I/O Efficient Machine Learning

faculty/staff/researcher names

Johns Hopkins University | Whiting School of Engineering | Baltimore, MD

CONTRIBUTION 1: Inference Latency Reduction in Tree Ensemble Models

MOTIVATION

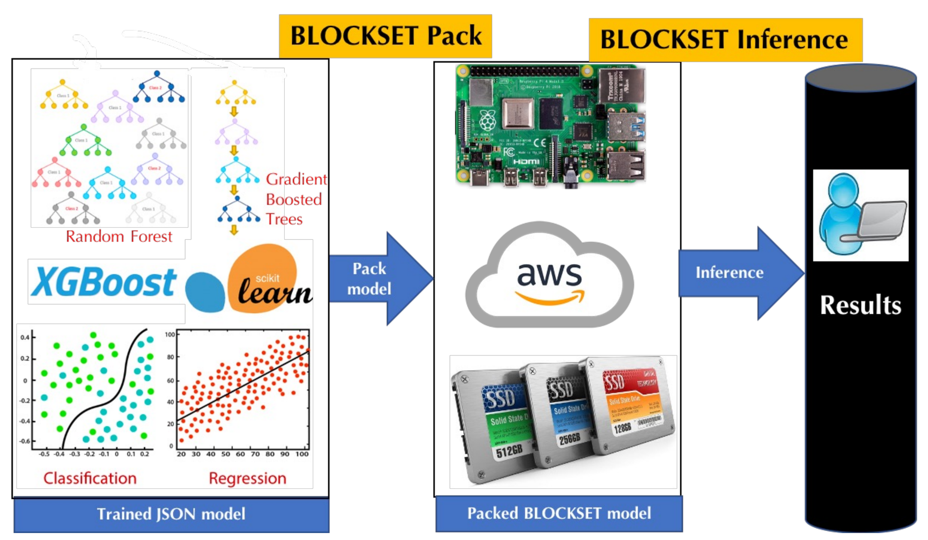
Tree ensembles, random forests and gradient boosted trees, are useful in resource-limited machine learning deployment. Traversing tree data structures is not cache friendly, which results in high latency during inference Moreover, existing tree ensemble frameworks are designed for batch throughput and require the model to fit in RAM.. We introduce two systems BLOCKSET and T-REX to alleviate these issues. BLOCKSET serializes gradient-boosted trees and random forests to optimize inference latency **when models are not loaded into memory .** We pack tree nodes and paths together using node access probabilities in a block-aligned fashion to reduce I/Os. T-REX performs efficient inference more generally – even when the models are stored in memory by mapping tree traversals to hyperrectangle enclosures due to the observation that decision trees partition d-dimensional spaces to convex polytopes.

Project 1: BLOCKSET (Block Aligned Serialized Trees)

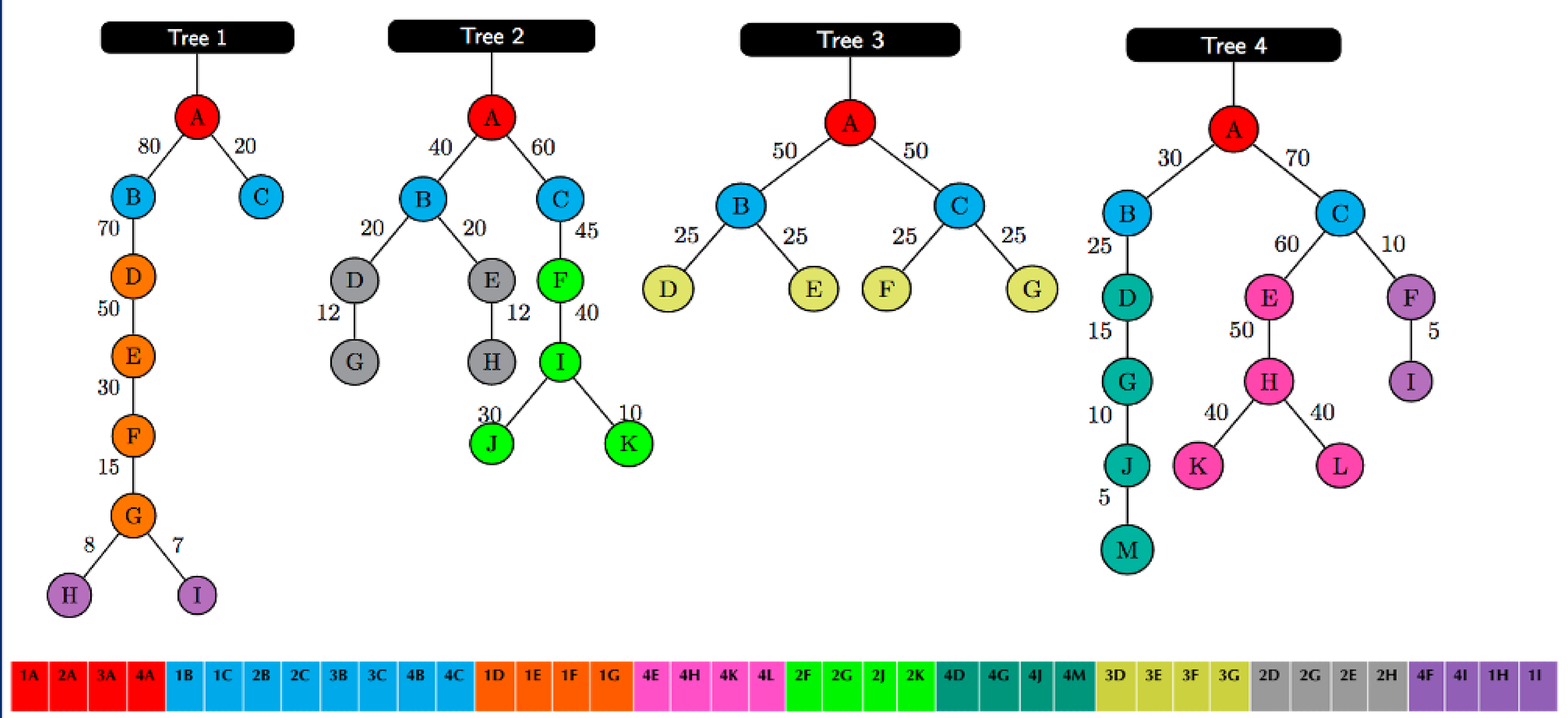
OVERVIEW

- BLOCKSET introduces the concept of selective access to load necessary parts of the model **on demand** during inference for models that don't fit in memory.
- We **block align** the serialization format in order to minimize the number of I/Os.
- BLOCKSET supports three use cases – inference on embedded devices, inference on microservices such as lambda and inference when the model is stored on SSDs.

SYSTEM DESIGN

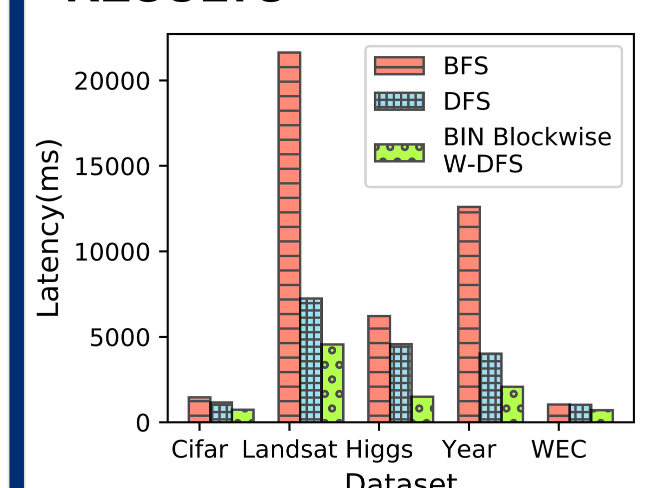


BLOCKSET reads a trained tree ensemble, applies node packing optimizations and finally performs inference on the packed model



Block-aligned layout of the interior nodes of a classification forest with a block size of 4 nodes. The top nodes (1A-4C) are interleaved . The highest cardinality paths are grouped into blocks. Colors indicate block boundaries for residuals.

RESULTS



- This graph shows average inference latency compared to existing layouts.
- BLOCKSET reduces inference latency by **2-5x times** over existing methods and a 100x reduction.
- It reduces memory requirements by 100x because the model is stored on the disk

FOR MORE INFORMATION VISIT

TODO: QR CODE WITH link

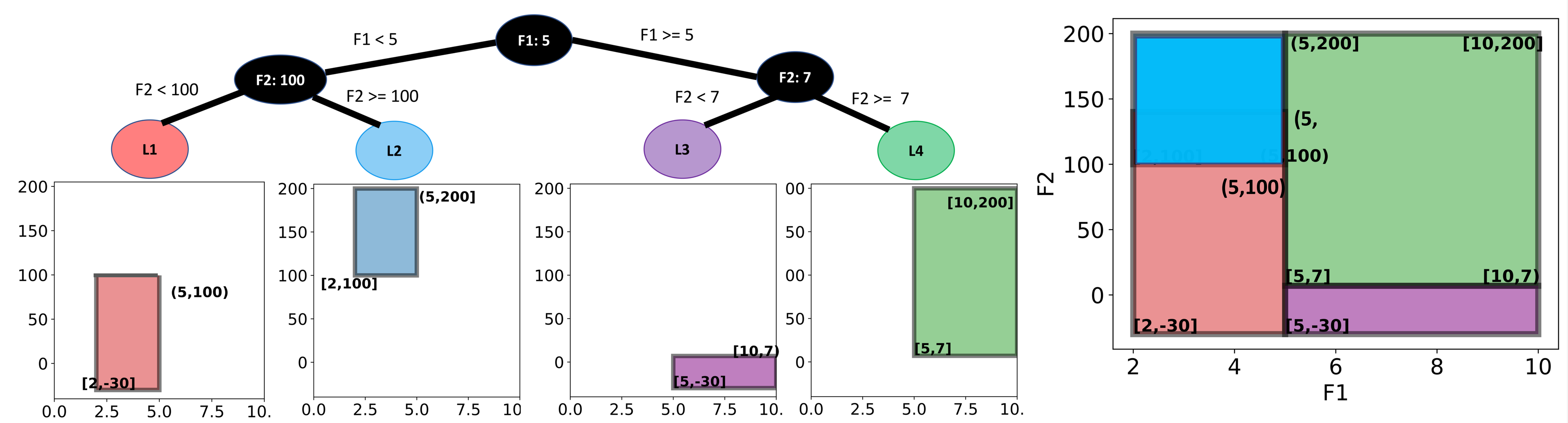
Project 2: T-REX (Tree Rectangles)

OVERVIEW

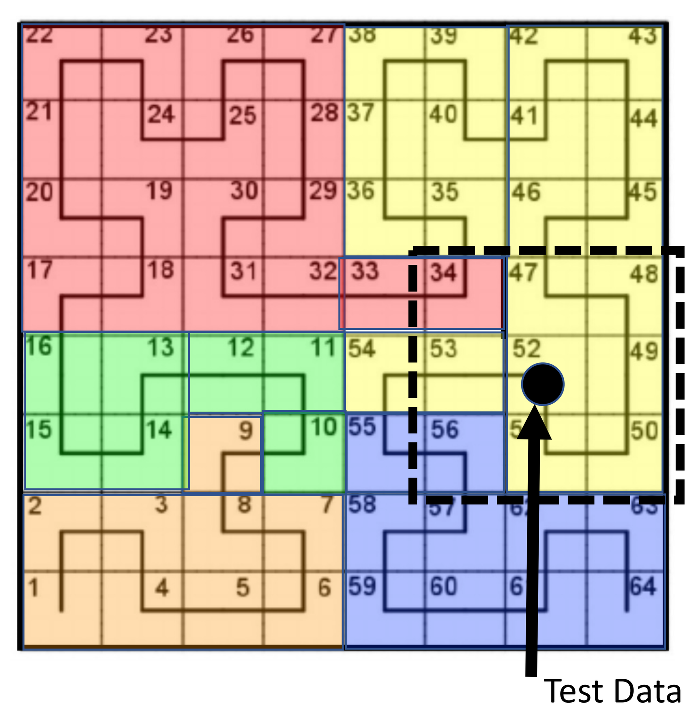
- T-REX trades random I/O for sequential I/O by remapping a random forest of trees into a single spatial index
- We make queries I/O efficient through pruning and space-filling curves.
- We optimize computation through quantization of hyperrectangle boundaries and vectorization of enclosure queries

PROBLEM MAPPING: TREE – HYPERRECTANGLE EQUIVALENCE

The leaves of an axis-aligned decision tree partition the feature space into hyperrectangles. Tree leaves can be represented the upper and lower coordinates of the hyperrectangle in each dimension, which correspond to the minimum and maximum feature split values respectively on the path from the root node to. Each test point is contained in exactly one hyperrectangle per tree.



SYSTEM DESIGN : EFFICIENT HYPERRECTANGE ENCLOSURE QUERIES



I. Distance Filter: Index Construction

1. Order hyperrectangles by the Hilbert number of the projected leaf centroids.
2. Combine the Hilbert cubes into regions along the space-filling curve so that each region contains a block's worth of centroids. The distinct colors represent regions (left) and blocks (below).
3. Build an index (below left) that maps regions of the Hilbert curve blocks.

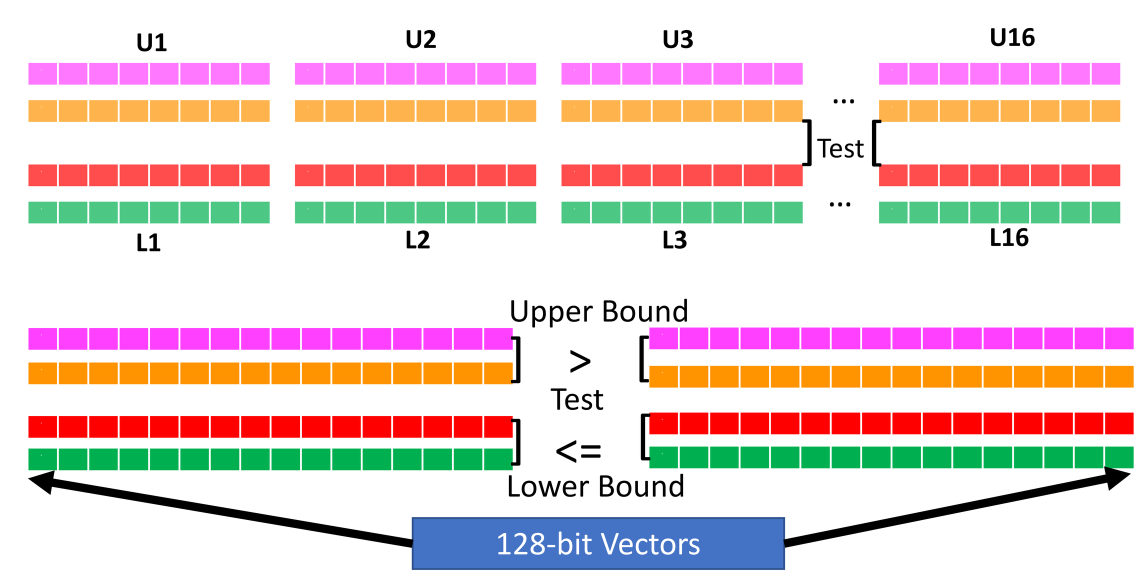
II. Distance Filter: Querying

1. Find the Hilbert numbers of the cube that contains the test observation and all adjacent cubes (dashed box).
2. Retrieve the block addresses of the Hilbert numbers from the index.
3. Retrieve the hyperrectangle boundaries from the blocks (below).

The Hilbert ids corresponding to the test observation and its immediate neighbors are (34,47,48,49,50,51,52,53,56)

Hilbert Indices	Block Addresses
1-9	0
10-16	1
17-34	2
35-54	3
55-64	4

Point enclosure queries are trivially parallelizable across the dimensions of the data and amenable to SIMD parallelism. : Quantized elements from groups of 16 dimensions are packed into 128-bit vectors.



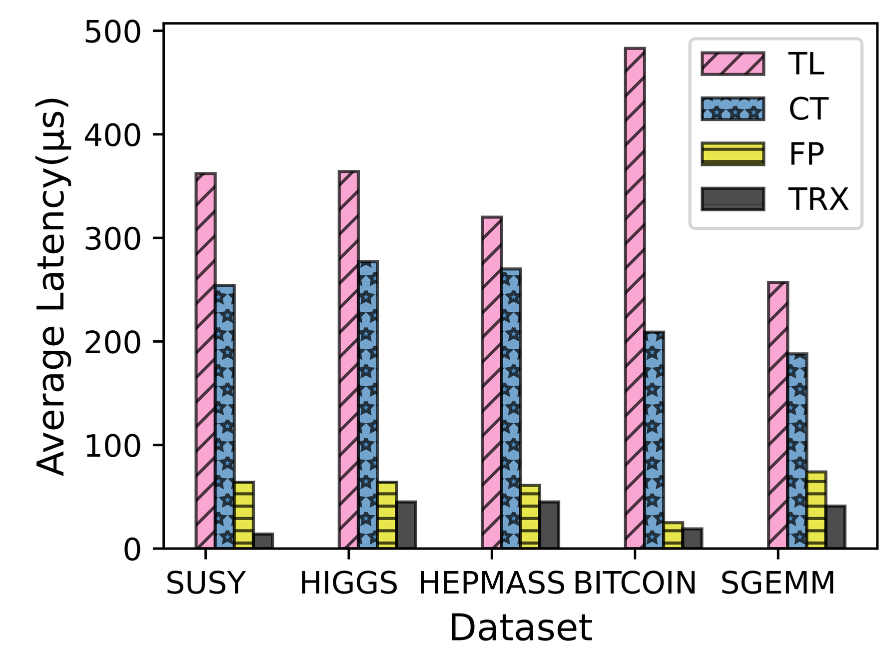
Note:

- **TRD:** Test Rounded Down
- **TRU:** Test Rounded Up

Code to Vectorize Point Enclosure

```
__m128i range_compare(__m128i U, __m128i TRD, __m128i TRU, __m128i L)
{
    return (__mm_cmpgt_epi(U, TRD) |
            __mm_cmpeq_epi(U, TRD) ) &
           (__mm_cmpgt_epi(TRU, L) |
            __mm_cmpeq_epi(TRU, L) ) ;
}
```

RESULTS



T-Rex reduces latency by up to 4 times when compared the cache-optimized trees of forest packing and up to 25 times when comparison with systems that compile trees

CONTRIBUTION 2: A Scalable Model Repository For Transfer Learning With Efficient Tensor Access

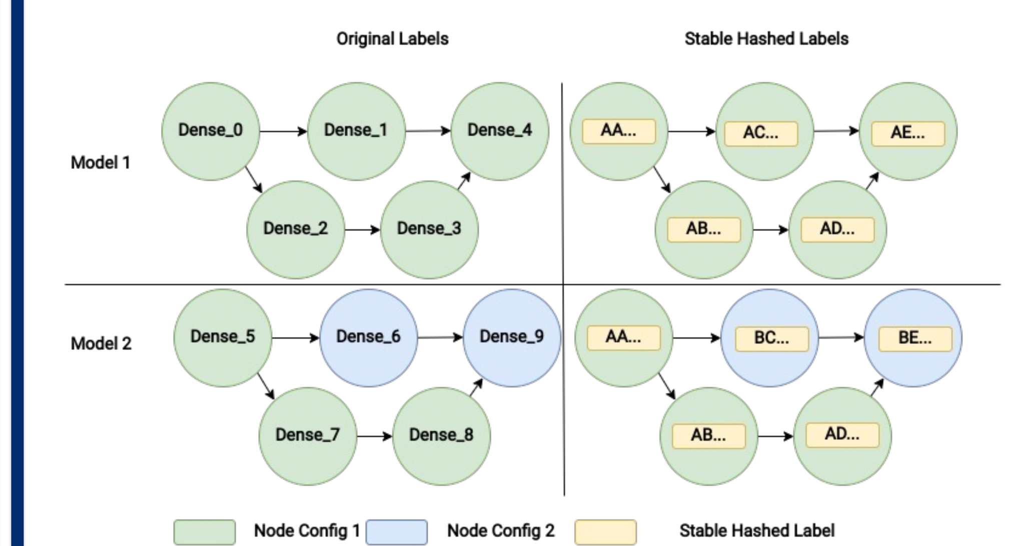
MOTIVATION

- DNN training is expensive.
- To avoid training from scratch models can transfer weights from existing pre-trained models (such as LLMs). This requires fine grained access to tensors.
- We design a model repository that can access tensors at a fine granularity. It is scalable under concurrency.

CONTRIBUTIONS

- We introduce a scalable model repository that stores tensors as key value pairs.
- It supports multiple servers and clients exchanging data via RDMA
- We pre-pin RDMA buffers to avoid the overhead of dynamic allocations and pinning.
- We introduced a stable hashing technique inspired my Merkle trees to provide a consistent naming scheme for model layers.

STABLE HASHING FOR EFFICIENT METADATA



Algorithm 1: Assign Unique IDs

Input : Architecture of model M

Output: A map of all leaf-layer → $unique\_id$

Function $LeafLayerHash(l, L)$:
- $H \leftarrow Hash(l.config)$
- **foreach** $l_i \in l.inputs$ **do**
  - **if** $l_i \notin L$ **then**
    - $L \leftarrow LeafLayerHash(l_i, L)$
  - $H = Hash(H, L[l_i])$
- $L[l] \leftarrow H$
- **return** L

$L \leftarrow \emptyset$

foreach $l \in outputs\ of\ M$ **do**
- $L \leftarrow LeafLayerHash(l, L)$

return L

Key Idea: Hash the structurally significant attributes of the layer together with hashes describing the ancestry of its input dependencies

- enables unique IDs regardless of how many repetitions of layers or entire substructures exist in the architectures