Studying Storage System Backends for ML workloads

Benita Kathleen Britto, Hemalkumar Patel, Reetuparna Mukherjee

PROBLEM STATEMENT

Our project aims to analyze the preprocessing performance impact for various ML workloads, viz. text, image, and graph, by using 3 storage backends - RocksDB, TileDB, and TensorStore, compared to using an ext4 filesystem, which is our baseline.

RELATED WORK

To the best of our knowledge, existing literature focuses on improving the data pipelines - caching, optimized I/O hardware, and data loaders, but the impact of the other available storage systems has not been explored.

APPROACH

- We explore the mentioned data stores and experiment with 3 kinds of workloads - text, image, and graph.
- We transform the data as supported by the data store and thereafter read the data using PyTorch map and iterable-style dataloaders.
- We used the Twitter sentiment analysis dataset for text, CIFAR-100 for images, and FB15K-237 for graphs. To work on larger versions of these datasets, we concatenated the dataset with itself multiple times to understand the storage and retrieval implications as the size of the dataset increases beyond the memory size.
- We also explore the impact of storing embeddings, on these storage backends, and thereafter making random reads-writes.

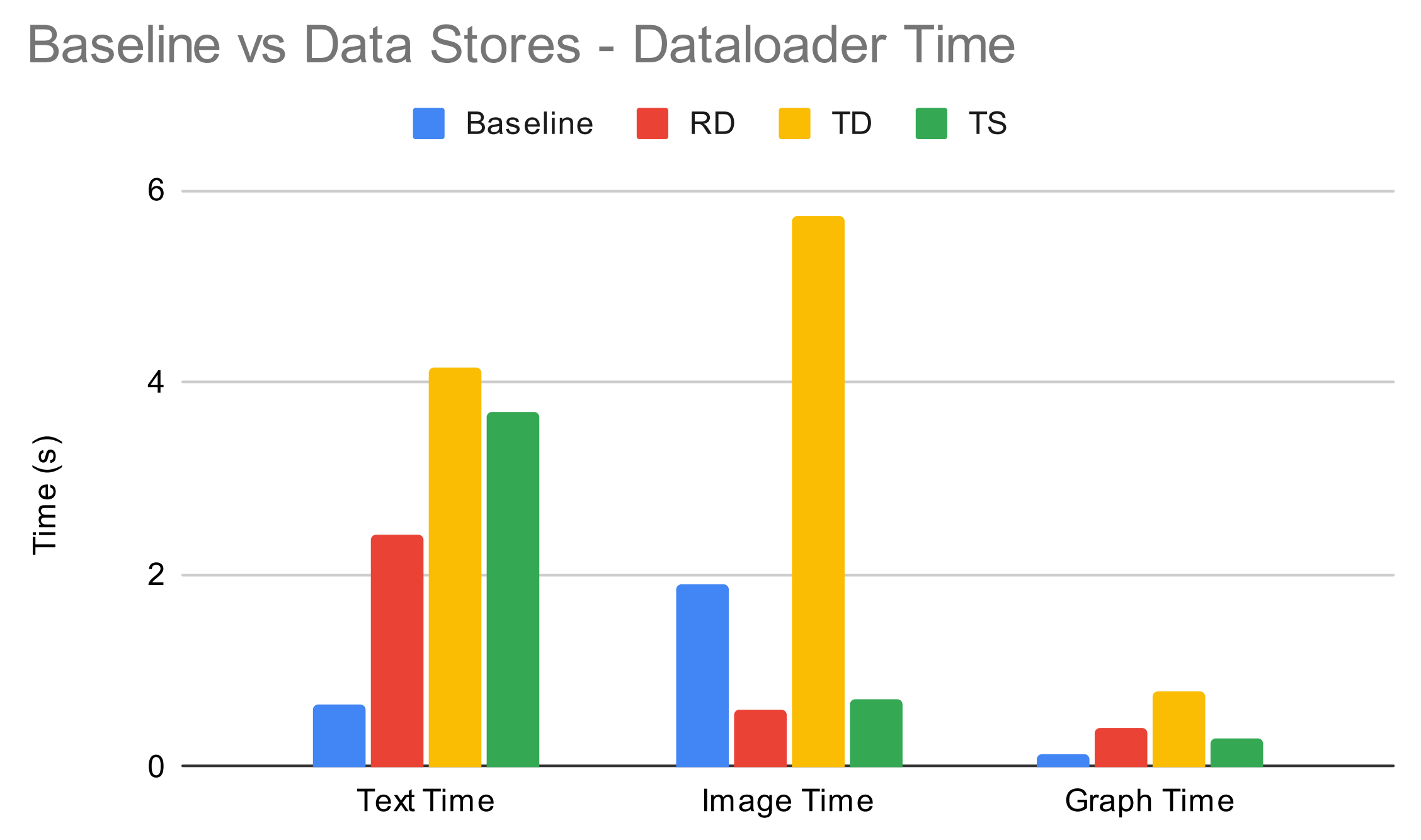
EXPERIMENTAL SETUP

- We experiment with the following parameters for the input data:
  - Map style dataloader: Batch size, Rows per key (Store a batch of input rows per key in RocksDB only, Number of workers
  - Iterable style dataloader: Batch size, Prefetch size, Number of workers
- We experiment with the following parameters for intermediate data: Dataset types (Zipf for text, Power Law for graph), Embedding size, Batch size

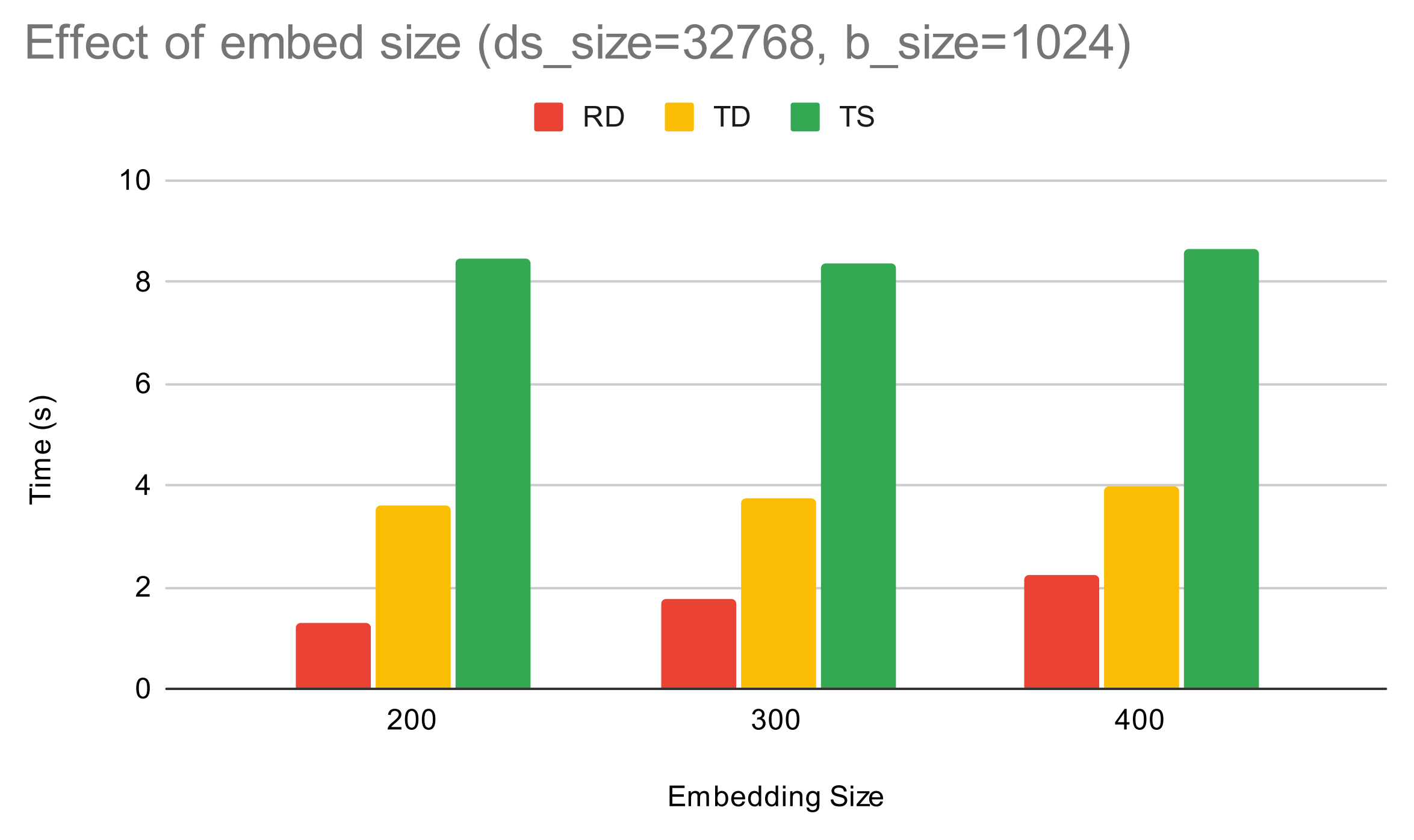
EARLY RESULTS

- **Workload:**
  - For text and graph workloads, the baseline outperforms the datastores as the datasets are small and fit in memory. However, this does not hold true for the image workload.
  - RocksDB (map-style) and TensorStore (iterative-style) outperform the baseline for the image workload. RocksDB performs well as it stores multiple images per key, while TensorStore is effective in handling numeric data.
  - We chose the best results obtained from varying the number of workers, dataloader type, batch size, prefetch size, and rows per key (for RocksDB).

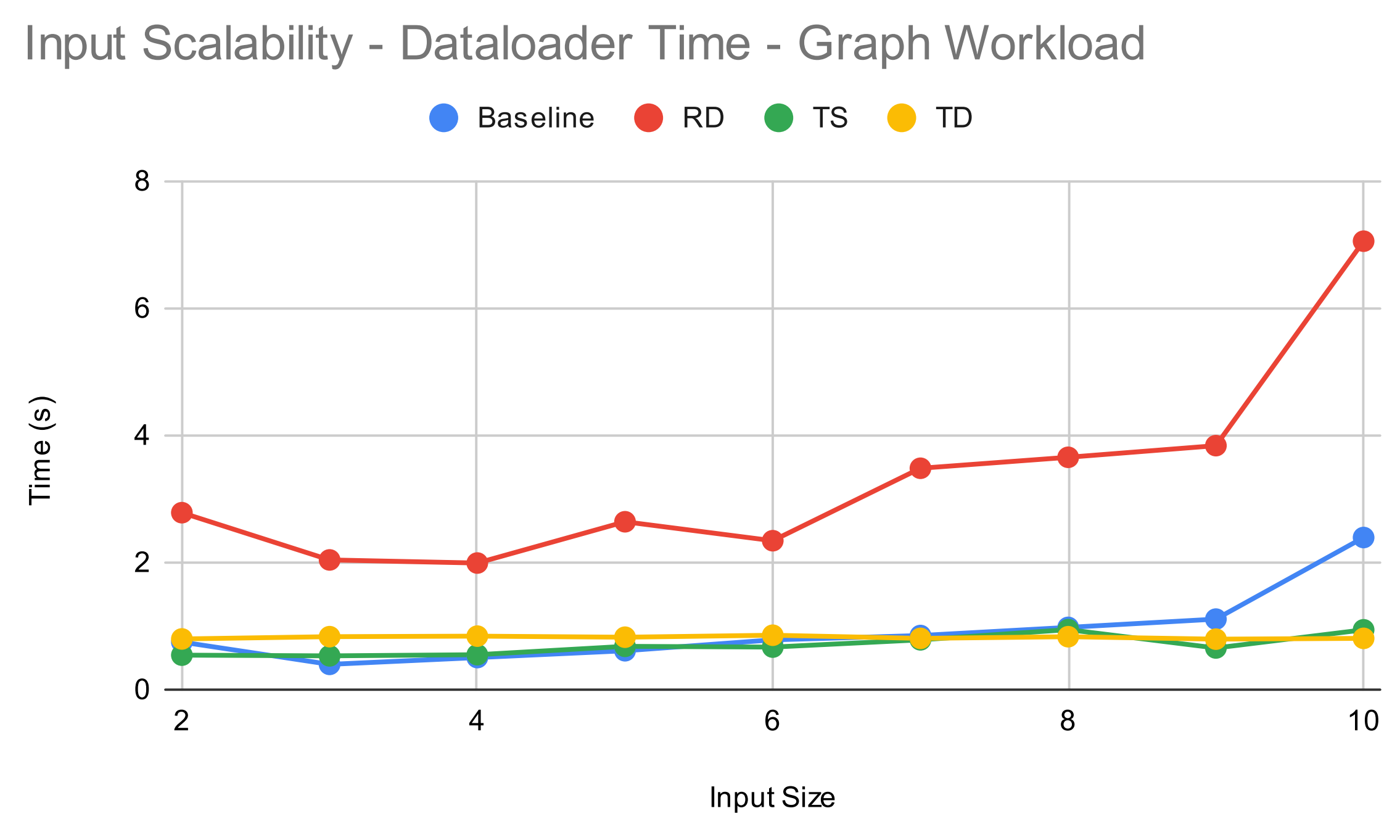
- **Dataloader type:**
  - In terms of data loader types (map vs iterable), map-style works well for RocksDB, iterable-style works well for TileDB, while it depends on the data for TensorStore (i.e. iterable-style is not supported for text-type and graph data).

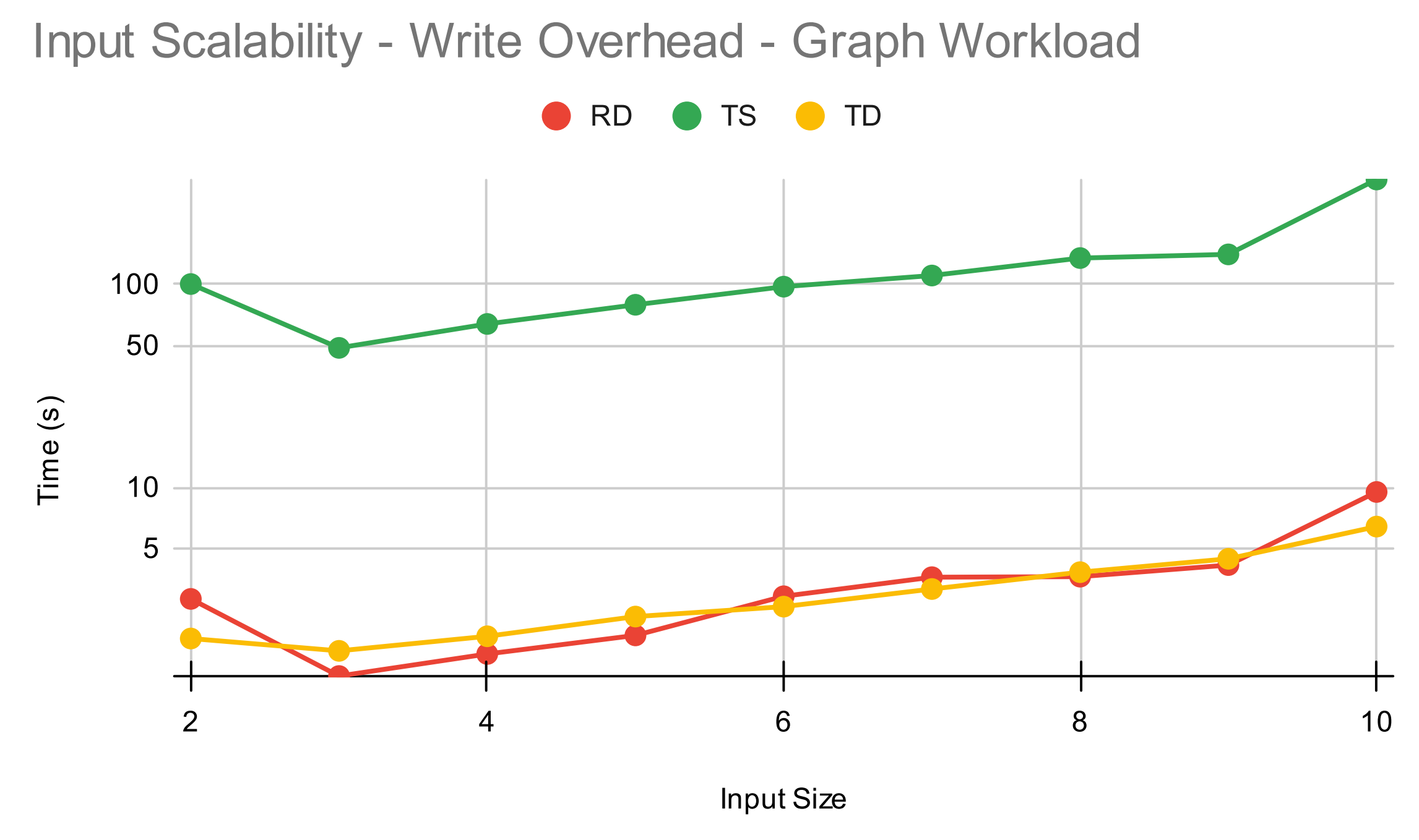


- **Intermediate data/Embedding:**
  - RocksDB performs significantly better than TileDB, which performs better than TensorStore for any configuration.
  - All data stores take double the read time as dataset_size doubles, which is expected considering the number of workers is set to 0 for reading from the file.
  - In the case of RocksDB, read time increases as embedding size increases. Other data stores show no variation. But RocksDB outperforms the other data stores by a significant margin.
  - In the case of TileDB and TensorStore, read time decreases with batch size, while RocksDB shows no variation. But RocksDB outperforms the other data stores by a significant margin.



- **Input data scalability:**
  - To view the effect of input size on these data stores, we augmented the input data. We noticed that on increasing the graph workload 6x, TensorStore (map-style) outperforms the baseline. However, for the text workload, we do not see the data stores outperforming the baseline.





WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON