Benchmark Proposal: Verifying Learned Indexes of Databases

1 Background: learned index

Index structures are widely used in systems which enables efficient data accesses. For example, B-Tree is a type of index in databases which allows a database to quickly pinpoint data positions in an underlying storage. As an alternative, *learned index structure* uses neural networks (NNs) to replace B-Trees for better performance.

One type of learned index, named *Recursive Model Index* [1] (RMI), is depicted in Figure 1. RMI has multiple stages and each stage has one or multiple models (NNs). During a key lookup, the model in upper stages (starting from stage 1) picks a model in the next stage to run, and a final stage model predicts the data position for the key. As a best practice [1], people use two-staged RMIs.

One challenge for RMIs is to ensure that models *always* produce data positions within certain error-bounds, so that RMIs can always find existing data (a required property for any index structures). Original RMIs achieve this by evaluating all existing keys on the trained NN models, and replace those exceeding the error-bounds with traditional B-Trees. But, this approach does not provide guarantees for *non-existing* keys—the predicted data positions can be arbitrary. This affects range queries whose upper/lower bound of the range might be non-existing keys (see detailed discussion in original RMI paper [1, §3.4]).

It will be great if we can somehow know whether NN models have bounded errors for *all* keys, non-existing keys included. NN-verification can help. Given a specification requiring that predicted positions are within an error-bound, NN-verification can comprehensively check whether models hold this property for all keys.

2 Benchmark: verifying RMIs

In this section, we propose a benchmark: verifying RMIs to check whether all predicted positions—including predictions for non-existing keys—are within a predefined error bound (denoted as ε).

We use the Integer Datasets [1, §3.7.1], in which keys and positions are both integers on three different data distributions: *normal*, *lognormal*, and *piecewise linear*. The stored data are sorted by their keys, a common scenario in databases for supporting range queries (see Figure 2 as an example).

For models, we follow the best practice RMI design

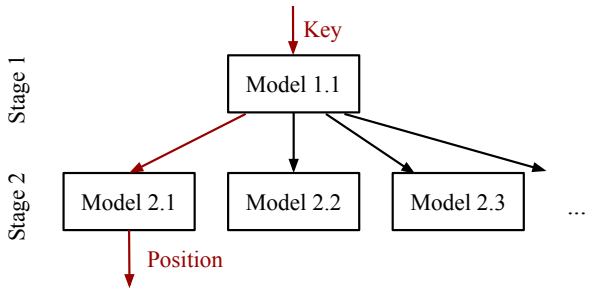

Figure 1: A two-stage Recursive Model Index (RMI). In this example, this RMI takes a "Key" as an input, chooses "Model 1.1" and "Model 2.1" for prediction, and finally produces the "Position" which is supposed to be where the data (indexed by "Key") is located in the database.

which has two stages: stage 1 has one large NN model with multiple fully connected layers, with ReLu as activation functions; stage 2 has many small NN (or linear) models. The RMI's input is an integer (the key) and the prediction is also an integer (the position).

The required property is as follows. For all keys, the RMI's predicted position must be at most ε slots away from its true position (for existing keys) or the position that the data should be if the key were inserted (for non-existing keys).

RMI specifications. To design a specification for the required property, we partition the input space (regarding existing keys) into *segments*. The specifications dictates that all predictions for the keys within one segment must fall into the corresponding output range, which bounded by the parameter ε. For example, an input segment $[x_i, x_j]$ has a corresponding prediction range $[y_i, y_j]$; that is, key x_i and x_j are located at position y_i and y_j, respectively. The specification may have a constraint $\langle [x_i, x_j], [y_i - \varepsilon, y_j + \varepsilon] \rangle$.

In particular, specifications have two parameters, the size of segment and the ε. The smaller the segment, the stricter the specification, similarly for ε. Database users can tune these parameters for their own needs. In our proposed benchmark, we provide a set of specifications that combine different segment sizes and ε to simulate different demands.

Take specifications in Figure 2 as an example. The segment we choose is $[2, 97]$ (in NN input space) which corresponds to $[0, 10]$ (in NN output space). In specifications,

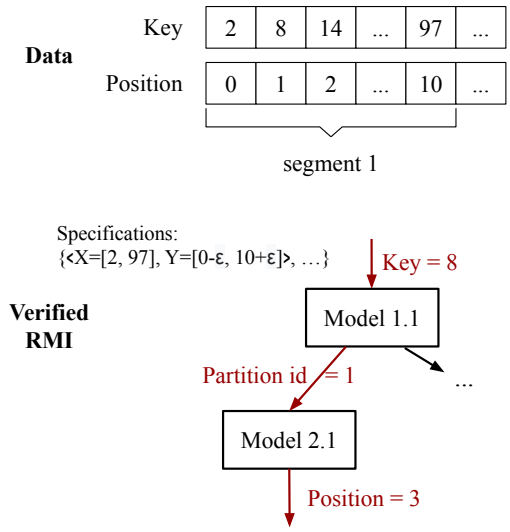

Figure 2: An example of an RMI with specifications. On the top is the (sorted) data that the RMI indexes. On the bottom is a two-staged RMI with specifictions. In this example, the RMI predicts that the data indexed by "Key=8" should locate at "Position=3", and the true position is "1" (see "Data"). This prediction satisfies the specification because "Key=8" is in the segment $[2,97]$ and its prediction "Position=3" is within the range $[0-\varepsilon, 10+\varepsilon]$.

$\langle \mathcal{X} = [2,97], \mathcal{Y} = [0-\varepsilon, 10+\varepsilon] \rangle$ reads as: if a key is in the range of $[0,97]$, then the predicted position must be within $[0-\varepsilon, 10+\varepsilon]$, where "0" and "10" are the true positions of "Key=2" and "Key=97", respectively; ε is the error bound. As we can see in Figure 2, the range $[2,97]$ captures the existing keys (e.g., "Key=8") and all non-existing keys that would have been placed immediately before or after the existing ones. And, the predicted positions of all these keys should be at most $len(segment)+\varepsilon$ slots away from their true positions.

References

[1] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pages 489–504, 2018.