Function Interpolation for Learned Index Structures

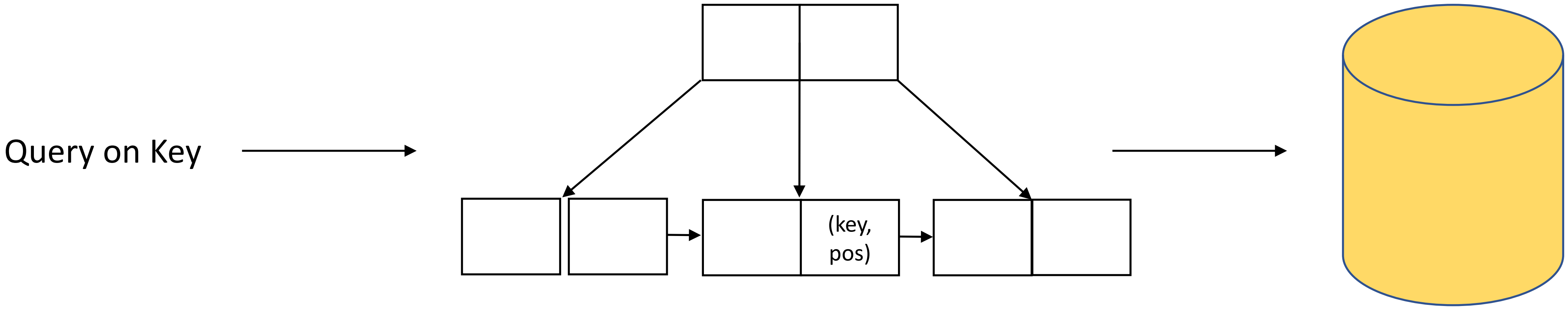
Naufal Fikri Setiawan,
Benjamin I.P. Rubinstein,
Renata Borovica-Gajic

University of Melbourne

Acknowledgement: CORE Student Travel Scholarship

Querying data with an index

- Indexes are external structures used to make lookups faster.
- B-Tree indexes are created on databases where the keys have an ordering.



$$key \rightarrow position$$

On Learned Indexes

- An experiment by Kraska, et al. [*] to replace range index structure (i.e. B-Tree) with neural networks to "predict" position of an entry in a database.
  - Reduce $O(\log n)$ traversal time to $O(1)$ evaluation time.

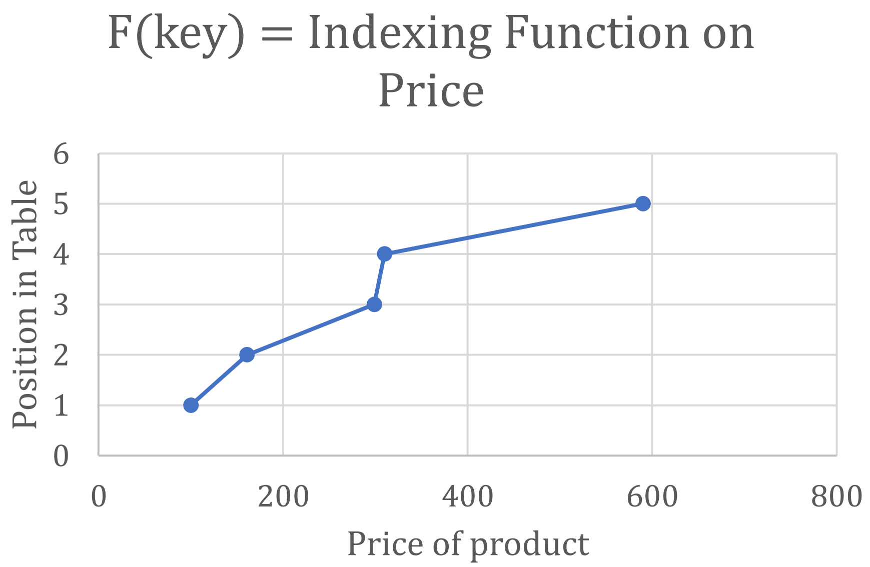
- Indexing is a problem on learning how data is distributed.

- Aim: To explore the feasibility of an alternative statistical tool: **polynomial interpolation** in indexing.

Kraska, Tim, et al. "The case for learned index structures."
Proceedings of the 2018 International Conference on Management of Data. 2018.

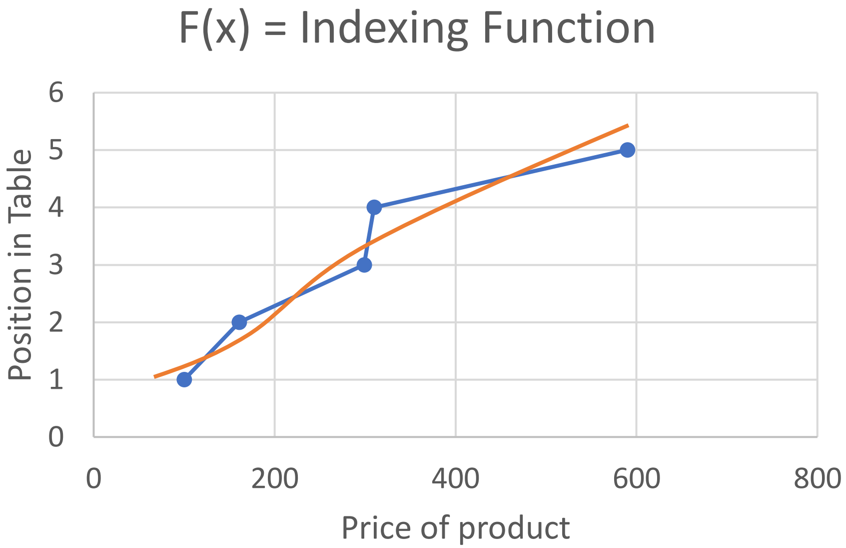
Mathematical View on Indexing

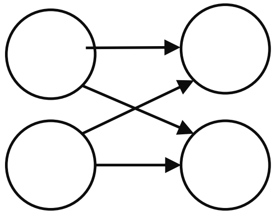
Product	Price (Key)
Product A	100
Product X	161
Product L	299
Product D	310
Product G	590



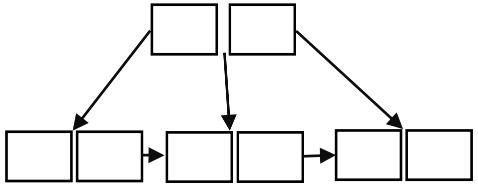
An index is a **function** $\boldsymbol{f}: \boldsymbol{U} \mapsto \boldsymbol{N}$ that **takes** a **query** and **return the position**.

So… we can build a model to predict them!





Neural Networks

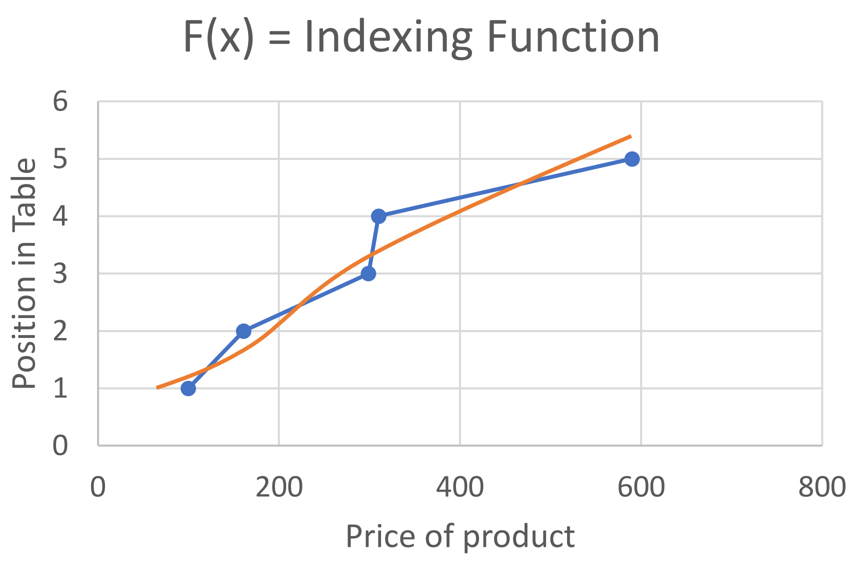


Trees!

$$f(x) \approx \sum a_i x^i$$

Polynomial Models

Polynomial Models - Preface



For a chosen degree n

$$position \approx a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Use two different interpolation methods to obtain a_i:

- Bernstein Polynomial Interpolation
- Chebyshev Polynomial Interpolation

Meet our Models

Bernstein Interpolation Method

$$\sum_{i=0}^{N} \alpha_i \binom{N}{i} x^i (1-x)^{N-i}$$

Model parameters $\langle \alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_N \rangle$

Where

$$\alpha_i = f\left(\frac{i}{N}\right)$$

And f is the function we want to approximate, scaled to $[0,1]$.

In memory: only need to store the coefficients

$$\alpha_i \cdot \binom{N}{i}$$

Meet our Models

Chebyshev Interpolation Method

$$\sum_{i=0}^{N} \alpha_i T_i(x)$$

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x)$$

Coefficients given by Discrete Chebyshev Transform

$$\alpha_i = \frac{p_i}{N} \sum_{k=0}^{N-1} \left[f\left(-\cos\left(\frac{\pi}{N}\left(k + \frac{1}{2}\right)\right)\right) \cdot \cos\left(\frac{i\pi}{N}\left(N + k + \frac{1}{2}\right)\right)\right]$$

$$p_0 = 1, p_k = 2 \text{ (if } k > 0)$$

Domain is $[-1, 1]$

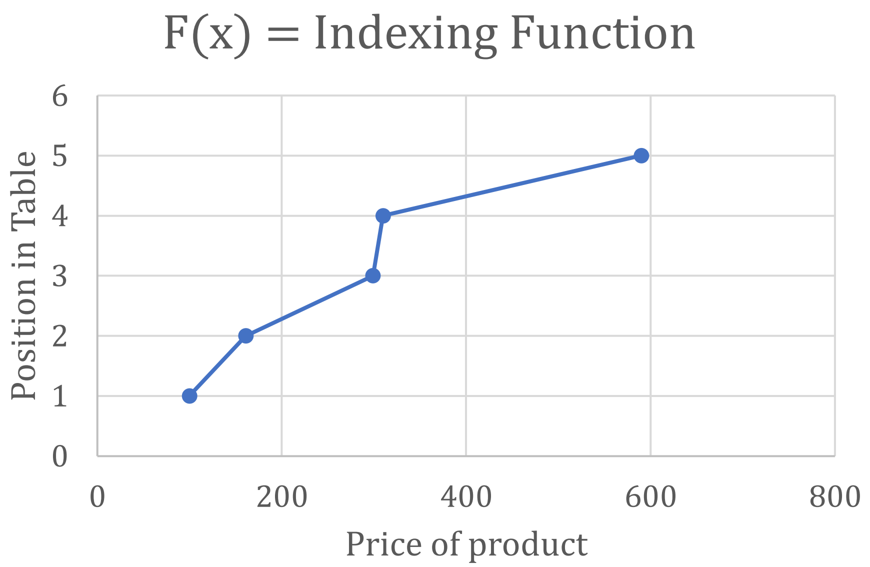
Indexing as CDF Approximation

If we:

- Pre-sort the values in the table, we get the following equation:

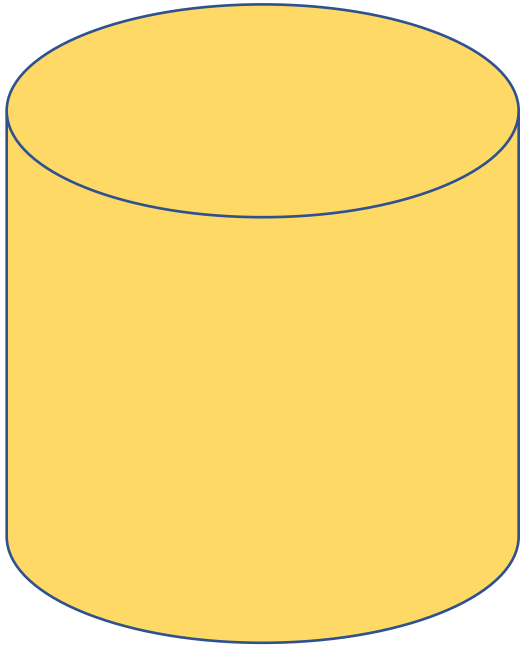
$$F(key) = P(x \leq key) \times N$$

Our polynomial models need to simply predict the CDF, with key rescaled to the interpolation domain.



A Query System

Data is not necessarily sorted in DB



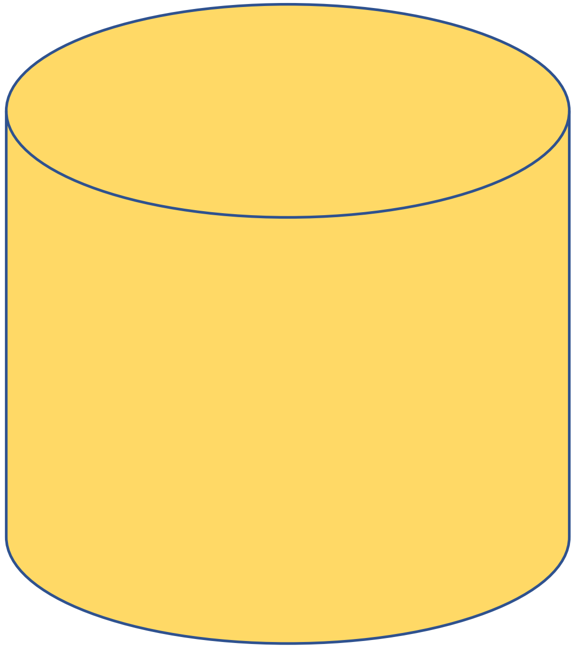
Query Model Step 1: Creation of Data Array

A Query System

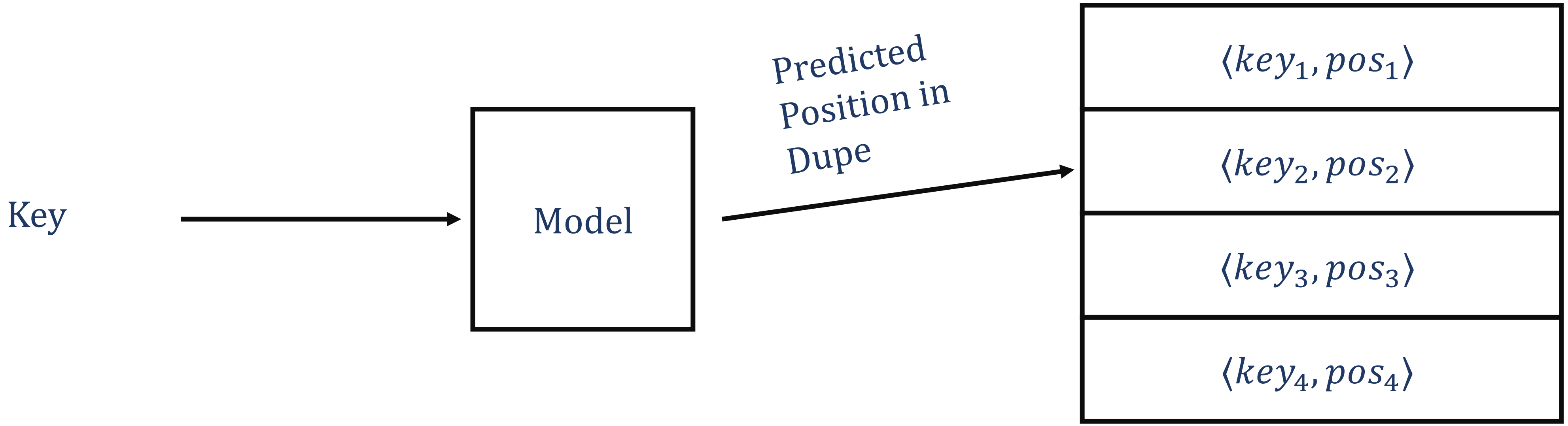
Sorted Data Dupe (A)

$\langle key_1, pos_1 \rangle$
$\langle key_2, pos_2 \rangle$
$\langle key_3, pos_3 \rangle$
$\langle key_4, pos_4 \rangle$

Data is not necessarily sorted in DB

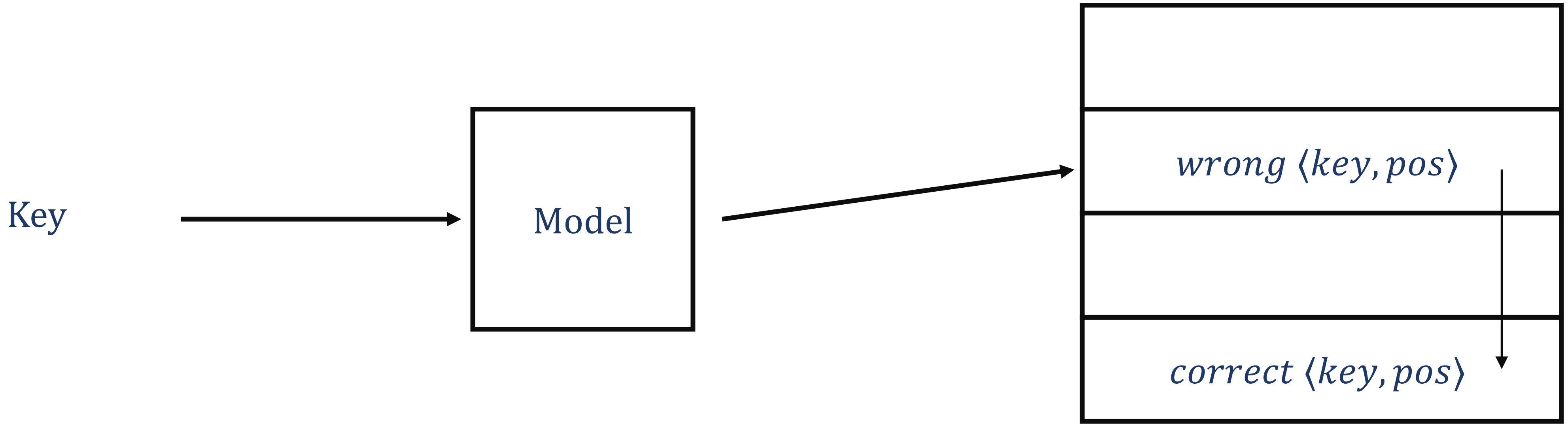


A Query System



Query Model Step 1: Predict position

A Query System



Query Model Step 2: Error correction

Experiment Setup

- Created random datasets with multiple distributions as keys:
  - Normal, Log-Normal, and Uniform.
- Each distribution:
  - 500k, 1M, 1.5M, 2M rows.
- We test the performance of each index
  - NN, B-Tree, polynomial
- Hardware setup:
  - Core i7, 16GB of RAM.
  - Python 3.7 on GCC running on Linux.
  - PyTorch for Neural Network purposes.
  - No form of GPU use.

Benchmark Neural Network

- Neural Network:
  - 1hr benchmark training time.
  - 2 hidden layers x 32 neurons.
  - RelU activation.

Index Creation / “Training” Time

Model Type	Creation Time
B-Tree	34.57 seconds
Bernstein(25) Polynomial	3.366 seconds
Chebyshev(25) Polynomial	3.809 seconds
Neural Network Model	1hr (benchmark)

- Polynomial models are created faster than B-Trees.
- Polynomial models do not require any hyperparameter tuning.
- NNs, however, can be incrementally trained.

Factor of 10 creation time reduction over B-Trees

Model Prediction Time

Model **Type**	**Prediction Time (nanoseconds)**		
	Normal	LogNormal	Uniform
B-Tree	24.4	40.1	41.5
Bernstein(25) Polynomial	277	336	166
Chebyshev(25) Polynomial	25.9	31.7	16.4
Neural Network Model	406	806	148

Model prediction time for 2 million rows.

Polynomial models are able to predict faster than NNs.

Model Accuracy

Model Type	Root Mean Squared Positional Error		
	Normal	LogNormal	Uniform
B-Tree	N/A		
Bernstein(25) Polynomial	9973.67	39566.59	62.58
Chebyshev(25) Polynomial	57.14	474.91	26.39
Neural Network Model	105.84	711.12	22.67

Average error for 2 million rows.

Chebyshev Models are ~50% more accurate

Total Query Speed

Model Type	Average Query Times (nanoseconds)		
	Normal	LogNormal	Uniform
B-Tree	31.5	46.0	56.3
Chebyshev(25) Polynomial	62.1	751	40.2
Bernstein(25) Polynomial	8080	11800	192
Neural Network Model	402	1100	516

Chebyshev Models are 30% - 90% faster at querying.

Memory Usage

Model Type	Size of Database (in Entries)			
	500k Entries	1M Entries	1.5M Entries	2M Entries
B-Tree	33.034 MB	66.126 MB	99.123 MB	132.163 MB
Neural Network	210.73 kB	210.73 kB	210.73 kB	210.73 kB
Bernstein(25) Polynomial	1.8kB	1.8kB	1.8kB	1.8kB
Chebyshev(25) Polynomial	1.8kB	1.8kB	1.8kB	1.8kB

99.4% Reduction from B-Trees

99.3% reduction from Neural Network Model

Main Key Insight

- “Indexing” is better interpreted as less of a learning problem and more of a fitting problem. Where **overfitting** is advantageous.

- Learning: **separate training and test data.**
- Fitting: **same training and test data.**

Conclusion

- We advocate for the use of function interpolation as a ‘learned index’ due to the following benefits:
  - No hyperparameter tuning.
  - Fast creation time on a CPU-only environment.
  - Provides a higher compression rate vs. Neural Networks and definitely vs. B-Trees.