ESIndex: Exploring Learned Index Structures for Strings

Alex Kumar, Marcus Ortiz

Description

Team BYODB proposes implementing learned indices. For simplification, we propose implementing learned indices separate from the LMDB implementation used in the course database management system. In our implementation independent of the LMDB, key/value pairs will be stored within a Pandas Series object, with the key serving as the index of the string in the table and the value being the record. Our new learned index functionality aims to improve point query performance in read-only tables by reducing comparisons and I/O operations when searching for data. This can be achieved by accurately predicting the index of the queried value and using an efficient last-mile search algorithm to find the record. Our approach also aims to reduce memory overhead by replacing larger stored index structures with smaller predictive models.

Motivation

Indexing is a critical feature for efficient querying. Today, classical auxiliary structures such as B-trees and hash maps enable indexing. However, these implementations are bulky and inefficient due to extensive string comparisons in B-tree navigation and inefficient due to poorly mapped data from hash functions. More recently, learned index models have been proposed as lightweight, efficient alternatives to traditional index data structures.

Our work aims to build upon the recent advancements in learned index structures by implementing a novel approach tailored for string keys. We explore different machine learning models, such as linear and non-linear models, to predict the index of a given string key. Additionally, we will investigate efficient last-mile search algorithms to further optimize the lookup performance. By combining accurate predictive models and efficient search techniques, we aim to outperform traditional indexing methods in terms of query latency and memory footprint.

Related Work

The Case for Learned Index Structures

Our work builds upon the recent advancements in learned index structures, as discussed in the paper "The Case for Learned Index Structures" [1]. That paper makes a compelling case for using machine learning models as an alternative to traditional index data structures like B-trees and hash maps. The key insight is that indexes can be viewed as models that predict the position of a given key within a dataset. Our work aims to leverage this concept of learned indexes, but with a specific focus on improving indexing for string keys.

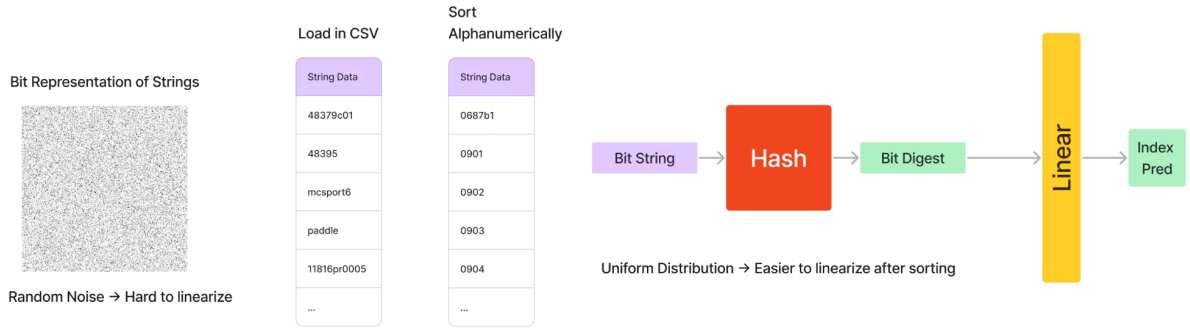
SIndex: A Scalable Learned Index for String Keys

The SIndex paper [2] presents a scalable learned index structure specifically designed for string keys. The key innovations of SIndex include a hierarchical model architecture that can handle large string key domains, as well as efficient search algorithms to quickly locate the desired record after the initial model

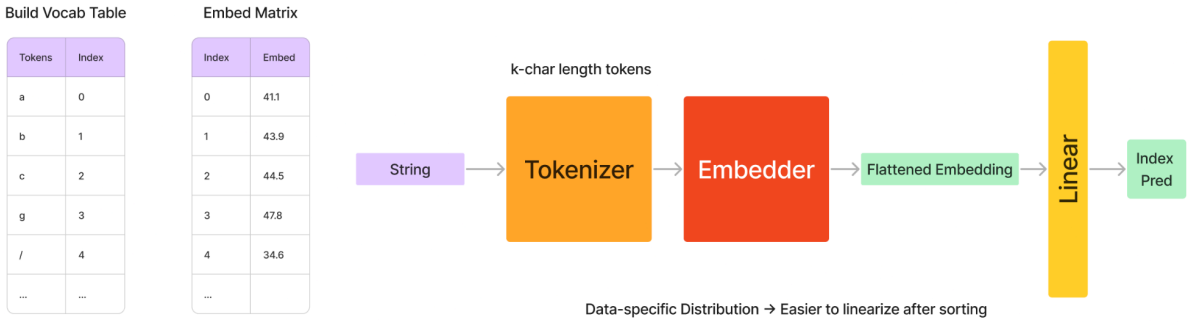
prediction. SIndex was shown to outperform traditional index structures like B-trees and hash maps in terms of both query latency and memory usage. By drawing on the lessons learned from SIndex, we aim to further advance the state-of-the-art in learned index structures for string-based data.

Our Work

This paper proposes two innovative solutions for indexing strings in read-heavy databases, drawing inspiration from the SIndex approach. The first solution relies on well-known randomized hash functions to uniformly redistribute the string data, such that a learned linear model can more accurately predict the indexes. By hashing the string keys, we can transform the original non-uniform data distribution into a more uniform one. We then sort the table based on the digests associated with each string key. This enables a simple linear model to more effectively learn the mapping between keys and their positions. This approach aims to act as a baseline for our more advanced embedding based solution.



The second solution leverages custom learned token and positional embeddings. By training an embedding matrix to capture the positional relationships of tokens within the string keys, we hypothesize that a learned linear model will be able to more accurately predict the indexes compared to a model operating directly on the raw string data or the sorted hashed string data.



In both cases, our proposed solutions build upon the key insights and techniques from the SIndex paper, aiming to deliver efficient and scalable learned index structures tailored for string-based data. By combining accurate predictive models with effective last-mile search algorithms, we seek to outperform traditional indexing methods in terms of query latency and memory overhead.

Goals

Basic Goals:

1. Design and code a learned index
    1. Perfectly realized (talked with Professor Yang to not build the index on top of the DDB system)
2. Train a machine learning model on a dataset of key-value pairs to capture the underlying patterns and distribution reasonably correctly
    1. Perfectly realized

Bonus Goals:

1. Evaluate the accuracy, speed, and storage of the learned index for *integers and floats*
    1. Realized but imperfect
2. Include *strings* as a supported indexed data type
    1. Perfectly realized
3. Perform model compression via quantization and weight pruning to further reduce memory requirement and inference time
    1. Not attempted

Evaluation

We conducted experiments using PyTorch Lightning with the following configurations:

- Batch size: 128
- Token length: 1
- Embedding size: 1
- Number of minutes: 1
- Maximum epochs: 100

We evaluated our learned index structures on three distinct datasets: IMDB, Lego, and Random. Each dataset was processed using two different mapping techniques (Hash and Embed) and indexed using a linear indexing function.

Dataset	Dataset Length	Mapper	Indexer	Percent Narrowed	Number of Parameters
IMDB	1000	Hash	Linear	0.3778	2
IMDB	1000	Hash	2L Linear	0.3775	4
IMDB	1000	Embed	Linear	0.2907	400
IMDB	1000	Embed	2L Linear	0.2916	98700
Lego	580251	Hash	Linear	0.3804	2
Lego	580251	Hash	2L Linear	0.3628	4

Lego	580251	Embed	Linear	0.2940	56
Lego	580251	Embed	2L Linear	0.2889	296
Random	100	Hash	Linear	0.5248	2
Random	100	Hash	2L Linear	0.5545	4
Random	100	Embed	Linear	0.4469	71
Random	100	Embed	2L Linear	0.3190	143
Random	1000000	Hash	Linear	0.3821	2
Random	1000000	Hash	2L Linear	0.5771	4
Random	1000000	Embed	Linear	0.5297	71
Random	1000000	Embed	2L Linear	0.2887	143

For the IMDB dataset with 1000 records, the learned index structure achieved a percent narrowed value of 0.3778 with the Hash mapping technique and 0.2907 with the Embed mapping technique. The lower percent narrowed value with Embed mapping suggests that the embedding-based approach may be more effective in capturing the underlying patterns in the dataset, leading to improved query performance. This is unsurprising when accounting for the larger number of learned parameters

On the larger Lego dataset comprising 580,251 records, the percent narrowed values were 0.3804 with Hash mapping and 0.2940 with Embed mapping. Despite the larger dataset size, the relative performance between Hash and Embed mapping remained consistent. Again, with more parameters, it appears we are better able to learn the underlying patterns.

We conducted experiments on the Random dataset with varying record lengths: 100 and 1,000,000 records. The learned index structures achieved percent narrowed values of 0.5248 and 0.3821 respectively, with Hash mapping, and 0.4469 and 0.5297, respectively, with Embed mapping.

Interestingly, we observed that the Embed mapping technique outperformed Hash mapping for smaller dataset sizes (100 and 1000 records), indicating its effectiveness in capturing patterns even in datasets with limited samples, but not for the large dataset size. We believe this is due to the constraints we placed on training. The model was not able to complete one full pass through the data with the embed method. Therefore, some of the embeddings remained in their randomly initialized state. However, we also notice that with an additional linear layer, the more complex indexing model is able to generally improve cutdown performance, especially on larger datasets. Our team was unable to find a sufficient explanation for the massive improvement on the Random-1M dataset using embedding, as well as the significant decrease in performance on the same large dataset using hashing.

Future Work

Despite the challenges with learning string indices, we are confident with further discussion and consideration, implementations can become practical and efficient. In particular we found the following topics interesting for future work,

- **Computational Loss Function:** The uniqueness of this problem with the Creating loss functions based on the amount of comparisons and computation needed for chosen last mile search rather than index prediction accuracy which is not wholly representative of what we would like to minimize
- **Catered Hash Functions:** Considering the strong performance for hashing and resorting, further consideration should be taken on the effects of different hash functions and what would cater best to creating a linear distribution of digests.
- **Simple Histograms:** Despite The Case for Learned Indices [cite] discouraging future research on Histograms serving as full replacements, we believe simple Histograms as secondary auxiliary structures are beneficial. They can highlight important distributions for inherent features in strings, such as prefixes and suffixes. These statistics could be dynamically integrated into loss functions or used to creatively construct the architecture, especially for Hierarchical implementations.

References

[1] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18). Association for Computing Machinery, New York, NY, USA, 489–504. https://doi.org/10.1145/3183713.3196909

[2] Wang, Youyun & Tang, Chuzhe & Wang, Zhaoguo & Chen, Haibo. (2020). SIndex: a scalable learned index for string keys. 17-24. 10.1145/3409963.3410496.

[3] Zhaoyan Sun, Xuanhe Zhou, and Guoliang Li. 2023. Learned Index: A Comprehensive Experimental Evaluation. Proc. VLDB Endow. 16, 8 (April 2023), 1992–2004. https://doi.org/10.14778/3594512.3594528

[4] Spector, B., Kipf, A., Vaidya, K., Wang, C., Minhas, U.F., & Kraska, T. (2021). Bounding the Last Mile: Efficient Learned String Indexing. ArXiv, abs/2111.14905.