Non-Standard-Datenbanken und Data Mining

Learned Index Structures

Prof. Dr. Ralf Möller
Universität zu Lübeck
Institut für Informationssysteme

Acknowledgements

Paper by Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, Neoklis Polyzotis

- Presentations taken from 2 talks by
  Deniz Altinbuken and John Yang (CS 294, 2019), resp.
- Presentations are possibly adapted or extended
- All faults are mine

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In Proc. of the 2018 International Conference on Management of Data (SIGMOD '18). **2018.**

Main Idea

Assumption:

- A model can be learned from specific data to encode the sort order of lookup keys

Goal:

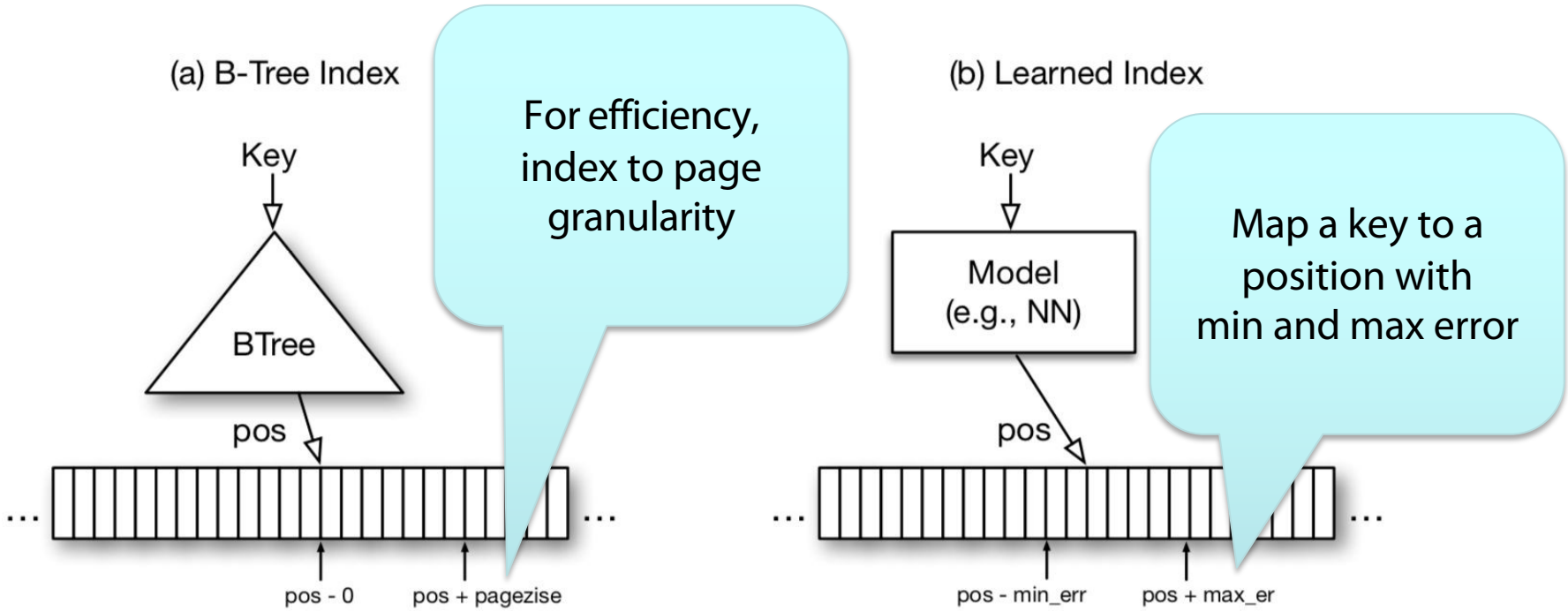
- The model is to be used to effectively predict the position or existence of records, such that …
- … data records can be found approximately or with minimal search (in case completeness matters)

Recap: Index Structures

- Predict the location of a value given a key
  - B-tree (range index)
    - Model that takes a key as input and predicts the position of the first data record in a range such that the specified range of records is found with minimal search (scan)
  - Hashmap
    - Model that takes a key as input and predicts the position of a single data record (search in case of collisions)
  - Bloom filter
    - Binary classifier model, which, given a key, predicts if a key exists in a set or not (but with false positives)

B-Tree

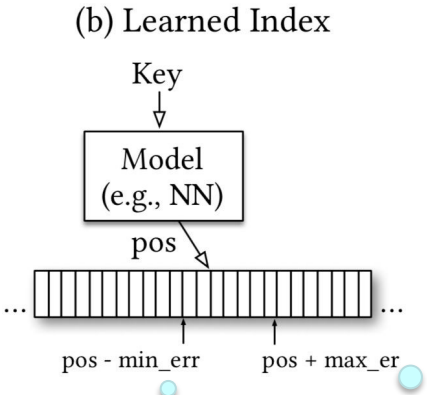
The B-tree provides a mapping from a lookup key into a position inside the sorted array of records (in case of a so-called clustered b-tree) or to the first page on disk to access

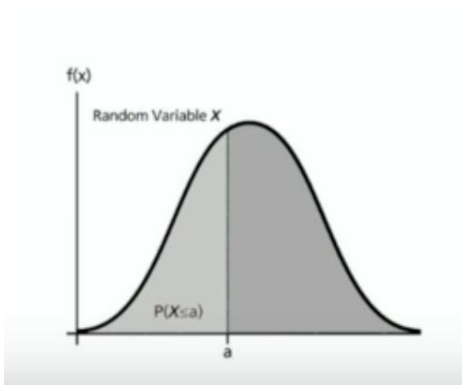


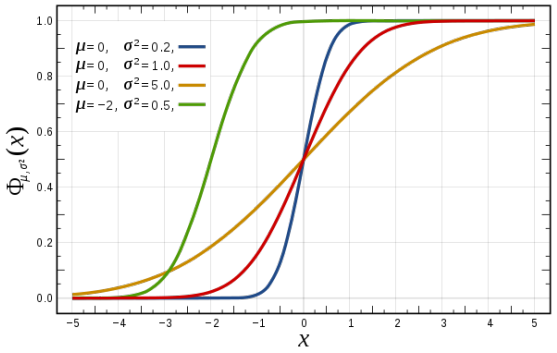
Key Idea

B-Trees as a Cumulative Distribution Function

Predicted Position = P(x ≤ key) * # of Keys





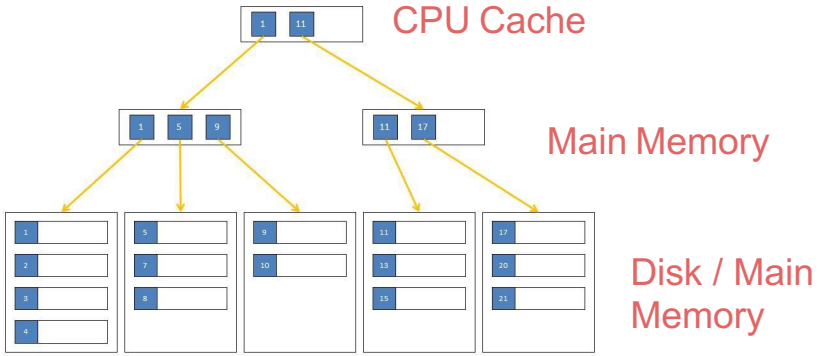


If completeness required, scan starts too early: more overhead

Need to scan to the left as well as to the right in order to be complete (doubly linked list of pages required)

Challenges

- Replace index structures such as B-trees with ML models providing guarantees about min and max error
- Retain properties of B-trees
  - Bounded cost for inserts and lookups
  - Taking advantage of caches
  - Map keys to pages that are not necessarily continuously mapped to memory or disk



ML-derived models

- Using ML models has the potential to transform log n B-tree look-up costs into constant costs (in the best case)
- For instance, networks are able to learn a variety of multidimensional data distributions, mixtures and other data peculiarities
  - Distributions allow for the estimation of positions
  - Error estimations, e.g., by Markov, Chebyshev, Hoeffding, or Chernoff inequations
- Balance the complexity of a model with its accuracy and the complexity of acquiring the model

First Approach

Tensorflow implementation of B-Tree-like index

- 200M Web Server Log Records sorted by Timestamp
- 2 layer network, 32-width fully connected, ReLU activation function
- Given the timestamp, predict the position!

Results:

- Tensorflow: 1250 Predictions / Sec ~ 80000 ns Lookup
- B-Trees: 300 ns Lookup,
- 900 ns w/ binary search across entire data set

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In Proc. of the 2018 International Conference on Management of Data (SIGMOD '18). **2018.**

The Recursive Model Index

Improve last-mile accuracy

- Reducing min/max error to 100 from 100M records using a single model is very hard
- Reducing the error to 10k from 100M is much easier to achieve even with simple models
- Reducing the error from 10k to 100 is simpler as the model can focus only on a subset of the data

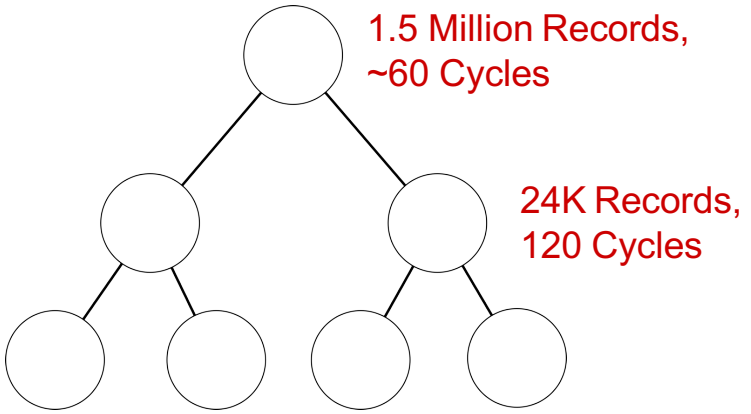
Use a hierarchical approach where we can have models focus on smaller subsets of data.

Recursive Model Index (RMI)

Problem: Accuracy of Last Mile Search

Solution: Recursive Regression Model

- Idea: Reduce error across a hierarchy of models focusing on subsets of data



More formally, for our model $f(x)$ where x is the key and $y \in [0, N)$ the position, we assume at stage ℓ there are M_ℓ models. We train the model at stage 0, $f_0(x) \approx y$. As such, model k in stage ℓ, denoted by $f_\ell^{(k)}$, is trained with loss:

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

$$L_0 = \sum_{(x,y)} \left(f_0(x) - y\right)^2$$

Initialization

Note, we use here the notation of $f_{\ell-1}(x)$ recursively executing $f_{\ell-1}(x) = f_{\ell-1}^{(\lfloor M_{\ell-1} f_{\ell-2}(x)/N \rfloor)}(x)$. In total, we iteratively train each stage with loss L_ℓ to build the complete model.

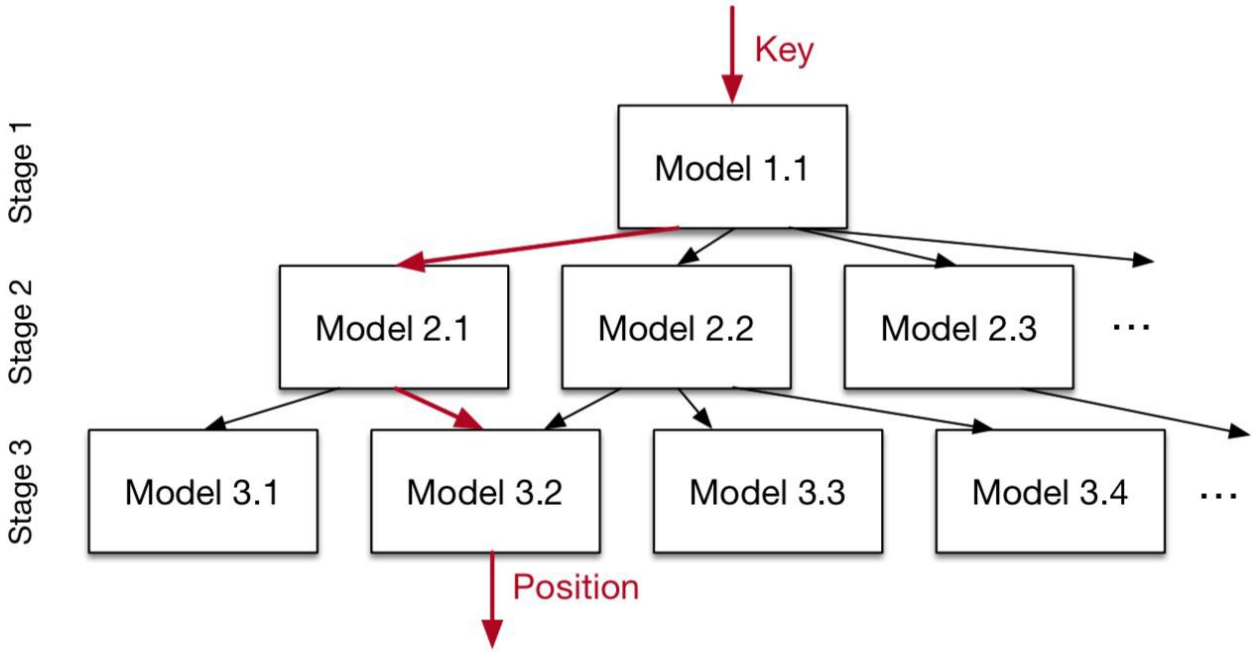
The Recursive Model Index

Take a layered approach and have models focus on limited layers:

Reduce from 100M to 1M

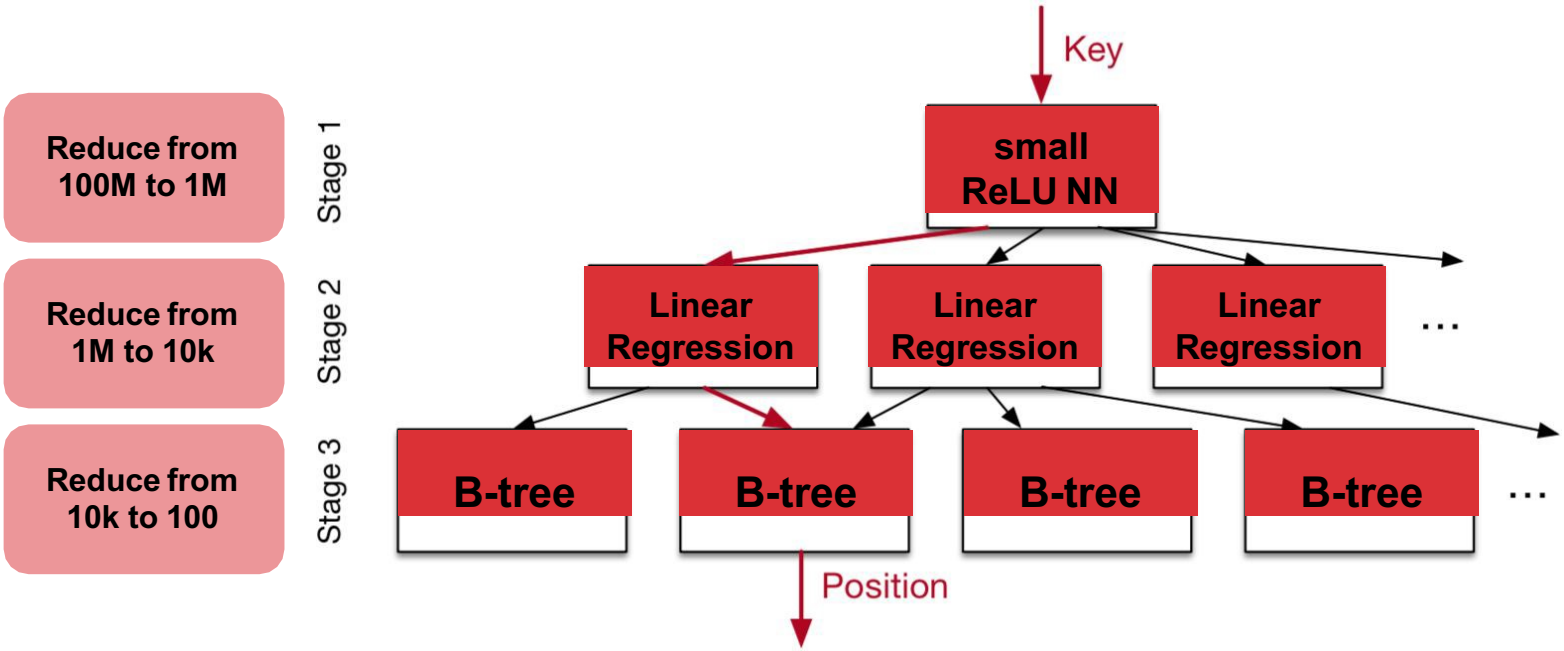
Reduce from 1M to 10k

Reduce from 10k to 100



Hybrid End-to-End Training

With a layered approach we can build mixtures of models!



Hybrid Recursive Model Index

Problem: Specific data at the bottom of RMI may be harder to learn

Solution: Combine different models at different layers of RMI

- Networks with ReLU activations at the top
- Simple linear regression in the middle
- Fall back on B-Trees if data is particularly difficult to learn

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In Proc. of the 2018 International Conference on Management of Data (SIGMOD '18). **2018.**

Search Strategies

To find the record either binary search or scanning is used

Models might generate more information than page location

- Model Binary Search
  - Set first *middle* point to pos predicted by the model.
- Biased Search
  - Use standard deviation σ of the last stage model to set *middle*.
- Biased Quaternary Search
  - Pick three *middle* points as $pos - \sigma$, pos, $pos + \sigma$.

Indexing strings

- Turn strings into inputs that the network model can use
  - Represent string as a vector, where each element is the decimal ASCII value of a char
  - Limit size of vector to N to have equally-sized inputs
- Vector inputs slow the model down significantly

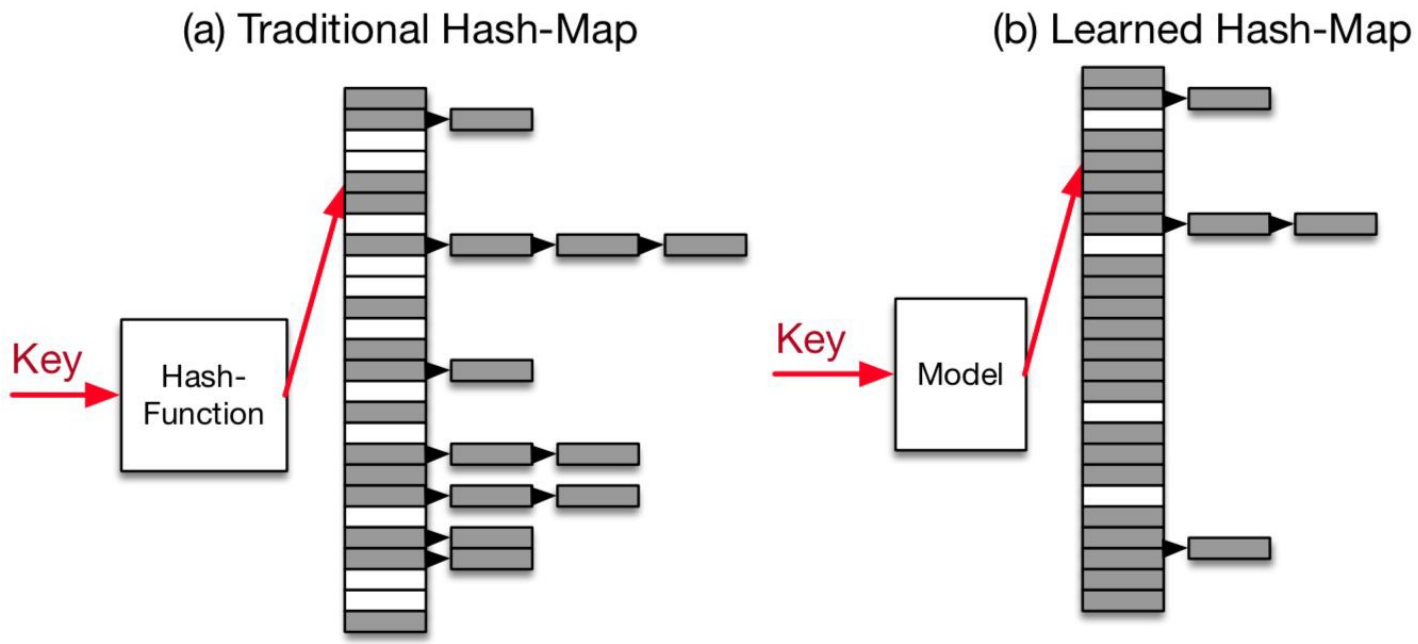
- Further research is needed to speed up this case :)

Inserts and Updates

- Appends
  - No need to relearn if the model is able to learn the key trend for the new items
- Inserts in the middle
  - If inserts follow roughly a similar pattern as the learned CDF, retraining is not needed since the index "generalizes" over the new items and inserts become an O(1) operation.

Hashmap

Hashmaps use a hash function to deterministically map keys to random positions inside an array



Hashmap

Main challenge is to reduce conflicts

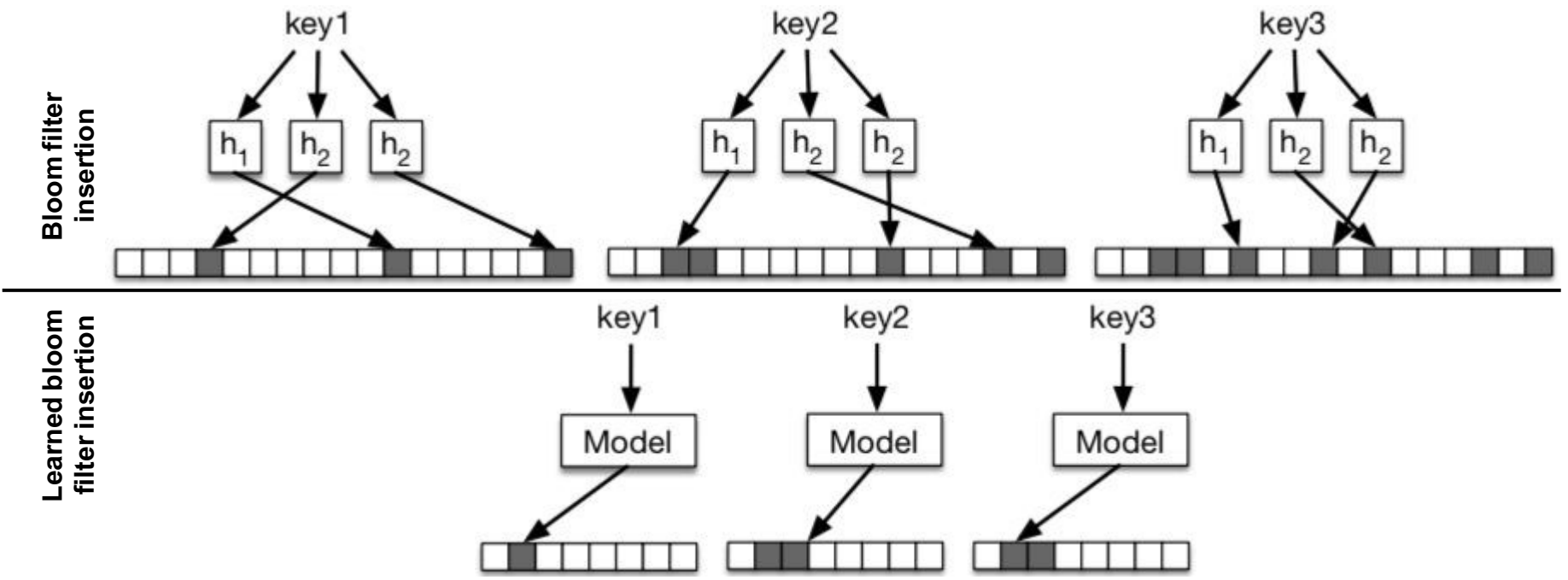
- Use a linked-list to handle the “overflow”
- Use linear or quadratic probing
- Most solutions allocate significantly more memory than records and combine it with additional data structures

Hashmap

- If we could learn a model which uniquely maps every key into a unique position inside the array, we could avoid conflicts
- Learned models are capable of reaching higher utilization of the hashmap depending on the learnt data distribution
- Scale the distribution by the targeted size M of the hashmap and use $h(K) = F(K) * M$, K is key
- If the model F perfectly learned the distribution, no conflicts would exist

Bloom Filters and Learned Hashmaps

Bloom filters are probabilistic data structures used to test whether an element is a member of a set.



Bloom Filter

- A bloom filter index needs to learn a function that separates keys from everything else
  - A good hash function for a bloom filter can have lots of collisions among keys and lots of collisions among non-keys, but few collisions of keys and non-keys

Bloom Filter

As a classification problem: learn a model *f* that can predict if an input *x* is a key or non-key

- Use sigmoid activation function to find probability value in [0,1]
- The output is the probability that input *x* is a key in our database
- Choose a threshold *t* above which we will assume the key exists in our database
- Tune threshold *t* to achieve the desired *false positive rate*
- To prevent *false negatives*, use overflow bloom filter

B-tree Results

- 4 datasets to compare the performance of learned index structures with B-trees
  - Compare lookup-time (model execution time + local search time)
  - Compare index structure size
  - Compare model error and error variance
- These results focus on read performance only, loading and insertion time are not included
  - A model without hidden layers can be trained on over 200M records in just few seconds

Web Log Dataset

- 200M log entries for requests to a major university website.
- Index over all unique timestamps.

The **model error** is the averaged standard error over all models on the last stage, whereas the **error variance** indicates how much this standard error varies between the models

Baseline

Type	Config	Search	Total (ns)	Model (ns)	Search (ns)	Speedup	Size (MB)	Size Savings	Model Err ± Err Var.
Btree	page size: 16	Binary	285	234	51	9%	103.86	700%	4 ± 0
	page size: 32	Binary	276	201	75	6%	51.93	300%	16 ± 0
	page size: 64	Binary	274	171	103	5%	25.97	100%	32 ± 0
	page size: 128	Binary	260	132	128	0%	12.98	0%	64 ± 0
	page size: 256	Binary	266	114	152	2%	6.49	-50%	128 ± 0
Learned Index	2nd stage size: 10,000	Binary	222	29	193	-15%	0.15	-99%	242 ± 150
		Quaternary	224	29	195	-14%	0.15	-99%	242 ± 150
	2nd stage size: 50,000	Binary	162	36	126	-38%	0.76	-94%	40 ± 27
		Quaternary	157	36	121	-40%	0.76	-94%	40 ± 27
	2nd stage size: 100,000	Binary	144	39	105	-45%	1.53	-88%	21 ± 14
		Quaternary	138	38	100	-47%	1.53	-88%	21 ± 14
	2nd stage size: 200,000	Binary	126	41	85	-52%	3.05	-76%	12 ± 7
		Quaternary	122	39	83	-53%	3.05	-76%	12 ± 7
Learned Index Complex	2nd stage size: 100,000	Binary	218	89	129	-16%	1.53	-88%	4218 ± 15917
		Quaternary	213	91	122	-18%	1.53	-88%	4218 ± 15917

Web Log Dataset

Model is 3 × faster and up to an order-of-magnitude smaller.

Type	Config	Search	Total (ns)	Model (ns)	Search (ns)	Speedup	Size (MB)	Size Savings	Model Err ± Err Var.
Btree	page size: 16	Binary	285	234	51	9%	103.86	700%	4 ± 0
	page size: 32	Binary	276	201	75	6%	51.93	300%	16 ± 0
	page size: 64	Binary	274	171	103	5%	25.97	100%	32 ± 0
	page size: 128	Binary	260	132	128	0%	12.98	0%	64 ± 0
	page size: 256	Binary	266	114	152	2%	6.49	-50%	128 ± 0
Learned Index	2nd stage size: 10,000	Binary	222	29	193	-15%	0.15	-99%	242 ± 150
		Quaternary	224	29	195	-14%	0.15	-99%	242 ± 150
	2nd stage size: 50,000	Binary	162	36	126	-38%	0.76	-94%	40 ± 27
		Quaternary	157	36	121	-40%	0.76	-94%	40 ± 27
	2nd stage size: 100,000	Binary	144	39	105	-45%	1.53	-88%	21 ± 14
		Quaternary	138	38	100	-47%	1.53	-88%	21 ± 14
	2nd stage size: 200,000	Binary	126	41	85	-52%	3.05	-76%	12 ± 7
		Quaternary	122	39	83	-53%	3.05	-76%	12 ± 7
Learned Index Complex	2nd stage size: 100,000	Binary	218	89	129	-16%	1.53	-88%	4218 ± 15917
		Quaternary	213	91	122	-18%	1.53	-88%	4218 ± 15917

Web Log Dataset

Quarternary search only helps a little bit.

Type	Config	Search	Total (ns)	Model (ns)	Search (ns)	Speedup	Size (MB)	Size Savings	Model Err ± Err Var.
Btree	page size: 16	Binary	285	234	51	9%	103.86	700%	4 ± 0
	page size: 32	Binary	276	201	75	6%	51.93	300%	16 ± 0
	page size: 64	Binary	274	171	103	5%	25.97	100%	32 ± 0
	page size: 128	Binary	260	132	128	0%	12.98	0%	64 ± 0
	page size: 256	Binary	266	114	152	2%	6.49	-50%	128 ± 0
Learned Index	2nd stage size: 10,000	Binary	222	29	193	-15%	0.15	-99%	242 ± 150
		Quaternary	224	29	195	-14%	0.15	-99%	242 ± 150
	2nd stage size: 50,000	Binary	162	36	126	-38%	0.76	-94%	40 ± 27
		Quaternary	157	36	121	-40%	0.76	-94%	40 ± 27
	2nd stage size: 100,000	Binary	144	39	105	-45%	1.53	-88%	21 ± 14
		Quaternary	138	38	100	-47%	1.53	-88%	21 ± 14
	2nd stage size: 200,000	Binary	126	41	85	-52%	3.05	-76%	12 ± 7
		Quaternary	122	39	83	-53%	3.05	-76%	12 ± 7
Learned Index Complex	2nd stage size: 100,000	Binary	218	89	129	-16%	1.53	-88%	4218 ± 15917
		Quaternary	213	91	122	-18%	1.53	-88%	4218 ± 15917

Web Log Dataset

The error is high, which influences the search time.

Type	Config	Search	Total (ns)	Model (ns)	Search (ns)	Speedup	Size (MB)	Size Savings	Model Err ± Err Var.
Btree	page size: 16	Binary	285	234	51	9%	103.86	700%	4 ± 0
	page size: 32	Binary	276	201	75	6%	51.93	300%	16 ± 0
	page size: 64	Binary	274	171	103	5%	25.97	100%	32 ± 0
	page size: 128	Binary	260	132	128	0%	12.98	0%	64 ± 0
	page size: 256	Binary	266	114	152	2%	6.49	-50%	128 ± 0
Learned Index	2nd stage size: 10,000	Binary	222	29	193	-15%	0.15	-99%	242 ± 150
		Quaternary	224	29	195	-14%	0.15	-99%	242 ± 150
	2nd stage size: 50,000	Binary	162	36	126	-38%	0.76	-94%	40 ± 27
		Quaternary	157	36	121	-40%	0.76	-94%	40 ± 27
	2nd stage size: 100,000	Binary	144	39	105	-45%	1.53	-88%	21 ± 14
		Quaternary	138	38	100	-47%	1.53	-88%	21 ± 14
	2nd stage size: 200,000	Binary	126	41	85	-52%	3.05	-76%	12 ± 7
		Quaternary	122	39	83	-53%	3.05	-76%	12 ± 7
Learned Index Complex	2nd stage size: 100,000	Binary	218	89	129	-16%	1.53	-88%	4218 ± 15917
		Quaternary	213	91	122	-18%	1.53	-88%	4218 ± 15917

Maps Dataset

Index of the longitude of ≈ 200M user-maintained features across the world. Relatively linear.

Type	Config	Search	Total (ns)	Model (ns)	Search (ns)	Speedup	Size (MB)	Size Savings	Model Err ± Err Var.
Btree	page size: 16	Binary	280	229	51	6%	104.91	700%	4 ± 0
	page size: 32	Binary	274	198	76	4%	52.45	300%	16 ± 0
	page size: 64	Binary	277	172	105	5%	26.23	100%	32 ± 0
	page size: 128	Binary	265	134	130	0%	13.11	0%	64 ± 0
	page size: 256	Binary	267	114	153	1%	6.56	-50%	128 ± 0
Learned Index	2nd stage size: 10,000	Binary	98	31	67	-63%	0.15	-99%	8 ± 45
		Quaternary	101	31	70	-62%	0.15	-99%	8 ± 45
	2nd stage size: 50,000	Binary	85	39	46	-68%	0.76	-94%	3 ± 36
		Quaternary	93	38	55	-65%	0.76	-94%	3 ± 36
	2nd stage size: 100,000	Binary	82	41	41	-69%	1.53	-88%	2 ± 36
		Quaternary	91	41	50	-66%	1.53	-88%	2 ± 36
	2nd stage size: 200,000	Binary	86	50	36	-68%	3.05	-77%	2 ± 36
		Quaternary	95	49	46	-64%	3.05	-77%	2 ± 36
Learned Index Complex	2nd stage size: 100,000	Binary	157	116	41	-41%	1.53	-88%	2 ± 30
		Quaternary	161	111	50	-39%	1.53	-88%	2 ± 30

Maps Dataset

Model is 3 × faster and up to an order-of-magnitude smaller.

Type	Config	Search	Total (ns)	Model (ns)	Search (ns)	Speedup	Size (MB)	Size Savings	Model Err ± Err Var.
Btree	page size: 16	Binary	280	229	51	6%	104.91	700%	4 ± 0
	page size: 32	Binary	274	198	76	4%	52.45	300%	16 ± 0
	page size: 64	Binary	277	172	105	5%	26.23	100%	32 ± 0
	page size: 128	Binary	265	134	130	0%	13.11	0%	64 ± 0
	page size: 256	Binary	267	114	153	1%	6.56	-50%	128 ± 0
Learned Index	2nd stage size: 10,000	Binary	98	31	67	-63%	0.15	-99%	8 ± 45
		Quaternary	101	31	70	-62%	0.15	-99%	8 ± 45
	2nd stage size: 50,000	Binary	85	39	46	-68%	0.76	-94%	3 ± 36
		Quaternary	93	38	55	-65%	0.76	-94%	3 ± 36
	2nd stage size: 100,000	Binary	82	41	41	-69%	1.53	-88%	2 ± 36
		Quaternary	91	41	50	-66%	1.53	-88%	2 ± 36
	2nd stage size: 200,000	Binary	86	50	36	-68%	3.05	-77%	2 ± 36
		Quaternary	95	49	46	-64%	3.05	-77%	2 ± 36
Learned Index Complex	2nd stage size: 100,000	Binary	157	116	41	-41%	1.53	-88%	2 ± 30
		Quaternary	161	111	50	-39%	1.53	-88%	2 ± 30

Maps Dataset

Quarternary search does not help.

Type	Config	Search	Total (ns)	Model (ns)	Search (ns)	Speedup	Size (MB)	Size Savings	Model Err ± Err Var.
Btree	page size: 16	Binary	280	229	51	6%	104.91	700%	4 ± 0
	page size: 32	Binary	274	198	76	4%	52.45	300%	16 ± 0
	page size: 64	Binary	277	172	105	5%	26.23	100%	32 ± 0
	page size: 128	Binary	265	134	130	0%	13.11	0%	64 ± 0
	page size: 256	Binary	267	114	153	1%	6.56	-50%	128 ± 0
Learned Index	2nd stage size: 10,000	Binary	98	31	67	-63%	0.15	-99%	8 ± 45
		Quaternary	101	31	70	-62%	0.15	-99%	8 ± 45
	2nd stage size: 50,000	Binary	85	39	46	-68%	0.76	-94%	3 ± 36
		Quaternary	93	38	55	-65%	0.76	-94%	3 ± 36
	2nd stage size: 100,000	Binary	82	41	41	-69%	1.53	-88%	2 ± 36
		Quaternary	91	41	50	-66%	1.53	-88%	2 ± 36
	2nd stage size: 200,000	Binary	86	50	36	-68%	3.05	-77%	2 ± 36
		Quaternary	95	49	46	-64%	3.05	-77%	2 ± 36
Learned Index Complex	2nd stage size: 100,000	Binary	157	116	41	-41%	1.53	-88%	2 ± 30
		Quaternary	161	111	50	-39%	1.53	-88%	2 ± 30

Lognormal Dataset

Synthetic dataset of 190M unique values to test how the index works on heavy-tail distributions. Highly non-linear, making the distribution more difficult to learn.

Type	Config	Search	Total (ns)	Model (ns)	Search (ns)	Speedup	Size (MB)	Size Savings	Model Err ± Err Var.
Btree	page size: 16	Binary	285	233	52	9%	99.66	700%	4 ± 0
	page size: 32	Binary	274	198	77	4%	49.83	300%	16 ± 0
	page size: 64	Binary	274	169	105	4%	24.92	100%	32 ± 0
	page size: 128	Binary	263	131	131	0%	12.46	0%	64 ± 0
	page size: 256	Binary	271	117	154	3%	6.23	-50%	128 ± 0
Learned Index	2nd stage size: 10,000	Binary	178	26	152	-32%	0.15	-99%	17060 ± 61072
		Quaternary	166	25	141	-37%	0.15	-99%	17060 ± 61072
	2nd stage size: 50,000	Binary	162	35	127	-38%	0.76	-94%	17013 ± 60972
		Quaternary	152	35	117	-42%	0.76	-94%	17013 ± 60972
	2nd stage size: 100,000	Binary	152	36	116	-42%	1.53	-88%	17005 ± 60959
		Quaternary	146	36	110	-45%	1.53	-88%	17005 ± 60959
	2nd stage size: 200,000	Binary	146	40	106	-44%	3.05	-76%	17001 ± 60954
		Quaternary	148	45	103	-44%	3.05	-76%	17001 ± 60954
Learned Index Complex	2nd stage size: 100,000	Binary	178	110	67	-32%	1.53	-88%	8 ± 33
		Quaternary	181	111	70	-31%	1.53	-88%	8 ± 33

Lognormal Dataset

The error is high, which influences the search time.

Type	Config	Search	Total (ns)	Model (ns)	Search (ns)	Speedup	Size (MB)	Size Savings	Model Err ± Err Var.
Btree	page size: 16	Binary	285	233	52	9%	99.66	700%	4 ± 0
	page size: 32	Binary	274	198	77	4%	49.83	300%	16 ± 0
	page size: 64	Binary	274	169	105	4%	24.92	100%	32 ± 0
	page size: 128	Binary	263	131	131	0%	12.46	0%	64 ± 0
	page size: 256	Binary	271	117	154	3%	6.23	-50%	128 ± 0
Learned Index	2nd stage size: 10,000	Binary	178	26	152	-32%	0.15	-99%	17060 ± 61072
		Quaternary	166	25	141	-37%	0.15	-99%	17060 ± 61072
	2nd stage size: 50,000	Binary	162	35	127	-38%	0.76	-94%	17013 ± 60972
		Quaternary	152	35	117	-42%	0.76	-94%	17013 ± 60972
	2nd stage size: 100,000	Binary	152	36	116	-42%	1.53	-88%	17005 ± 60959
		Quaternary	146	36	110	-45%	1.53	-88%	17005 ± 60959
	2nd stage size: 200,000	Binary	146	40	106	-44%	3.05	-76%	17001 ± 60954
		Quaternary	148	45	103	-44%	3.05	-76%	17001 ± 60954
Learned Index Complex	2nd stage size: 100,000	Binary	178	110	67	-32%	1.53	-88%	8 ± 33
		Quaternary	181	111	70	-31%	1.53	-88%	8 ± 33

Important Observations

- 3 × faster and being up to an order-of-magnitude smaller.
- Quarternary search only helps for some datasets.
- The model accuracy varies widely. Most noticeable for the synthetic dataset and the weblog data the error is much higher.
- Second stage size has a significant impact on the index size and lookup performance.
  - This is not surprising as the second stage determines how many models have to be stored. Worth noting is that our second stage uses 10,000 or more models.

Web Document Dataset

The web-document dataset consists of the 10M non-continuous document-ids of a large web index used as part of a real product at a large internet company.

	Config	Total (ns)	Model (ns)	Search (ns)	Speedup	Size (MB)	Size Savings	Std. Err ± Err Var.
Btree	page size: 32	1247	643	604	-3%	13.11	300%	8 ± 0
	page size: 64	1280	500	780	-1%	6.56	100%	16 ± 0
	page size: 128	1288	377	912	0	3.28	0	32 ± 0
	page size: 256	1398	330	1068	9%	1.64	-50%	64 ± 0
Learned Index	1 hidden layer	1605	503	1102	25%	1.22	-63%	104 ± 209
	2 hidden layers	1660	598	1062	29%	2.26	-31%	42 ± 75
Hybrid Index t=128	1 hidden layer	1397	472	925	8%	1.67	-49%	46 ± 29
	2 hidden layers	1620	591	1030	26%	2.33	-29%	38 ± 28
Hybrid Index t=64	1 hidden layer	1220	440	780	-5%	2.50	-24%	41 ± 20
	2 hidden layers	1447	556	891	12%	2.79	-15%	34 ± 20
Learned QS	1 hidden layer	1155	496	658	-10%	1.22	-63%	104 ± 209

Web Document Dataset

Speedups for learned indexes is not as prominent, so hybrid indexes, which replace bad performing models with B-trees actually help to improve performance.

	Config	Total (ns)	Model (ns)	Search (ns)	Speedup	Size (MB)	Size Savings	Std. Err ± Err Var.
Btree	page size: 32	1247	643	604	-3%	13.11	300%	8 ± 0
	page size: 64	1280	500	780	-1%	6.56	100%	16 ± 0
	page size: 128	1288	377	912	0	3.28	0	32 ± 0
	page size: 256	1398	330	1068	9%	1.64	-50%	64 ± 0
Learned Index	1 hidden layer	1605	503	1102	25%	1.22	-63%	104 ± 209
	2 hidden layers	1660	598	1062	29%	2.26	-31%	42 ± 75
Hybrid Index t=128	1 hidden layer	1397	472	925	8%	1.67	-49%	46 ± 29
	2 hidden layers	1620	591	1030	26%	2.33	-29%	38 ± 28
Hybrid Index t=64	1 hidden layer	1220	440	780	-5%	2.50	-24%	41 ± 20
	2 hidden layers	1447	556	891	12%	2.79	-15%	34 ± 20
Learned QS	1 hidden layer	1155	496	658	-10%	1.22	-63%	104 ± 209

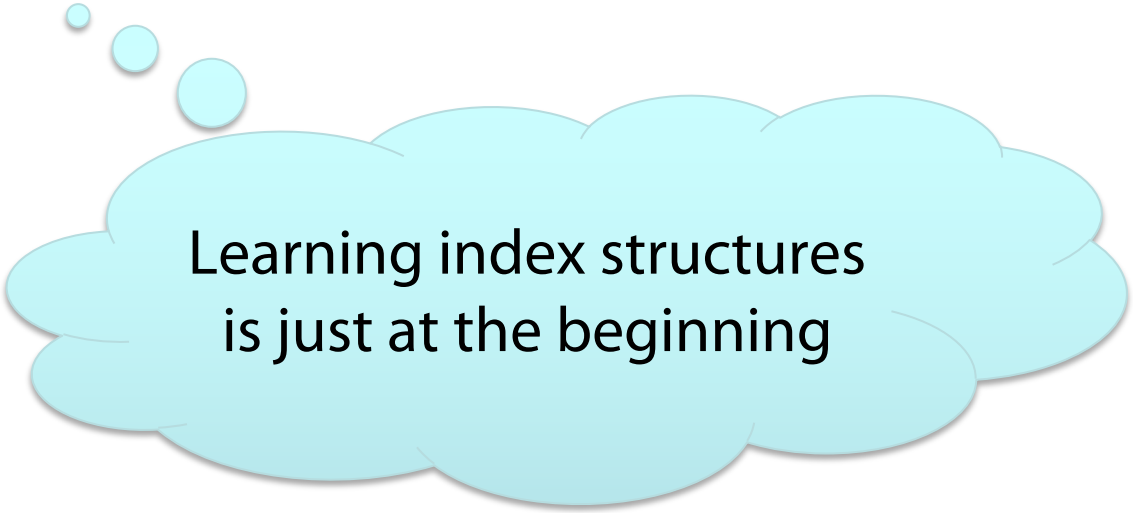
Web Document Dataset

Because cost of searching is higher, the different search strategies make a bigger difference. The reason why biased search and quaternary search performs better is that they can take the standard error into account.

	Binary Search		Biased Search		Quaternary Search	
	Total	Search	Total	Search	Total	Search
NN 1 hidden layer	1605	1102	1301	801	1155	658
NN 2 hidden layer	1660	1062	1338	596	1216	618

Conclusion

- Well known “static” structures perform surprisingly well
- It is quite difficult to be faster with learning…
- … and hierarchies of index function are pretty complex
- But there is light at the end of the tunnel



Future Work

- Multi-Dimensional Indexes:
  - Extend learned indexes to multi-dimensional index structures.
  - Network models, especially ConvNets, are extremely good at capturing complex high-dimensional relationships
- Learned Algorithms: A model can also speed-up sorting and joins, not just indexes
- GPU/TPUs: GPU/TPUs will make the idea of learned indexes even more viable