Machine Learning for Data Management Systems

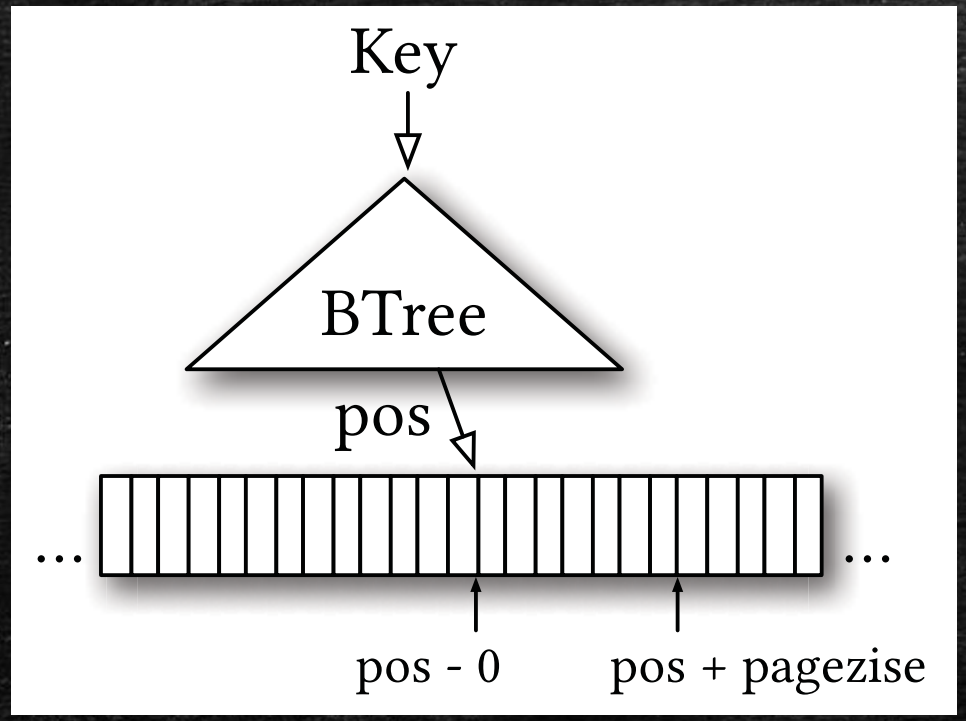
Learned Indexes

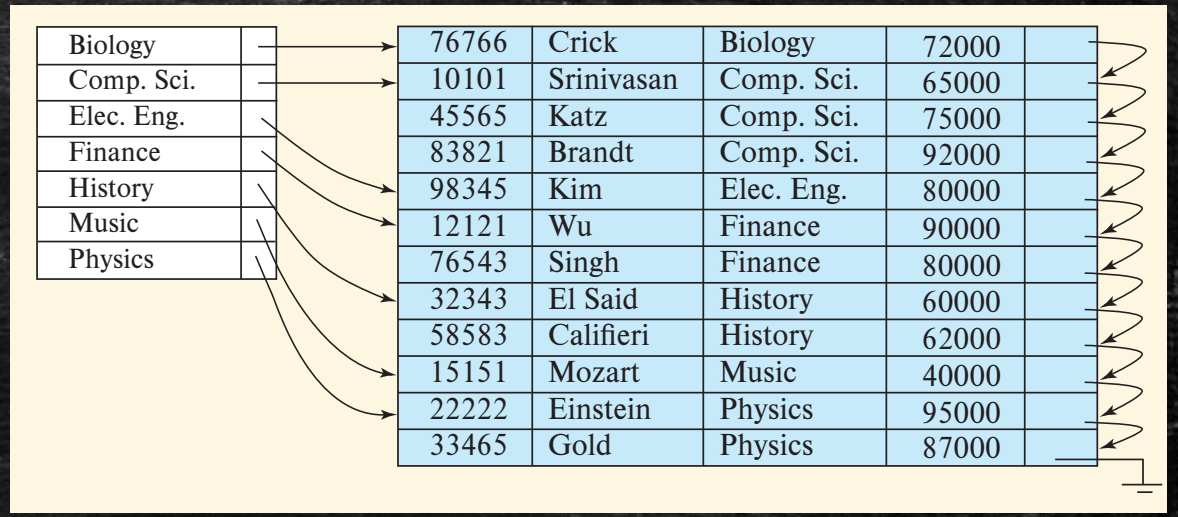
Amol Deshpande
February 2, 2023

Outline

- **The Case for Learned Indexes**
- Follow-up Work
- Discussion

Indexes as Prediction Models !

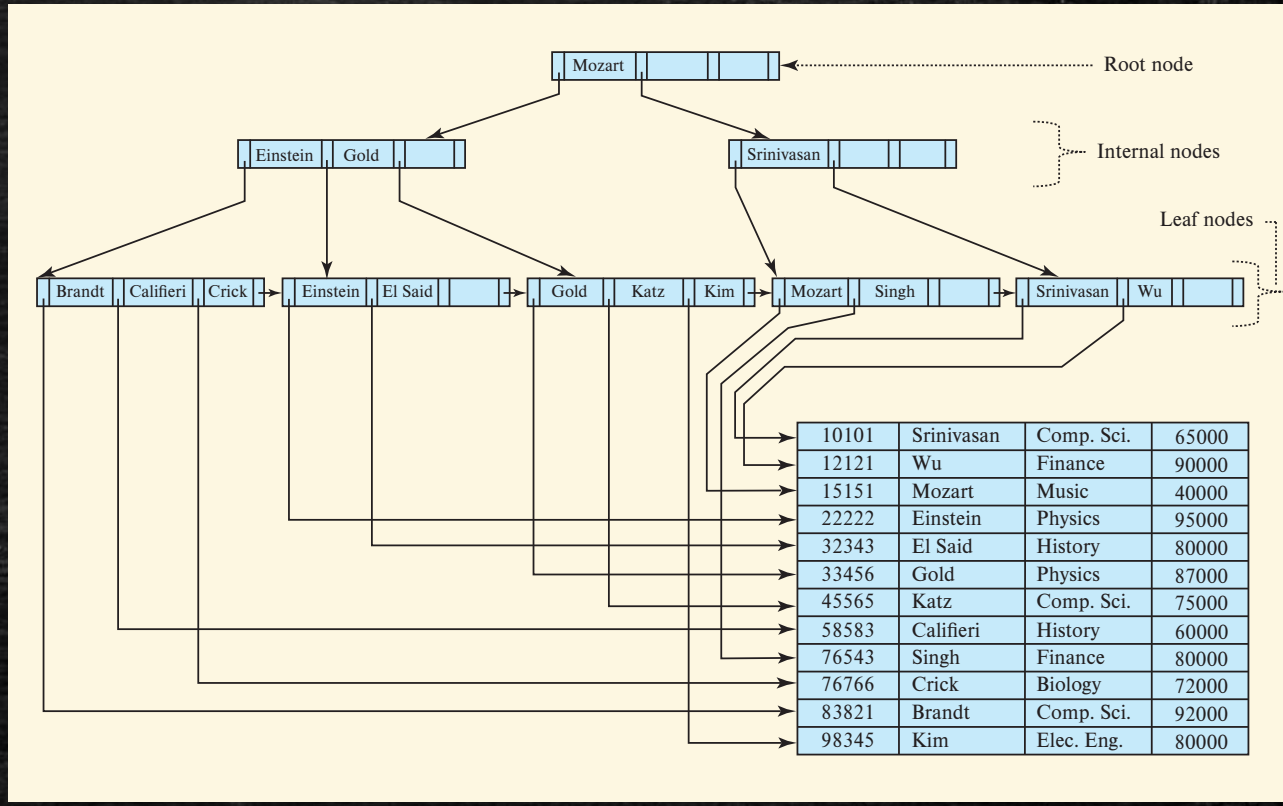




Basically a function that predicts the position
with some error bounds
(assuming sorted order, i.e., primary index)

Indexes as Prediction Models !

- Doesn't work as well for secondary indexes
  - Multiple positions per search key
  - Also, no tolerance for errors
  - But can do the prediction on the lowest layer
    - i.e., predict the leaf that contains the pointers to the data



Bitmap Indexes

- Widely used in Data Warehouses
  - Allow quickly finding tuples that satisfy a predicate/property
- Can be seen as a classification function
- Approximations through Bloomfilters
  - Next class

record number	*ID*	*gender*	*income_level*
0	76766	m	L1
1	22222	f	L2
2	12121	f	L1
3	15151	m	L4
4	58583	f	L3

Bitmaps for *gender*

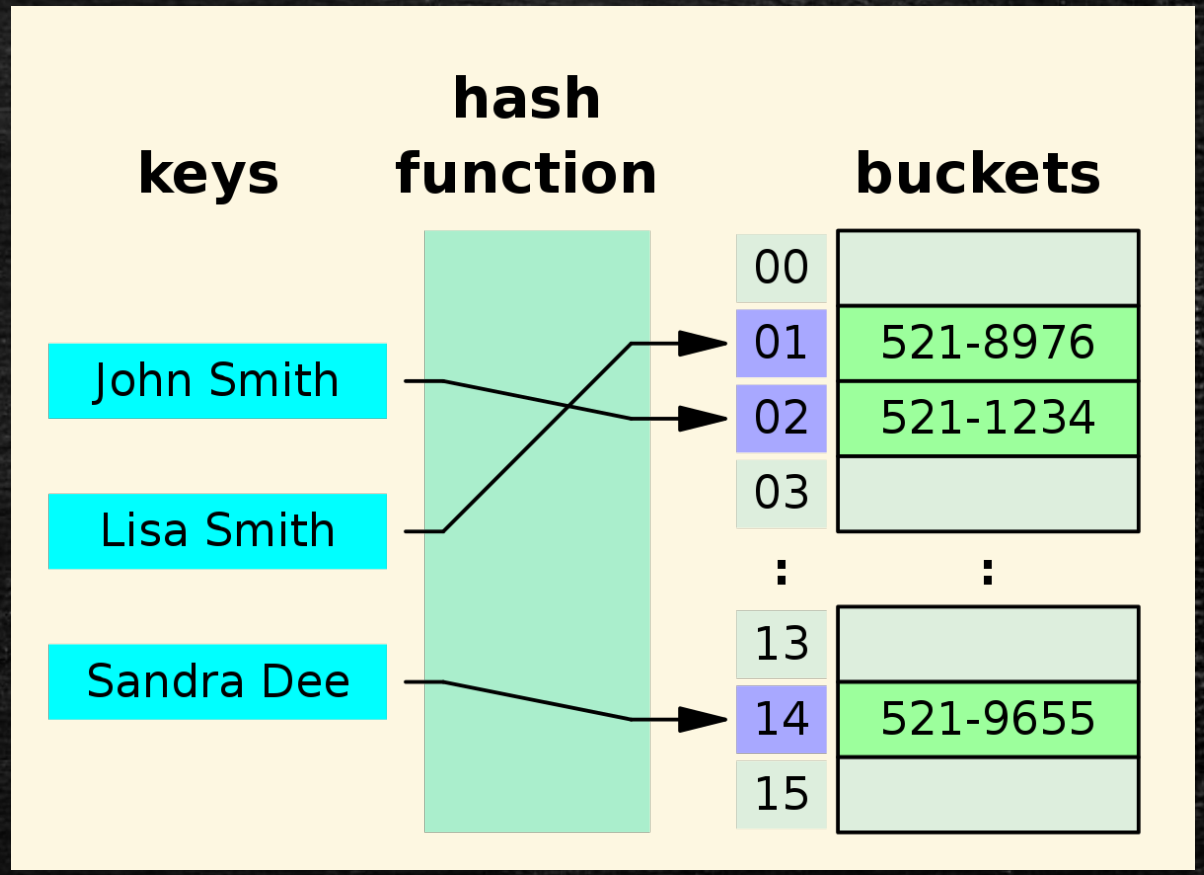
m	10010
f	01101

Bitmaps for *income_level*

L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

Hash Indexes

- Predict the "bucket" in which a key would fall
- Not tolerant to errors



Learned "range" indexes

- Replace the prediction model with a ML model

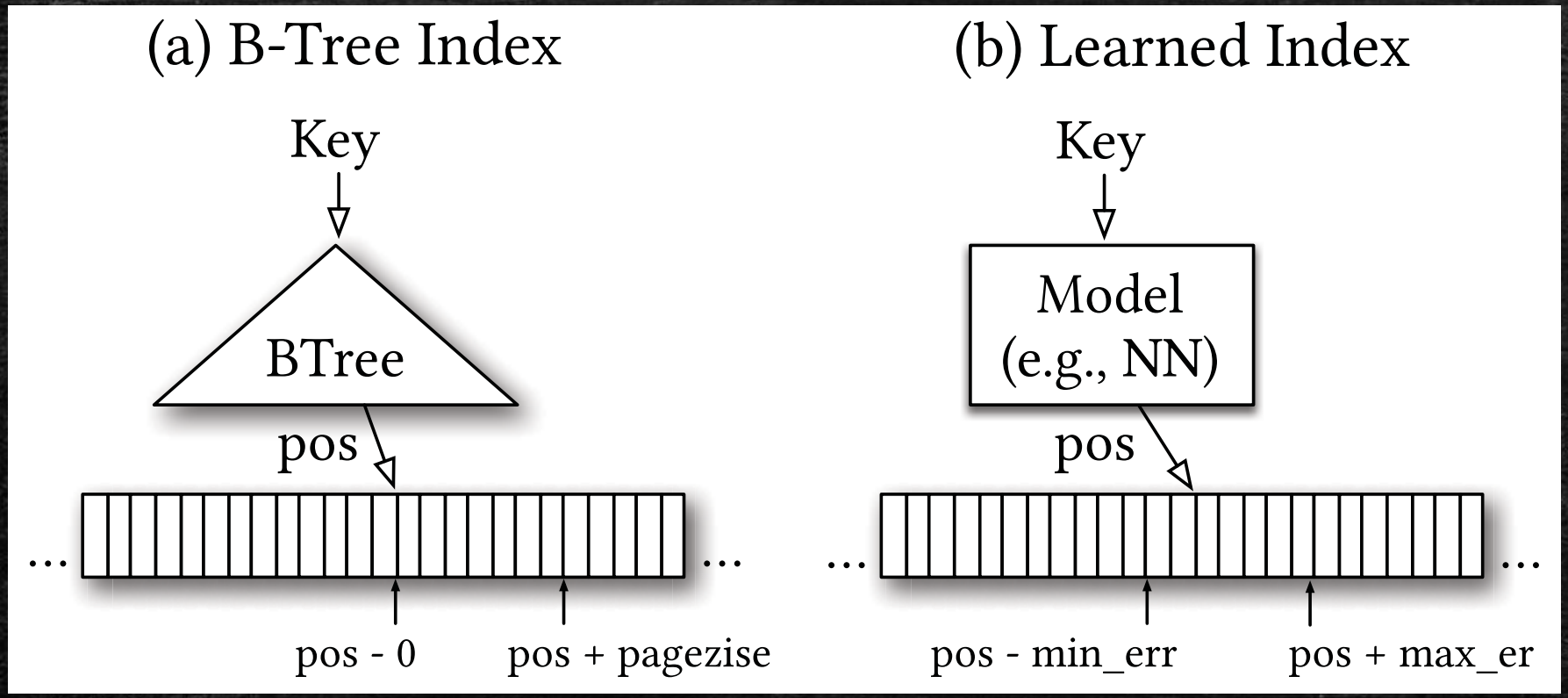


Figure 1: Why B-Trees are models

Learned "range" indexes

- Alternate view: We are trying to learn a CDF
  - Given a key K, we want to know *prob(a random value < K)*
  - CDF(K) * N gives us the position of the search key K

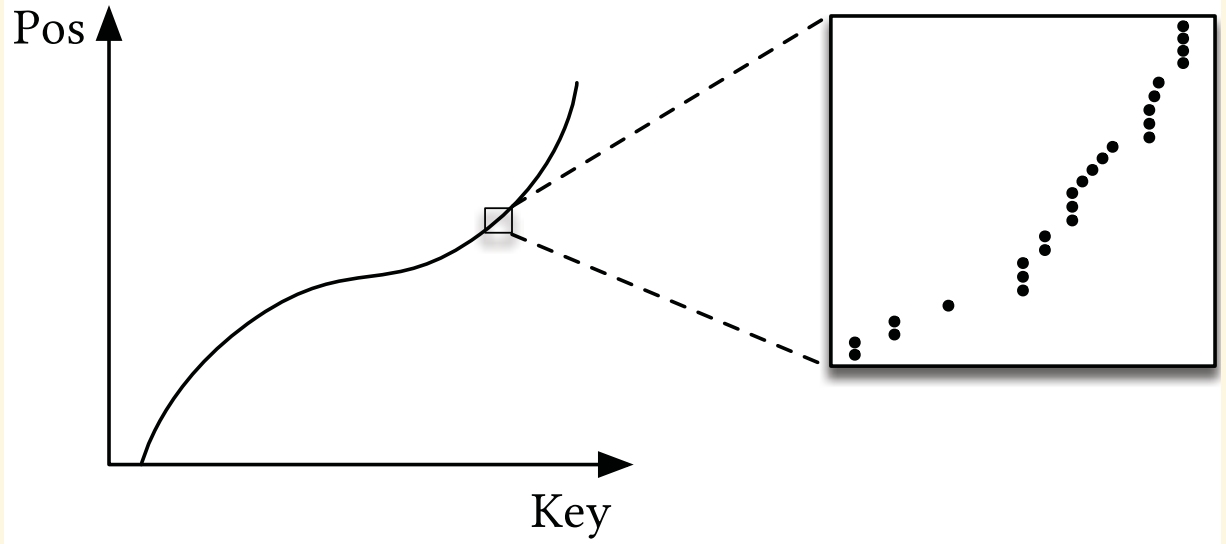


Figure 2: Indexes as CDFs

Another Example

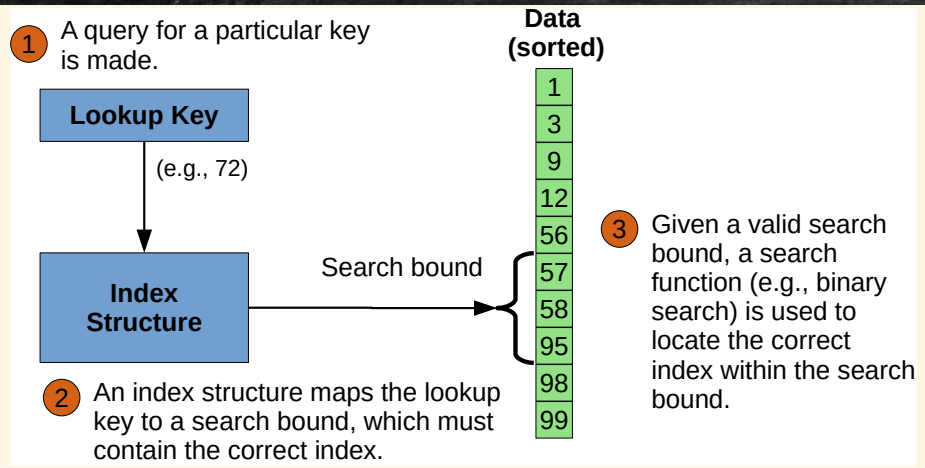


Figure 1: Index structures map each lookup key to a *search bound*. This search bound must contain the "lower bound" of the key (i.e., the smallest key greater than or equal to the lookup key). The depicted search bound is valid for the lookup key 72 because the key 95 is in the bound. A search function, such as binary search, is used to locate the correct index within the search bound.

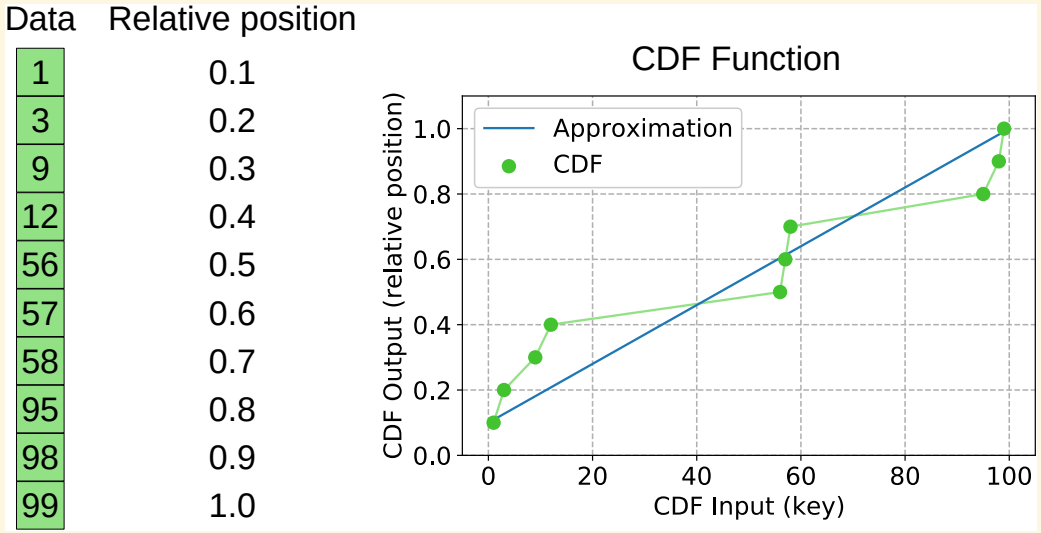


Figure 2: The cumulative distribution function (CDF) view of a sorted array.

Benchmarking Learned Indexes; Marcus et al.; PVLDB 2021

Learned "range" indexes

- Replace the prediction model with a ML model
- Should be a "monotone" model
  - i.e., if x > y, then f(x) >= f(y)
  - Otherwise hard to handle keys that don't exist in the data
- Other challenges/issues
  - B+-Trees don't require the leaf level nodes to be contiguous
  - Most indexes have bounded insert/delete/update costs
  - B+-Trees work well with caches
    - Top 2 levels often in the cache

A First Attempt

- Use a single 2-layer NN on a 200M web-server log records
- 1250 predictions per second
  - Much slower than B+-Trees (about 300x slower)
- Some notes
  - Tensorflow not designed to run small models – too much overhead
  - B-Trees do a really good job on small amounts of data
- Raw performance improvements in GPU likely not enough to make up the gap

Recursive Model Index

- Use a hierarchy of models instead
- Can use different models in different places, including decision trees (i.e., B+-trees)

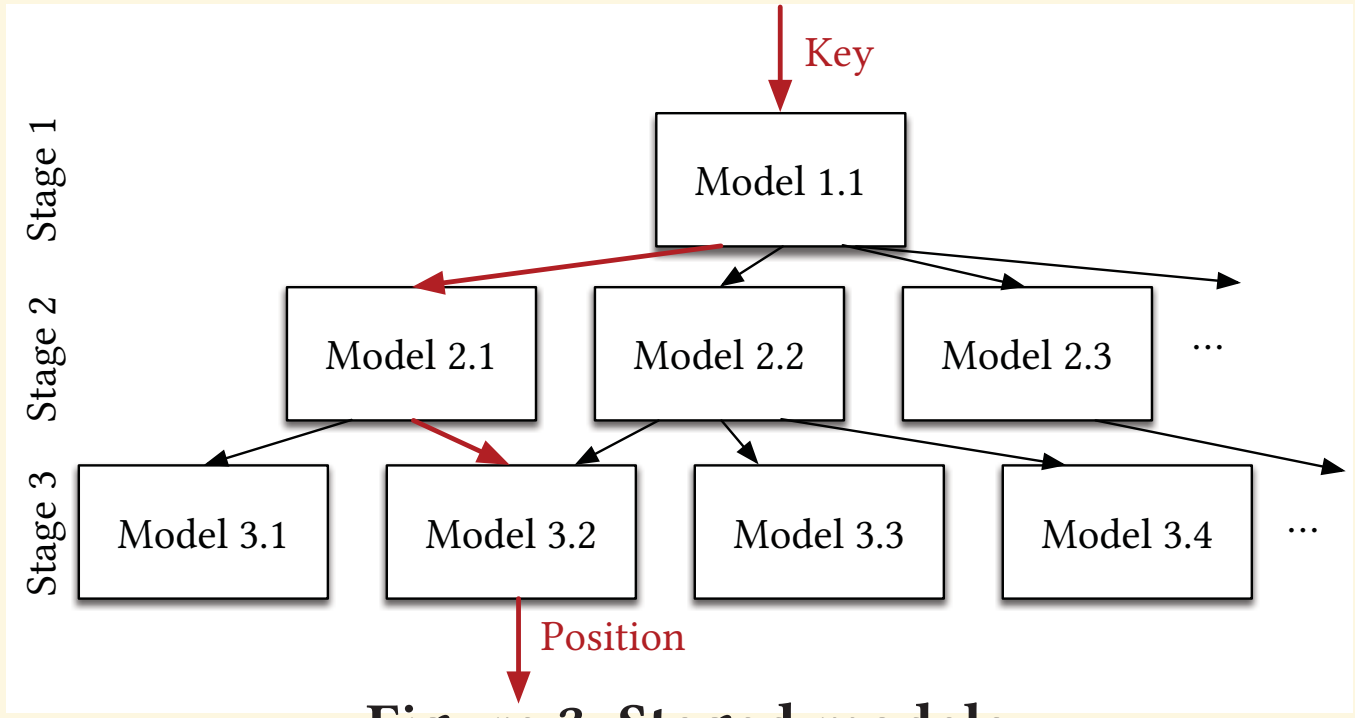


Figure 3: Staged models

Recursive Model Index

- Works much better than B+-Trees across several datasets using fairly simple models
  - Linear, or 1-2 layer NNs (for the first stage)

		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%)

Learned Hash Index

- Learn a hash function with the goal to minimize conflicts
  - Less focus on this in the follow-up work

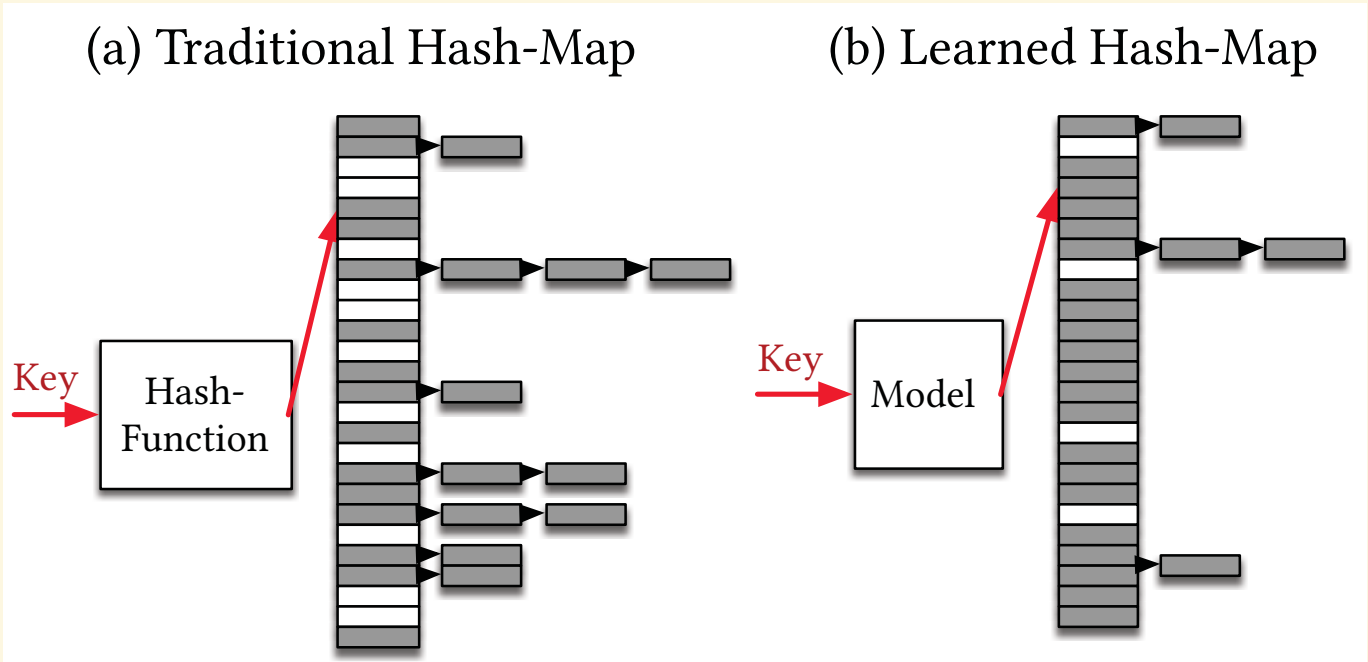


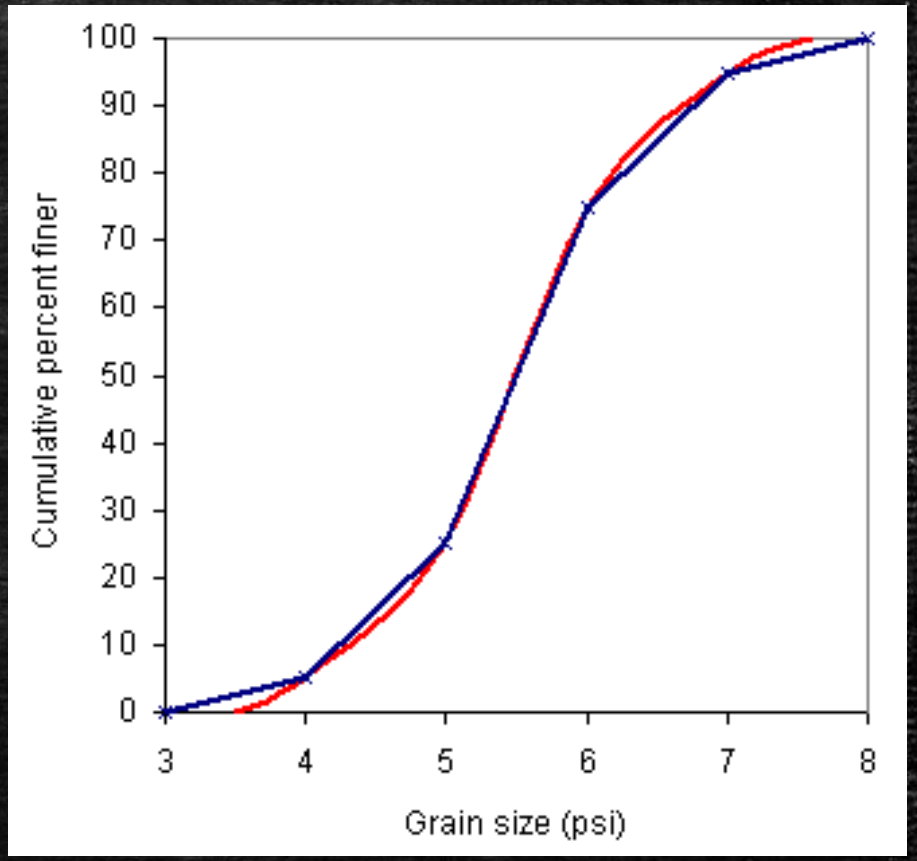
Figure 7: Traditional Hash-map vs Learned Hash-map

Outline

- The Case for Learned Indexes
- **Follow-up Work**
- Discussion

Spline Interpolation

- CDF can be approximated using splines
  - i.e., a collection of linear segments
- How about:
  - Increase the page size for the B+-Tree
    - So many more pointers/data values in each page
    - Much smaller index
  - Use Interpolation within each page
    - Instead of binary search
- Results appear to match learned index
- Much discussion on the link



http://databasearchitects.blogspot.com/2017/12/the-case-for-b-tree-index-structures.html

RadixSpline: A Single-Pass Learned Index

- Faster learning in a read-only setting
- Steps:
  - Learn a linear spline over the data
  - Build a small index on top using first few bits of the search key
- Results:
  - Generally comparable latencies, but higher storage space for some datasets

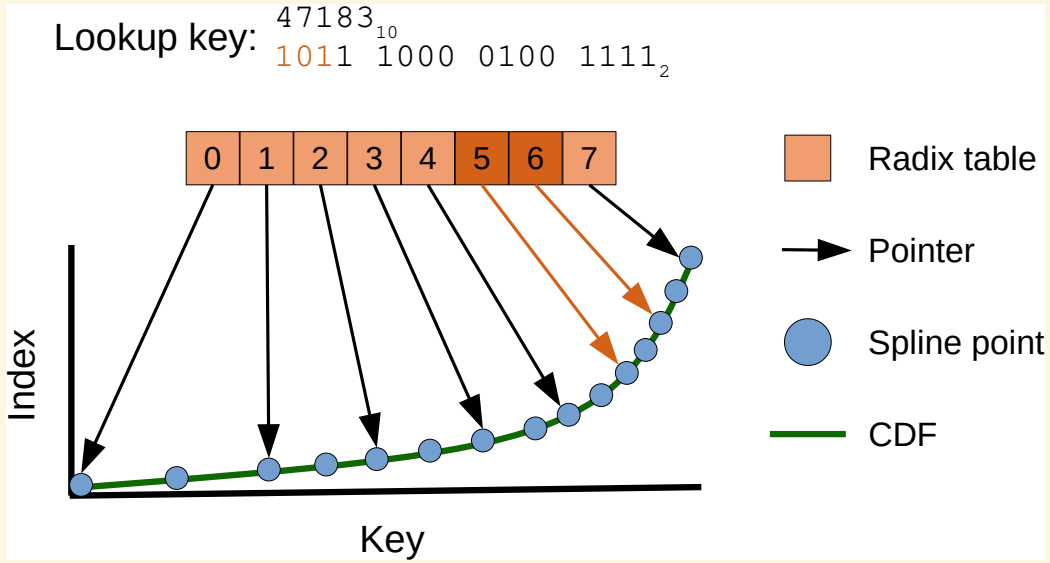


Figure 1: A radix spline index and example lookup process. The r (here, 3) most significant bits b of the lookup key are used as an index into the radix table. Then, a binary search is performed on the spline points between the bth pointer and the $b+1$th pointer.

Kipf et al.; aiDM 2020 (SIGMOD Workshop)

ALEX: Updatable Learned Index

- Goals:
  - Inserts competitive with B+-Trees
  - Lookups faster than B+-Trees or RMI
  - Better storage space
  - Data storage space (leaf level) comparable with dynamic B+Trees
- Key ideas:
  - Have gaps in the leaves for new data
  - Use "exponential search" at the leaves (instead of binary)
  - Allow individual nodes to shrink/grow as needed (only works in memory)

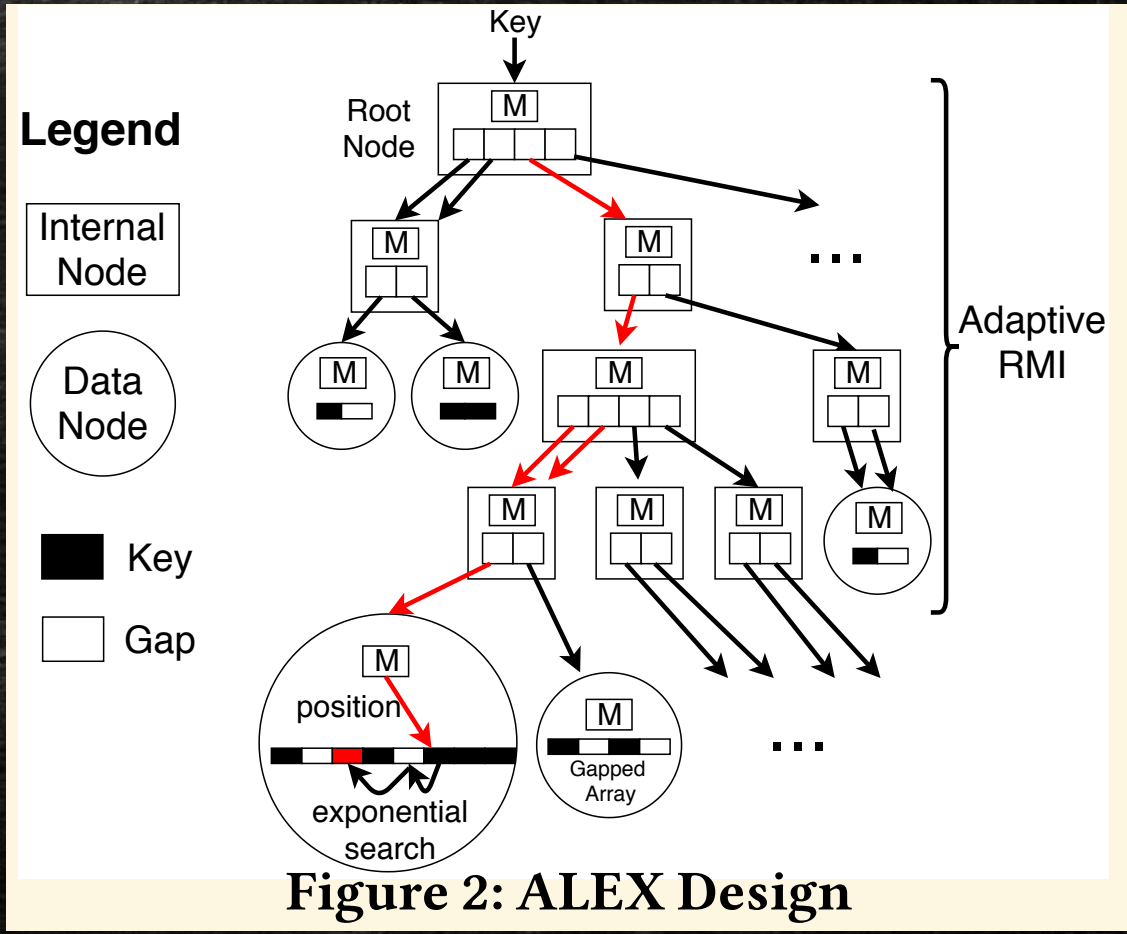
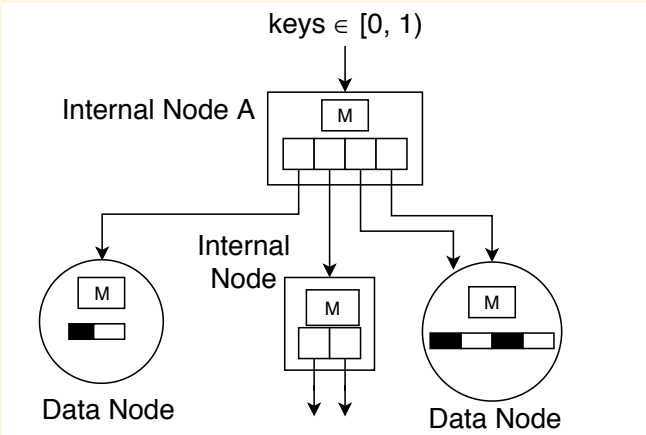


Figure 2: ALEX Design

Ding et al.; SIGMOD 2020

ALEX: Updatable Learned Index

- Some details:
  - Leaves have arrays with gaps in between (i.e., not just at the end)
  - Use a bitmap to skip over gaps
  - Allow a node to have a mix of internal and leaf children



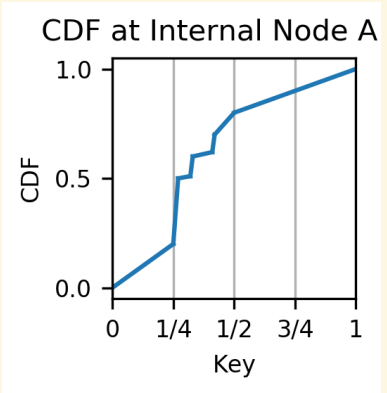


Figure 3: Internal nodes allow different resolutions in different parts of the key space $[0,1)$.

Ding et al.; SIGMOD 2020

And more…

- PGM-Index: Built in a bottom-up fashion, fully dynamic
  - Ferragina, Vinciguerra; PVLDB 2020
  - Piecewise linear regressions (instead of spline)

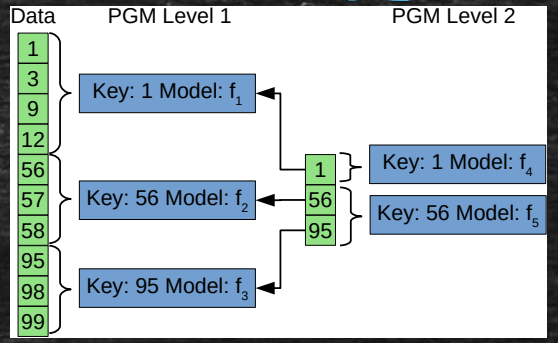
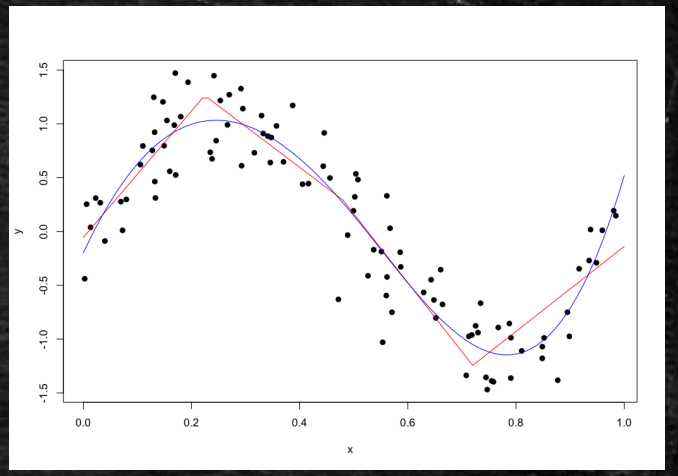
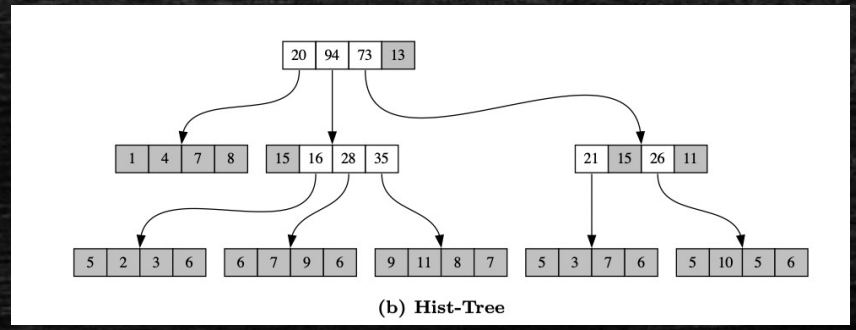


Figure 5: A piecewise geometric model (PGM) index.



- Hist-tree: More similar to B+-Trees
  - Crotty; CIDR 2021
  - Followup experiments: works very well but requires a lot of storage



(b) Hist-Tree

Benchmarking Learned Indexes

Method	Updates	Ordered	Type
PGM [13]	Yes	Yes	Learned
RS [18]	No	Yes	Learned
RMI [19]	No	Yes	Learned
BTree [7]	Yes	Yes	Tree
IBTree [15]	Yes	Yes	Tree
FAST [16]	No	Yes	Tree
ART [20]	Yes	Yes	Trie
FST [33]	Yes	Yes	Trie
Wormhole [31]	Yes	Yes	Hybrid hash/trie
CuckooMap [6]	Yes	No	Hash
RobinHash [3]	Yes	No	Hash
RBS	No	Yes	Lookup table
BS	No	Yes	Binary search

Table 1: Search techniques evaluated

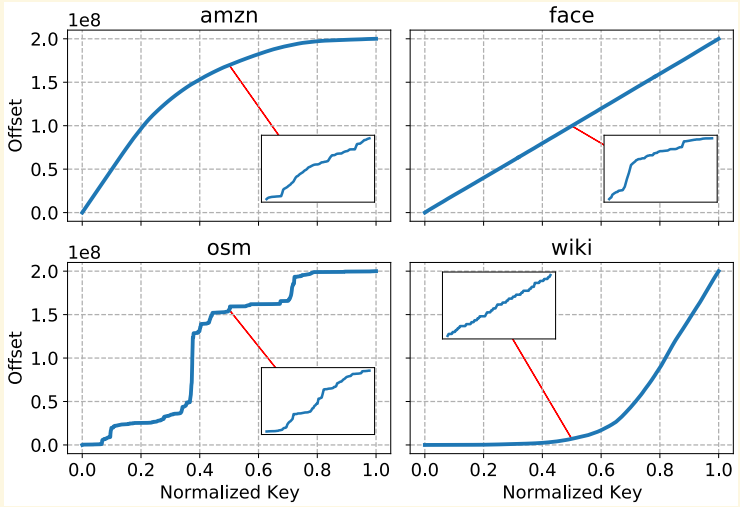


Figure 6: CDF plots of each testing dataset. The `face` dataset contains ≈ 100 large outlier keys, not plotted.

Marcus et al.; PVLDB 2021

Benchmarking Learned Indexes

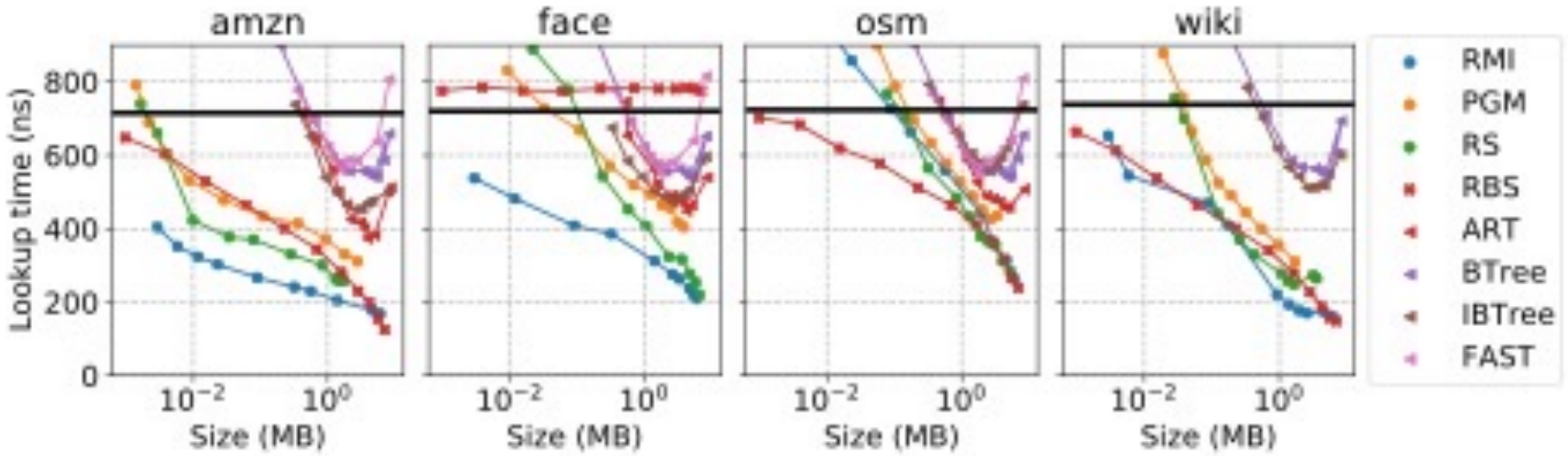


Figure 7: Performance and size tradeoffs provided by several index structures for four different datasets. The black horizontal line represents the performance of binary search (which has a size of zero). Extended plots with all techniques are available here: `https://rm.cab/lis1`

Marcus et al.; PVLDB 2021

Outline

- The Case for Learned Indexes
- Follow-up Work
- **Discussion**

Some Thoughts

- Relatively simple models being used by the recent work
- Are these helping with autotuning, simplicity, etc.