16]


Mutable

One-d

Multi-d

Dynamic Data Layout

LEARNED INDEX

Indexing Learned Models

BENCHMARKING

One-d

SOSD[19]

Benchmark[28]

Sequential

SpatioTemporal

Handwritten Trie[22]

Music Retrieval[46]
Learned Index for Spatial Queries \([\text{MDM’19}]\)

**Mechanism**

- **Z-address Computation**
  - Multi-dimensional Data → Z Address
  - ZM-Index:
    - Spatial Query Processing: Point and range queries
    - Uses the Z-order to map the multi-dimensional values to the one-dimensional space
    - Uses a multi-staged model (e.g., RMI) for learning

**Core Idea**

- Spatial Query Processing: Point and range queries
- Uses the Z-order to map the multi-dimensional values to the one-dimensional space
- Uses a multi-staged model (e.g., RMI) for learning

---

The ML-Index: A Multidimensional, Learned Index for Point, Range, and Nearest-Neighbor Queries [EDBT’20]

Core Idea

ML-Index:
• Z/Morton order cannot be easily learned by ML models.
• Multi-dimensional data should be sorted in an order that can be easily learned.
• Partition and transform the data into one-dimensional values based on distribution-aware reference points.
• Combines the scaled ordering with ML models

Efficient Scaling

Offset Method:
• \( m \) reference points \( O_i \) are chosen each can be thought as a centroid of the data partition \( P_i \).
• The closest reference points of \( O_i \) are used to build the partition \( P_i \).
• The minimal distance of a point to the reference points is \( d_l \)
• Scaled value = \( \text{offset}_i + \text{dist}(O_i, d_l) \)
• For reference points \( O_1, O_2, \ldots O_m \) and their partitions \( P_1, P_2, \ldots P_m \) of \( fset_i = \sum_{j<i} r(j) \)
• \( r \): The maximal distance from \( O_j \) to the points in partition \( P_j \)

1. Find the closest reference point \( O_i \) and calculate the scaled value.
2. Model (key)→ predicted position.
3. Local search

Query Processing (Point)

Lifting the Curse of Multidimensional data with Learned Existence Indexes

[ML for Systems@NeurIPS’18]

Motivation

- Maximizing performance of learned bloom filter for handling multi-dimensional data

Core Idea

- Model high-cardinality attributes with Recurrent Neural Network (RNNs)
- Bloom Filter Sandwiching
- Robust Learning

[27] Stephen Macke, Alex Beutel, Tim Kraska, Maheswaran Sathiamoorthy, Derek Zhiyuan Cheng, and EH Chi. 2018. Lifting the curse of multidimensional data with learned existence indexes. In Workshop on ML for Systems at NeurIPS.
Qd-tree: Learning Data Layouts for Big Data Analytics [SIGMOD’20]

Motivation

- For disk-based systems, an important performance metric is:
  - The number of data blocks accessed by a query.
- Problem Statement:

Core Idea

- Minimize the number of blocks/records accessed by a workload
  - Generating block-level layouts with excellent I/O performance
- Query-data routing trees (Qd-trees) are neural network-generated decision trees that
  - Recursively partition the data space into smaller subspaces.
- Use Deep Reinforcement Learning to create Qd-trees
  - Proximal Policy Optimization (PPO)

System Architecture

For disk-based systems, an important performance metric is: the number of data blocks accessed by a query. The problem statement motivates the need for an efficient data layout that minimizes the number of accesses. Qd-trees are generated using deep reinforcement learning with proximal policy optimization. The system architecture involves partitioning the multi-dimensional data to maximize the total number of skipped data blocks/records. The queries are routed through the Qd-tree constructor, which is generated offline using candidate cuts. The data router samples the data online to create the learned data layout.
• Experiments over benchmark and real workloads
  • Compared to current blocking schemes:
    • Provides physical speedups of more than an order of magnitude

• For data skipping based on selectivity:
  • Performs within 2X of the lower bound
Learning Multi-dimensional Indexes [SIGMOD’20]

Core Idea

- Proposed index structure: “Flood”
  - Read optimized grid-based index over the multi-dimensional space
  - Co-optimize the data layout and the index structure
    - For particular data and query distributions
- Two components:
  - Offline (pre-processing)
    - Chooses an optimal layout
    - Creates an index based on that layout
  - Online
    - Query execution

System Architecture

Queries → Optimize Layout → Preprocess Dataset → Optimal Layout → Find Intersecting Cells → Estimate Physical Index and Rectify → Scan and Filter → Execution Engine → Queries

Flood’s Workflow

- **Projection:**
  - Identifies the intersecting cells
  - Identifies the physical index range in each intersecting cell

- **Refinement:**
  - Utilizes the ordering of points within each cell to refine each physical index range

- **Scan and Filter**

![Diagram showing Flood’s Workflow]

Cell’s physical index range

Refined physical index range

Matched by query filter
Not matched by query filter
• Experimental Results:
  • Outperforms optimally tuned spatial indexes
  • Uses only a fraction of the space comparing with traditional indexes

• Limitations:
  • Cannot adapt to skewed query workload
  • If dimensions are correlated,
    • Performance and memory usage are affected

## Taxonomy of Learned Indexes

**LEARNED INDEX**

- Learning the Index
- Indexing Learned Models
- Benchmarking

### Indexing Learned Models

- Fixed Data Layout
- Dynamic Data Layout

#### Immutable

- One-d
- Multi-d

#### Mutable

- One-d
- Multi-d

### Benchmarking

- SOSD\(^{19}\)
- Benchmark\(^{28}\)
- LearnBF\(^{27}\)
-Learned BF\(^{27}\)

### Learning the Index

- Hands-off\(^{16}\)
- SoftFunctional\(^{11}\)
- Acc\(^{26}\)
- Trie with HMM
- R-Tree with HMM

### Indexing Learned Models

- Music Retrieval\(^{46}\)
- Handwritten Trie\(^{22}\)
- Trie with HMM

### Benchmarking

- RSMI\(^{33}\)
- B+ Tree\(^{26}\)
- Acc\(^{26}\)
- Trie with HMM

### Immutable

- SageDB\(^{21}\)
- ZM-index\(^{41}\)
- ML-index\(^{3}\)
- Learned BF\(^{27}\)
- Flood\(^{31}\)
- Tsunami\(^{5}\)
- Case\(^{32}\)
- Qd-tree\(^{45}\)

### Mutable

- ALEX\(^{4}\)
- MADEX\(^{15}\)
- Acc\(^{26}\)
- Slindex\(^{42}\)
- PGM\(^{7}\)
- ASLM\(^{25}\)
- Fiting-tree\(^{10}\)
- Doraemon\(^{38}\)
- XIndex\(^{39}\)
- BF-Sandwich\(^{30}\)
- AIDEL\(^{23}\)
- IFB-Tree\(^{14}\)
- Hybrid-O\(^{34}\)
- Drift Model\(^{13}\)

### Dynamic Data Layout

- One-d
- Multi-d

### Fixed Data Layout

- One-d
- Multi-d
• Extend the idea of Flood to overcome its limitations
  Tsunami: A Learned Multi-dimensional Index for Correlated Data and Skewed 
  • Adaptable to changes in workload
  • Scales across data size, query selectivity, and dimensionality
  • Up to 6× faster
Taxonomy of Learned Indexes

LEARNED INDEX

- Learning the Index
  - Immutable
  - Fixed Data Layout
    - One-d
      - RMI[22]
      - CDF Shop[29]
      - Pavo[44]
      - RS[20]
  - Multi-d
    - SageDB[21]
    - ZM-index[41]
    - ML-index[3]
    - Learned BF[27]
    - Flood[31]
    - Tsunami[5]
    - Case[32]
    - Qd-tree[45]

- Mutable
  - Dynamic Data Layout
    - One-d
      - ALEX[4]
      - MADEX[15]
      - Acc B+ Tree[26]
      - Slxndex[42]
      - PGM[7]
      - ASLM[25]
    - Multi-d
      - Fitting-tree[10]
      - Doraemon[38]
      - XIndex[39]
      - BF-Sandwich[30]
      - AIDEL[23]
      - IFB-Tree[14]
      - Hybrid-O[34]
      - Drift Model[13]

- Hands-off[16]

Indexing Learned Models

- SpatioTemporal
- Sequential

Benchmarking

- SOSD[19]
- Benchmark[28]
Discussion

Core Idea

● Apply the techniques in Flood to five other multi-dimensional indexes to answer spatial range queries.
  ○ Fixed-grid, Adaptive-grid, Kd-tree, Quadtree and STR

Major Insights

● Replace binary search with a learned index within each partition
  ○ Improve query execution time by 11.79% to 39.51%
● Filter on 1D using traditional index then refine using learned indexes
  ○ 1.23x to 1.83x times faster than methods that filter on 2D
● Learned indexes are more effective on queries with low selectivity (e.g., 0.00001%) but less effective on queries with high selectivity (e.g., 0.1%).

Hands-off Model Integration in Spatial Index Structures [AIDB@VLDB'20]

**Motivation**
- In-memory hierarchical trees require:
  - Excessive pointer-chasing
  - Time for chasing pointers impacts significantly the query execution time
- New approaches to design indexes are encouraged to utilize the modern hardware platforms

**Core Idea**
- Interpolation Friendly (IF) Indexes: IF-X
  - X is any multi-dimensional index
- Why Linear Interpolation?
  - Complex models have a higher capacity to fit the CDF
  - But complex models
    - Requires more parameters
    - Slower to compute
  - Linear interpolation is:
    - Simpler
    - Computationally inexpensive
  - Can eliminate expensive training process.

**Leaf Node Layout**
- IF-X indexes sort the records in each leaf node
  - Based on the best order using which the interpolation error is minimized.
- Store all required information in the header of the leaf node
  - No additional computation is needed
- The leaf node structure:
  - pDim: most predictable dimension which is used as the storage order

<table>
<thead>
<tr>
<th>No of records (#)</th>
<th>Max error (delta)</th>
<th>Prediction axis (pDim)</th>
<th>Slope (A)</th>
<th>Base (C)</th>
<th>Records sorted by pDim</th>
</tr>
</thead>
</table>

**Performance**
- Query execution time can be reduced by up to 60%
- Memory footprint can be reduced by over 90%

---

Motivation
- Build a disk-based learned multi-dimensional index for spatial queries.
- Support updates

Core Idea
- Representation of grid cells
- Mapping function:
  - \( M(\text{spatial keys}) \rightarrow \text{1D mapped values} \)
- Learned Shard Prediction Function:
  - \( SP(\text{mapped value}) \rightarrow \text{Shard Id} \)
    - Use ML models to generate searchable data layout in disk pages for arbitrary spatial dataset
- Local models:
  - Assign pages for all shards and perform intra-shard operations

Performance
- Outperforms traditional spatial indexes for range and KNN queries:
  - Memory consumption
  - IO cost

Effectively Learning Spatial Indices [VLDB’20]

**Introduction**

- Selecting grid resolution for Z-order for learned multi-dimensional index (e.g. ZM-Index[41]) is difficult:
  - Large cells
    - More false positives due to many points per cell
  - Small cells
    - Hard to learn due to uneven gaps in Cumulative Distribution Function (CDF)

**Motivation**

- Spatial index based on ordering the data points by a rank space-based transformation*
  - Simplify the indexing functions to be learned
  - \( M(\text{search keys}) \rightarrow \text{disk block Ids (location)} \)

- For scaling to large datasets, proposes:
  - Introduce a Recursive Spatial Model Index (RSMI) (in lieu of RMI)
  - Support point, window, and kNN queries
  - Support updates

---


Recursive Spatial Model Index (RSMI):
- Recursively partitions a dataset
- Partitioning is learned over the distribution of data
- Steps:
  - Initially distribute the data into equal sized partitions
  - Use a Space Filling Curve (SFC) to assign Ids to partitions
  - Learn the partition Ids using a model \( M_{0,0} \)
  - Rearrange the data based on the prediction of \( M_{0,0} \)
  - Recursively repartition
    - Until each partition can be learned with a simple model

Discussion
- Window and kNN query results are highly accurate but not exact.
  - i.e., over 87% across a variety of settings
  - Separate mechanism has been proposed for exact answer.
- Does not support query for spatial objects with non-zero extent

<table>
<thead>
<tr>
<th>Point</th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>p4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial partition Id</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Model predicted Id</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Learned partition Id</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Outline of the Tutorial

• Introduction and Taxonomy
• Indexing the Learned Models vs. Learning the Indexes
• Static vs. Dynamic Learned Indexes
• Fixed vs. Dynamic Data Layout
• Learned One-Dimensional Indexes
• Learned Multidimensional Indexes
• Open Problems and Future Research
• Traditional Indexes:
  • Theoretical guarantee on performance
  • Well studied and successfully integrated in real systems

• Learned Indexes:
  • Learn search-key distribution with some error correction mechanism
  • Better performance with less space requirement

• Hybrid Indexes:
  • Optimizing traditional indexes with helping (e.g., ML) models
Some Open Problems

• Efficiently support Inserts/Updates
• Support for other spatial operations, e.g., KNN, spatial join, closest pairs
• What types of ML models to use?
• Integrate with real database engines
• Concurrency support
• Develop benchmark for Learned Multidimensional Indexes
Evolution of Learned Indexes
References


References


References


References


• [27] Stephen Macke, Alex Beutel, Tim Kraska, Maheswaran Sathiamoorthy, Derek Zhiyuan Cheng, and EH Chi. 2018. Lifting the curse of multidimensional data with learned existence indexes. In Workshop on ML for Systems at NeurIPS.


References


References


Q&A

Website of Tutorial: https://www.cs.purdue.edu/homes/aref/learned-indexes-tutorial.html