Michael Chung, Gautam Sethi, David Wang

CSE 598

Dr. Zou

23 April 2024

Problem Formulation

In the modern age, classification tasks are readily used on an everyday basis. From smart cars to spam emails, it is evident that this machine learning problem has a plethora of real-world applications, including identifying bad actors in network traffic. Here, it is significant to be able to find malicious activity as quickly as possible to either eliminate any danger it poses or mitigate the damage it can cause to victims. While many approaches tackle the computational complexity aspect of machine learning algorithms, this project will concentrate on enhancing resource allocation and streamlining relational database operations instead.

Big data tasks face a big challenge with scaling as the data size grows larger, leading to a need for efficient data management and processing. Some other challenges are "storage, security, and scaling" (Simplilearn, 2023). While these are big challenges in big data, Velox focuses on the processing part of big data by accelerating it. This is where database acceleration systems, like Velox, can play a crucial role. Velox can immensely cut down the runtimes for data operations like retrieval and manipulation. Query response times would decrease to a significant degree and overall throughput would be reduced with the use of data acceleration methods like Velox.

As such, this project will focus on the performance comparison between a Velox-assisted machine learning workload against a strictly Tensorflow workload. Velox is an execution engine that quickens data management operations and processing data. With its application to machine learning and AI, Velox can boost the efficiency of classification tasks and outspeed traditional models. Thus, this project will compare the execution time between the two frameworks to assess how much Velox enables accelerated computation and minimizes compilation/run time. The workload will consist of a malware classification task with the utilization of the random forest algorithm. The dataset explains the links between flows connected with harmful or malicious activity. In this project, we are looking to use binary classification labels between benign(0) and malicious(1). By using various input features including, but not limited to, the network protocol, service, and byte sizes, trees will be constructed with the appropriate thresholds to decide whether or not the entry is malware-related. Lastly, The outcome of this project will be to observe the effectiveness of Velox on machine learning tasks as well as serve as a motivator to imbue Velox into other common applications of machine learning and artificial intelligence.

Literature Survey

To gain better insight into how one can use a random forest to detect malware threats, a research paper was read that determined the top significant factors for classifying malicious activity via the recursive feature elimination strategy and implementing the random forest algorithm in R (Joshi et al., 2018). For feature selection, they measured the variable importance through the mean decrease in accuracy where cross-validation was used to drop the lowest k features. This could be important for enhancing the execution time of the project's model since one can eliminate insignificant features while not impacting the accuracy. Furthermore, they determined that the model can predict malware with an accuracy of 90.0% of testing data that contained a total of 62,999 records (Joshi et al., 2018). Given these results, the random forest strategy is likely a viable model to use for the project, and this paper also gives ideas on how to optimize the model even further.

The focus of this project will be to understand how VeloxML can enhance a strictly TensorFlow-based decision forest. However, before exploring the advances that Velox has made, it is vital to understand the TensorFlow decision forest implementation. The Yggdrasil decision forests (YDF) library was created in 2018 and is available in multiple languages including C++, Javascript, Go, and "Python (under the name of TensorFlow Decision Forests)" (Guillame-Bert et all., 2023). The YDF learners were able to handily outperform native TensorFlow learning models (BoostedTree Estimators, Linear), XBoost, LightGBM, and SKLearn in cross-validation accuracy across 70 different datasets (Guillame-Bert et all., 2023). Additionally, the YDF learners only trailed XGBoost in inference time on the datasets, while the previously mentioned TensorFlow models came in last (Guillame-Bert et al., 2023). Thus, YDF shows real promise in its ability to deliver top-of-the-line performance without sacrificing execution time performance and will be the library utilized for this project. When trying to understand the improvements that Velox can make with machine learning applications, its performance must be put into context with the current ML libraries that are at the top of the industry.

The 'Velox: Meta's Unified Execution Engine' paper was about VeloxML and how it is used to address challenges in the proliferation of specialized computation engines in the data landscape. It is a C++ database acceleration library. Velox has reusable, high-performance, and dialect-agnostic components for building execution engines and enhancing data management systems. Velox is integrated with several other technologies such as Presto, Spark, PyTorch, Xstream, etc….(Velox: Meta's Unified Execution Engine). It aims to streamline and unify data processing workflows across diverse workloads. This paper also went over how Meta created the workflow and engineering design for Velox. The functions used in Velox such as scalars and aggregates. Operators, such as table scans, aggregate, hash joins, and memory management methods were mentioned.

The function types mentioned in the paper were scalar and aggregate functions. Scalar functions operate on individual values in the dataset. They have a wide range of operations such as arithmetic operations and string manipulation. This allows for easier vectorization techniques in machine learning. Aggregate functions operate on a set of values. This is usually used in summary statistics or aggregating results across groups of data. Examples of this include the sum, count, and average functions. The operator types mentioned were TableScan, Filter, Aggregation, OrderBy, HashJoin, MergeJoin, and many others. Operators work together to execute the queries efficiently and handle many data manipulation tasks that are vital to Velox's function.

Solution

This project will involve comparing two different approaches that leverage TensorFlow and XGBoost random forest implementations. The first approach will include a purely TensorFlow-based approach that utilizes the Yggdrasil-based Decision Forests for both training the forest on the given malware detection dataset and classification predictions on the training dataset. Specifically, the team will use the RandomForestLearner algorithm, training it on the eighteen million entries in the training dataset. The second approach will use the XGBoost library to train a standalone random forest on the training dataset and load the weights for this random forest into a VeloxML project. The Velox project repository provided to the team features a few ML functions that can be used with the database engine, namely the Decision Forest function. This VeloxML function generates and registers multiple vector functions, namely the 'velox_decision_tree_construct' and 'velox_deicsion_tree_predict' functions that are crucial to implementing decision forest prediction capabilities within Velox. Then, a decision forest will be created using the XGBoost model's weights which are fed into a 'vectorizeForestFolder' function that provides the tree paths based on the model's weights. The Velox vector functions mentioned earlier will leverage these trees and are a key part of the query created and run on the test data for the malware detection dataset. This approach for testing the VeloxML approach is based on the DecisionForestTest.cpp file from the asu_cactus organization on GitHub. Additionally, a key part of understanding how these two approaches perform with inference tasks is exploring execution times for different-sized forests. The team will run two different versions of the decision forest, one that has 10 trees and the other that has 100 trees, as a way to better understand which framework scales better. Each of these versions will be tested with two different tree_depth values to understand how this hyperparameter also scales between the two frameworks.

Experimental Environment / Experiment Setup:

For our experimental environment, we utilized a r4.2xlarge AWS Ubuntu EC2 instance to run both Velox and TensorFlow frameworks. This is to ensure a controlled baseline for comparative analysis between the two models. The instance features an Intel Xeon processor and roughly sixty-two gigabytes of memory. Moreover, the team used SSH network protocol to access the instance to conduct our tests. Lastly, this environment was chosen since running the Velox framework demands a significant amount of CPU, GPU, and memory resources. This instance contains the necessary hardware components to run such programs that the team's local machines either could not handle completely or at an extremely slow pace.

By leveraging the AWS Ubuntu EC2 instance, we compared the performance of the two approaches on the malware detection dataset. For the Velox approach, the team utilized the XGBoost library to create random forest models with various parameters such as the number of trees and max depth. The team then exported these model weights via the graphviz library, modified the provided DecisionForestTest.cpp file found in the Velox repository, and tested certain functions with the newly imported model weights. For the TensorFlow approach, the team created the models utilizing the Yggdrasil Decision Forests library and its RandomForestLearner() function. This approach was solely implemented in Python. In both approaches, time was kept track of according to their unique steps. For Velox, parsing the CSV file, building the row vectors, constructing the forest, and inference time were all recorded. On the other hand, the TensorFlow approach noted the time it takes to complete the data ingestion and inference stages. For CPU utilization and memory tracking, the "sar -u 1" and "vmstat -a 1" commands respectively displayed the results every second.

Evaluation Results

Five major comparisons are made to test the performance between the Velox and TensorFlow models. This included the CPU utilization percentage and memory usage with max depth and number of trees as control variables, the model runtime with max depth as the control variable, the model runtime with number of trees as the control variable, and the data ingestion and inference runtime. These results are shown below:

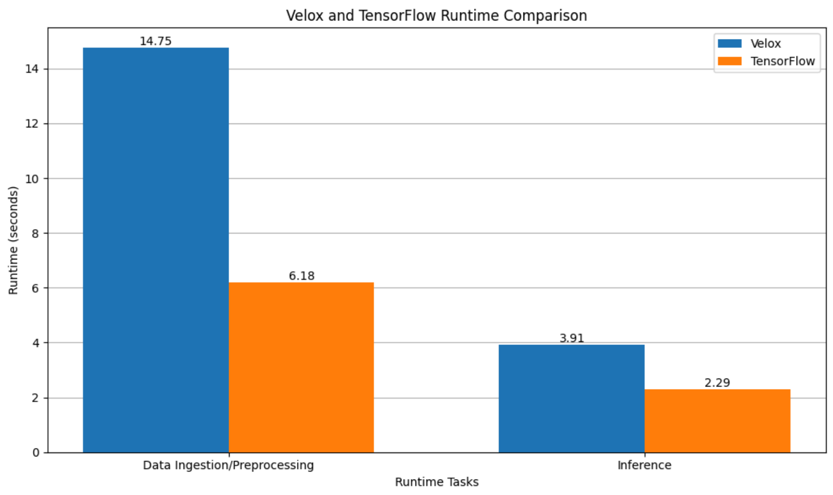

Figure 1

Firstly, a comparison was made between the overall data ingestion and inference stage time between Velox and TensorFlow in Figure 5. In this case, TensorFlow had both lower data ingestion and inference time in comparison to Velox. Because Velox is a framework that assumes that the database is already prepared to be used in building the models, it does not optimize the data ingestion task nearly as well as TensorFlow and could also impact the inference time. This is one possible explanation for why TensorFlow did better on both stages than Velox.

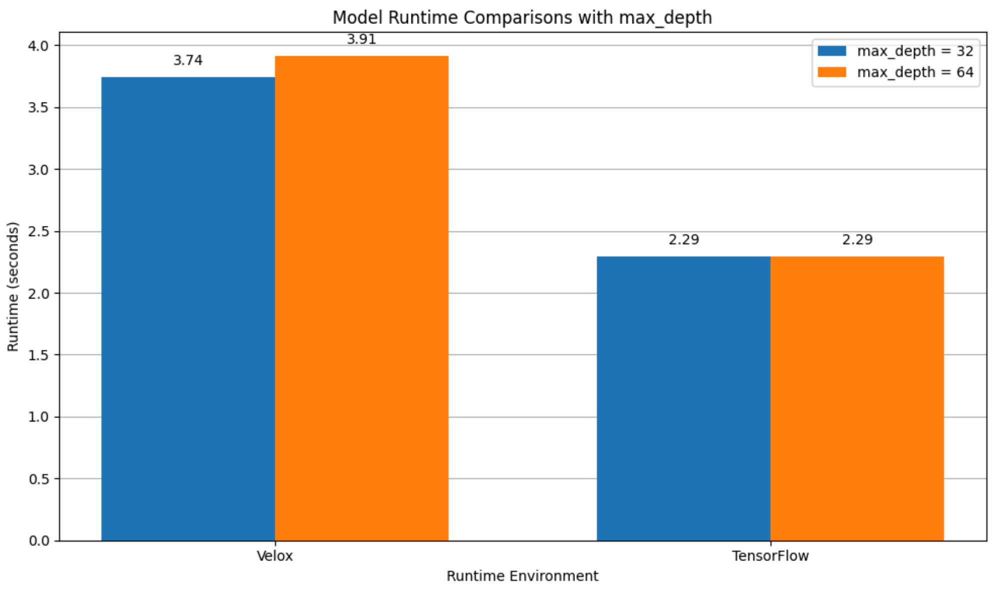

Figure 2

For Figure 2, it is evident that the max depth of the trees is irrelevant to the runtime. This is because when the model is being built in both the Velox and TensorFlow frameworks, the depth of the decision forest model never exceeds thirty-two. As such, max depth has virtually no impact on the runtime.

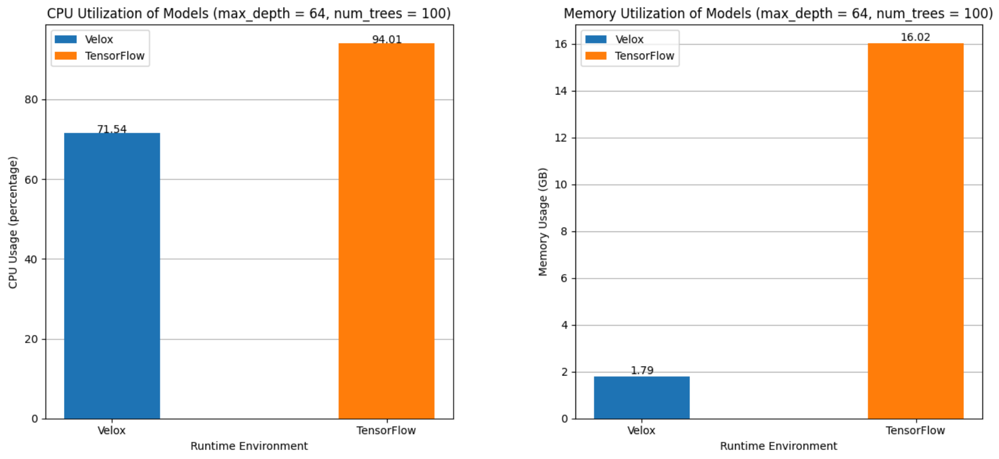

Figure 3 & 4 (respectively)

In Figure 3, Velox significantly outperformed TensorFlow in CPU utilization by roughly 22.5% given the same max depth and number of tree parameters. Furthermore, this also suggests that Velox has better scalability compared to TensorFlow as it has less CPU utilization when there are more trees involved in the random decision forest. Hence, Velox optimizes CPU operations more effectively through its plan building and custom functions. Figure 4 displays the memory utilization that also kept the max depth and number of trees static across both frameworks, and the results demonstrate that Velox is approximately fourteen gigabytes more efficient than TensorFlow in terms of active memory, making it astronomically better than TensorFlow. This could also be attributed to Velox's plan building and custom functions for tree construction and prediction.

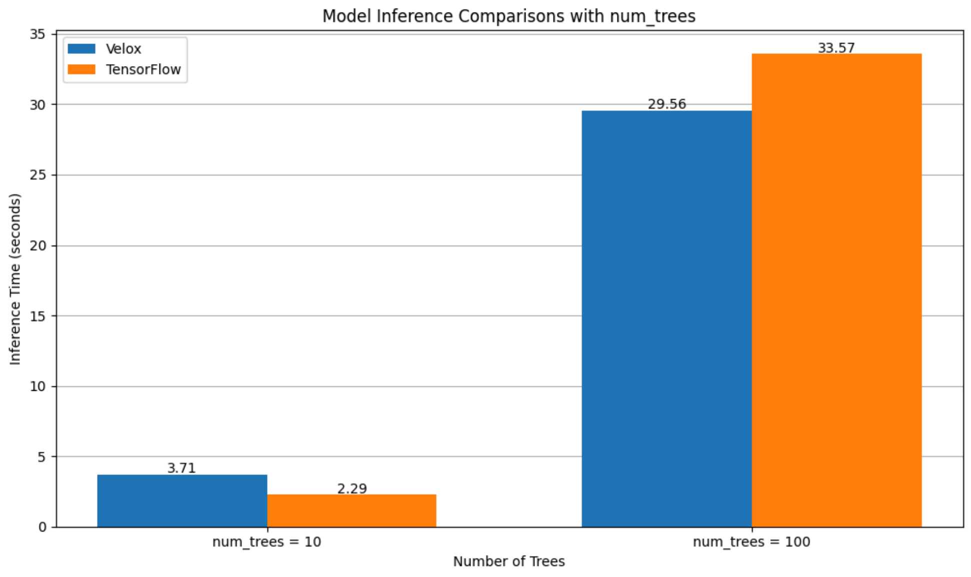

Figure 5

Next, Figure 5 further supports the notion that Velox has better scalability compared to TensorFlow. When comparing inference times between both models and keeping the max depth parameter as the control, Velox has a lower time than TensorFlow. One interesting observation is that on models with a lower number of trees, TensorFlow outperformed Velox in inference time. A possible explanation is that there may be additional overhead when building the plan in Velox, unlike the TensorFlow model. However, the plan building becomes advantageous when scaling up the number of trees as evidenced by Figure 5.

Overall, Velox surpassed TensorFlow in CPU and active memory utilization and scalability. In use cases where data has not been preprocessed or ingested, TensorFlow overcomes Velox in terms of time. However, it is evident that Velox, when properly prepared with data, exceeds TensorFlow's limitations. As a natural consequence, Velox is recommended for optimizing machine learning models such as decision trees.

Future Works

One area to do for future work is to look at changes in performance for multi-class classification with decision forests. Velox's database acceleration capabilities can be significantly impacted with decision forests and allow for results that show much better performance compared to TensorFlow. Metrics of interest include training time, inference speed, and resource utilization to see how efficient each framework is in these benchmarks.

Another area for future work is changing the testing environment to have multiple GPUs. Multiple GPUs can significantly cut down on the runtime of these data tasks. It would be worthwhile to look at how much improvement can be achieved on both methods and compare how much improvement is achieved on both Velox and TensorFlow. Velox will likely benefit the most from using multiple GPUs as it can already distribute tasks across multiple GPUs, enhancing its advantages over TensorFlow.

Lastly, it is in the best interest of Velox developers to compare different testing data to see how robust the models are. In particular, the observation of Velox's data acceleration on other data types and what type sees the biggest runtime decrease from using Velox are topics worth investigating. Velox can run with "physical, logical, custom, and complex types among plenty of other types" (Facebook Incubator, n.d.). This is how Velox is robust but needs to be put to the test. Changing the model architecture when using Tensorflow would also need to be taken into consideration to see how it can best accommodate different data types. By testing through these three various experiments, we can get better insight into how Velox operates and get a holistic view of how it compares to TensorFlow.

Conclusion

The focus of this report is to understand the differences in inference capabilities between Velox and TensorFlow. With the rise of popularity for machine learning applications, it has become more and more crucial to find ways to optimize big data pipelines, and this report explores the possibilities that Velox can offer in this context. Figure 1 shows that it took Velox much longer to ingest and process the data; however, Velox assumes that the data is already in the cache. When it comes to inference time, regardless of tree_depth, it became clear that there might be some overhead associated with the use of the Velox query and Hive Splits. Nevertheless, Velox performed better than TensorFlow when scaling the size of the decision forest. While outperforming TensorFlow for inference tasks with a hundred-tree decision forest, Velox used less than an eighth of the RAM and had a lower CPU utilization. Clearly, there is a lot of potential with leveraging Velox for big data tasks given its capabilities as a database acceleration engine, and it requires much fewer resources while outperforming TensorFlow for a much more dense decision forest. Many modern big data pipelines could leverage this framework due to its performance and efficiency.

References

Guillame-Bert, M., Bruch, S., Stotz, R., & Pfeifer, J. (2023, August). Yggdrasil decision forests: A fast and extensible decision forests library. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (pp. 4068-4077).

Santosh Joshi, Himanshu Upadhyay, Leonel Lagos, Naga Suryamitra Akkipeddi, and Valerie Guerra. 2018. Machine Learning Approach for Malware Detection Using Random Forest Classifier on Process List Data Structure. In Proceedings of the 2nd International Conference on Information System and Data Mining (ICISDM '18). Association for Computing Machinery, New York, NY, USA, 98–102. https://doi.org/10.1145/3206098.3206113

Simplilearn. (2023, February 28). *Challenges of Big Data: Basic concepts, case study, and more*. Simplilearn.com. https://www.simplilearn.com/challenges-of-big-data-article

Pedro Pedreira, Orri Erling, Masha Basmanova, Kevin Wilfong, Laith Sakka, Krishna Pai, Wei He, Biswapesh Chattopadhyay. 2022. Velox: Meta's Unified Execution Engine. Proceedings of the VLDB Endowment, Volume 15, Issue 12. https://vldb.org/pvldb/vol15/p3372-pedreira.pdf

Types. Types - Velox documentation. (n.d.). https://facebookincubator.github.io/velox/develop/types.html