Project Proposal: Learned Index Structure Implementation

Anders von Mirbach, Johnson Giang, Navkirat Puri, Renee Saulnier,
Sergio Diaz, Shiana Khehra

Computer Science Program University of Lethbridge

CPSC4660 – Database Management Systems

Prepared for: Dr. Samiha Brahimi

October 3, 2025

Table of Contents

Introduction

The paper *The Case for Learned Index Structures* makes the argument that indexes in databases can be thought of as models, and that machine learning models can sometimes do the same job faster and with less memory. For example, instead of using a B-Tree to map keys to positions in a sorted array, a model can learn the distribution of the keys and predict where to look. This turns the indexing problem into more of a prediction problem.

For this project, we want to take the core ideas from the paper and put together a working prototype. The focus will be on testing recursive model indexes (RMIs) and comparing them against normal B-Trees on different datasets. If we have enough time, we'll also try extending the same idea to hash indexes and Bloom filters. The end goal is not to build a production-ready system but to show how learned indexes actually perform, where they win, and where they don't.

Motivation and Problem Statement

Traditional indexes like B-Trees are efficient but can be memory-heavy and less adaptable to skewed data distributions. Learned indexes offer a new approach by treating indexing as a prediction problem, using models to estimate a key's position in a sorted array. This project aims to evaluate the effectiveness of Recursive Model Indexes (RMIs), a two-stage learned index structure, compared to standard B-Trees. We will benchmark both on datasets of varying sizes and distributions, measuring lookup time, memory usage, and accuracy. The goal is to understand where learned indexes outperform traditional methods and where they may fall short.

Tools & Implementation Plan

The main work will be done in Python, but we won't rely heavily on machine learning libraries. Instead, most of the code will be written by hand so we understand how the models and indexes work. For example, we'll implement our own linear regression, B-Tree, and recursive model index (RMI).

We will use standard Python modules for basics like random numbers, timing, and CSV handling. Libraries such as NumPy or Matplotlib may be used only for convenience with arrays and plotting results, but the learning logic itself will be coded directly. If time allows, we might also try porting the models to C++ for faster benchmarks.

- Definitely: Python (NumPy, pandas, PyTorch, Matplotlib)
- Optional: C++ for speed and benchmarks if time permits

Machine Learning Approach

For this project we'll treat indexing as a prediction problem. Instead of using a B-Tree to find where a key is, we'll train small models that guess the position of a key in a sorted list. Our starting point will be simple linear regression, which we'll code ourselves.

To improve accuracy, we'll also build a two-stage Recursive Model Index (RMI): a top model makes a rough guess, and smaller models refine it. This should keep lookups fast while using less space. If time allows, we may test a small neural net, but the main focus is on simple regression since it's enough to show how learned indexes work.

Testing & Benchmarking

We will evaluate learned indexes against traditional B-Trees using datasets of increasing size (1,000; 10,000; 100,000; and 1,000,000 keys). Each dataset will be stored as a sorted array to allow range queries. We want to make sure our tests are fair and easy to repeat. To do this, all benchmarks will run on the same computer. Before each test, we will clear or warm up the system so that things like caching don't give one method an unfair advantage. We will also run each test several times and average the results, to ensure random spikes don't throw things off. We also could run the program on a virtual machine for more controlled environment.

Metrics:

- Build Time
- Lookup time
- Memory Footprint
- Accuracy

How we will test:

- Varying on data sets in increasing size
- Same machine
- Minimum of five tests with an overall average score.

Project Deadlines

October 12	Review the paper and related work. Generate dataset. Set up coding environment. Implement baseline B-Tree for comparison.
October 19	Build first learned index with linear regression. Add local search correction. Run initial tests vs. B-Tree.
October 26	Implement Recursive Model Index (two-stage). Try linear + shallow NN at the top stage. Collect early benchmarks.
November 2	Add hybrid approach (B-Trees for hard cases). Stress test on irregular data. Document trade-offs.
November 9	(Optional) Extend to learned hash indexes. If not, refine RMI + hybrid experiments.

November 16	Wrap up experiments. Write comparisons (learned vs. B-Tree vs. hash) Prepare graphs and tables.
November 23	Buffer time.
November 28	Finalize report and code. Proofread and polish. Submit by Nov 28.

References

Kraska, T., Beutel, A., Chi, E. H., Dean, J., & Polyzotis, N. (2018). *The case for learned index structures*. Proceedings of the 2018 International Conference on Management of Data (pp. 489–504). Association for Computing Machinery. https://doi.org/10.1145/3183713.3196909