Accelerating End-to-End Machine Learning Using the Intel® oneAPI Toolkit and Intel® Flex Series GPU

January 2023

H19432

White Paper

Abstract

This white paper describes the accelerated performance of an E2E ML pipeline using Intel's oneAPI AI Analytics Toolkit as compared to a baseline pandas implementation.

Dell Technologies

Copyright

Contents

Executive summary

Overview

An end-to-end (E2E) Machine Learning (ML) pipeline consists of three general stages:

- Extraction, Transfer, and Loading (ETL) of data
- Model Training
- Inference and Visualization

These stages are generally computationally expensive, which impacts the performance and effectiveness of the resulting ML pipeline. The goal of this project is to accelerate the performance of an E2E ML pipeline using Intel's oneAPI AI Analytics Toolkit and compare its performance against a baseline pandas implementation[1].

In evaluating the ETL stage (the most time-consuming stage in the pipeline), we compare the ETL performance when implemented using Modin (with Ray, Dask, or Omnisci backends) against the native pandas library. We showed that all three Modin backends performed better than the baseline serial panda library, with Modin-Dask the best performing (~5x speedup). This experiment verified that the Modin API can efficiently make use of all available CPU cores to process data in parallel, thus allowing Modin to support the pandas API efficiently. Based on this result, an optimized E2E ML pipeline was implemented and deployed on a computing node of the Intel® DevCloud (consisting of an Intel® Ice Lake CPU and a 150W Intel® Arctic Sound GPU), to test the acceleration of large-scale workloads using Intel's OneAPI toolkit.

The optimized implementation uses Modin-Dask to accelerate the ETL stage and oneDAL to accelerate the Inference stage. The comparison was performed using a NYC Taxi historical dataset (~64GB) for ML regression analysis; and a Higgs dataset (~7.5GB) for the classification problem. The optimized E2E ML pipeline was compared against an unoptimized baseline version (using native pandas and XGBoost libraries). The ETL stage of the E2E ML pipeline achieved ~5x speedup for the NYC dataset and ~3x for the Higgs dataset. For the backend Inference stage, a significant improvement was also seen (~3x for NYC dataset, ~2.25x for Higgs dataset) when using oneDAL with XGBoost in an optimized ML pipeline. The reason is that oneDAL utilizes all the capabilities of the Intel hardware by using Intel® Advanced Vector Extensions 512 (Intel® AVX-512) vector instruction set to maximize the utilization of the Intel® Xeon® processors. Finally, experiments were performed for both datasets to observe the speedup trends as the dataset sizes were varied.

Revisions

Date	Description
January 2023	Initial release

[1] Data structures for statistical computing in Python, McKinney, Proceedings of the 9th Python in Science Conference, Volume 445, 2010.

We value your feedback

Dell Technologies and the authors of this document welcome your feedback on this document. Contact the Dell Technologies team by email.

Authors:

Carol Pagolu, SHREC Center[2], University of Florida
Manjari Misra, SHREC Center[1], University of Florida
Bhavesh Patel, Dell Technologies
Herman Lam, SHREC Center[1], University of Florida

Contributors: Trokon Johnson, SHREC Center[1], University of Florida

Note: For links to other documentation for this topic, see the Artificial Intelligence Info Hub.

[2] SHREC: NSF Center for Space, High-Performance, and Resilient Computing

Introduction

An end-to-end (E2E) Machine Learning (ML) pipeline consists of three general stages: (1) Extraction, Transfer, and Loading (ETL) of data; (2) Model Training; and (3) Inference and Visualization, as illustrated in Figure 1. These stages are generally computationally expensive, which impacts the performance and effectiveness of the resulting ML pipeline.

The goal of this project is to accelerate the performance of an end-to-end ML pipeline using Intel's oneAPI AI Analytics Toolkit [1]. The oneAPI AI Analytics toolkit is a set of powerful and familiar Python packages and frameworks which are optimized to deliver drop-in acceleration for Intel architectures. With oneAPI libraries built for low-level compute optimizations, it is possible to achieve highly efficient multithreading, vectorization, and memory management, and to scale scientific computations efficiently across a cluster.

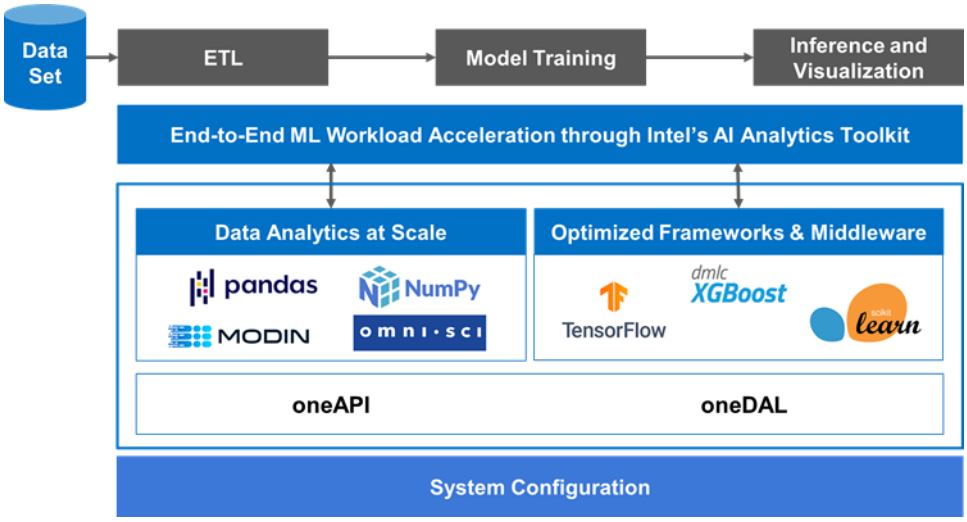

Figure 1. End-to-end ML Pipeline using Intel's oneAPI AI Analytics Toolkit

Some of the key features of this toolkit, as shown in Figure 1, are:

1. Data Analytics at Scale:
   - Delivers an excellent out-of-the-box developer experience using open, optimized software tools coupled with optimal compute and memory hardware configurations for AI applications.
   - Considerable reduction in time for preprocessing: sort, filter, label. and transform data using high-memory systems that can accommodate large datasets efficiently.
   - Accelerate pandas workloads across multiple cores and multiple nodes using Intel's distribution of Modin.
2. Optimized Frameworks and Middleware:
   - Efficient data transfer performance across packages through efficient copy data exchange with data-parallel Python technology.

- Intel optimized frameworks for TensorFlow and PyTorch, pre-trained models, and low-precision tools to deliver high-performance training on Intel XPUs and incorporate fast inference into the AI development workflow.
- Integrated Intel performance libraries such as Intel® oneMKL (Math Kernel Library) and Intel® oneDAL (Data Analytics Library) to accelerate NumPy, scikit-learn and XGBoost.

The oneAPI AI Analytics Toolkit [1] is implemented using the oneAPI Data Analytics Library (oneDAL), a powerful machine learning library that helps speed up big data analysis. oneDAL is an extension of Intel® Data Analytics Acceleration Library (DAAL) and is a part of oneAPI. oneAPI is a cross-industry, open, standards-based unified programming model that delivers a common developer experience across accelerator architectures. For more details about these libraries, see the References section for oneAPI [2] and oneDAL [3].

This study was conducted using two datasets: the New York City (NYC) Taxi historical dataset [4] and the Higgs dataset [5]:

- The NYC dataset is used to study the performance of the regression function for machine learning. The data was collected and provided to the NYC Taxi and Limousine Commission (TLC) by technology providers authorized under the Taxicab & Livery Passenger Enhancement Programs (TPEP/LPEP). It contains historical records accumulated and saved on individual monthly files from 2014 to 2016 (total: ~64GB).
- The Higgs dataset is used for classification problems to distinguish between a signal process which produces Higgs bosons and a background process which does not. The data was produced using Monte Carlo simulations. The first 21 features (columns 2-22) are kinematic properties measured by the particle detectors in the accelerator. The last seven features are functions of the first 21 features and are high-level features derived by physicists to help discriminate between the two classes. The Higgs dataset has 11 million instances (total: ~7.5GB).

As stated above and illustrated in Figure 1, the E2E ML pipeline consists of three general stages. The Extraction, Transfer, and Loading (ETL) stage is the first stage of the pipeline. In this stage, the data is extracted from different sources and transformed into the required format to be used for the Model Training stage. Because the ETL stage is the most time-consuming stage in the pipeline, it is essential to optimize this stage. In the section Evaluation of oneAPI compatible ETL libraries in this paper, we describe the experiments that we performed to evaluate the various oneAPI-compatible ETL libraries. We compared the ETL performance when implemented using Modin [6] (with Ray, Dask, and Omnisci) and the native pandas library. The performance comparison is further broken down into the most commonly used ETL functions: df.read_csv, df.rename, df.drop, df.fill_na, and df.concat. Based on the performance advantage of Modin-Dask, we optimized the ETL stage of our ML pipeline using the Modin-Dask library.

The optimized end-to-end ML pipeline was implemented and deployed on a computing node of the Intel® DevCloud [7], containing an Intel® Ice Lake CPU (with 96 CPU cores) and a 150W Intel® Arctic Sound GPU. In the section Optimized End-to-End ML Pipeline vs. Serial Pipeline, the performance of the optimized implementation (using Modin-Dask and oneDAL) is compared against the unoptimized version (using native panda and

XGBoost libraries), with a performance breakdown by the three stages in the pipeline. The comparison is performed for both a NYC dataset (for ML regression analysis) and a Higgs dataset (for classification problem). In the section Evaluation of ML Pipeline with Varying Dataset Sizes, the optimized code is then used to study the pipeline performance based on varying dataset size. The NYC dataset was varied from 2 GB to 64 GB (increment of power of 2). The Higgs dataset was varied from 1 GB to 7.5 GB (increment of 1 GB).

Evaluation of oneAPI compatible ETL libraries

As stated, the ETL stage is the most time-consuming stage in the ML Pipeline and thus, it is essential to optimize this stage. In this section, we compare the ETL performance when implemented using Modin ([8] (with Ray, Dask, and Omnisci backends) against the native pandas library. Because the performance of the ETL stage depends mainly on the size of the dataset, the evaluation for the ETL stage was performed using the NYC Taxi dataset at 64GB. Also, we determined experimentally that the best performance occurs when the maximum number of CPU cores (96) are used to process the 64 GB dataset. Thus, all 96 CPU cores are used when studying the NYC dataset.

As shown in Figure 2, for the ETL stage of the ML pipeline, all three Modin backends performed better than the baseline serial panda library. Modin-Dask was the best performing, with a ~5x speedup vs. the serial pandas library. These results are as expected because the Modin API can efficiently use all available CPU cores to process data in parallel. Additionally, Modin promotes parallelism by supporting both row and column cell-oriented partitioning of a dataframe. The Modin API transparently reshapes the partitioning of the dataframe based on the operation being performed. This allows Modin to support the pandas API efficiently. Modin's fine-grained control over partitioning allows it to support various pandas operations that are otherwise very challenging to parallelize. Also, Modin with Dask as backend is the fastest because it supports more of the pandas API as compared to the Ray and OmniSci and hence minimal operations must be defaulted to its original pandas implementations.

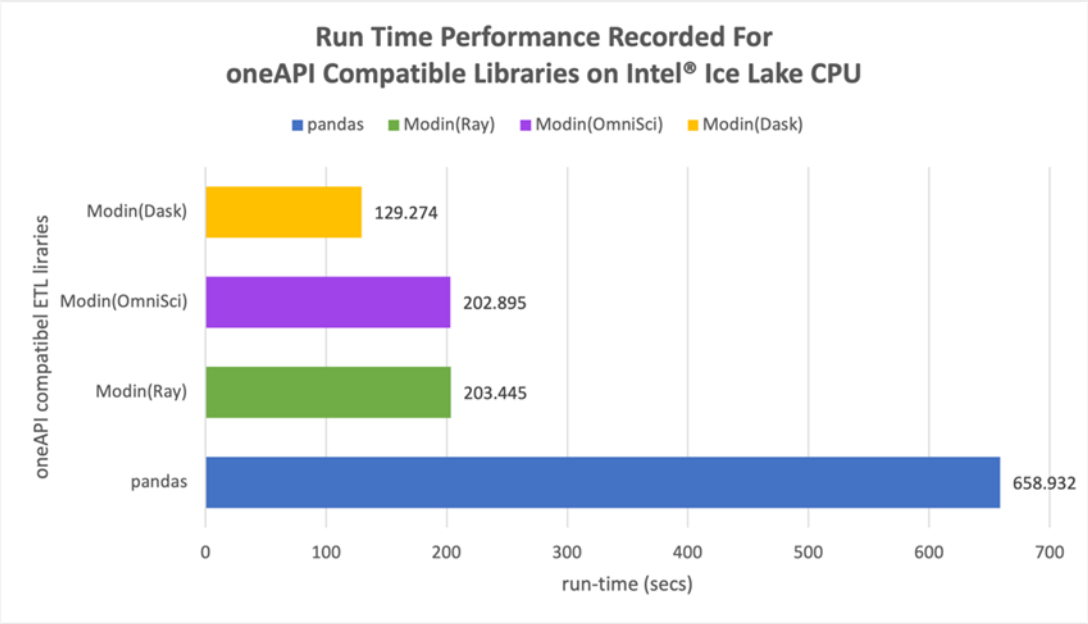

Figure 2. Performance comparison of Modin libraries and native pandas library

The performance comparison is further broken down into the most commonly used ETL functions: df.read_csv, df.rename, df.drop, df.fillna, and df.concat [8]. The results are shown in Figure 3 and Figure 4. Figure 3 shows the run time (in secs) of each ETL function for the serial pandas library vs. Modin-Dask. The corresponding speedups can be observed in Figure 4. Note that df.drop (~110x) and df.rename (~75x) have tremendous speedups, however, their corresponding run times are very small, thus contributing very little to the overall speedup. The most time consuming ETL functions shown in Figure 3 are df.read.csv and df.concat, which have speedups of ~4.6x and ~9.4x, respectively. Finally, the performance of the overall ETL stage (~5x shown in Figure 2) is limited by the serial performance of another ETL function df.query, which is used to divide the dataframe into train and test samples. Because this function is not completely supported by Modin, there is a computational penalty involved to convert Modin's partitioned dataframe to serial pandas.

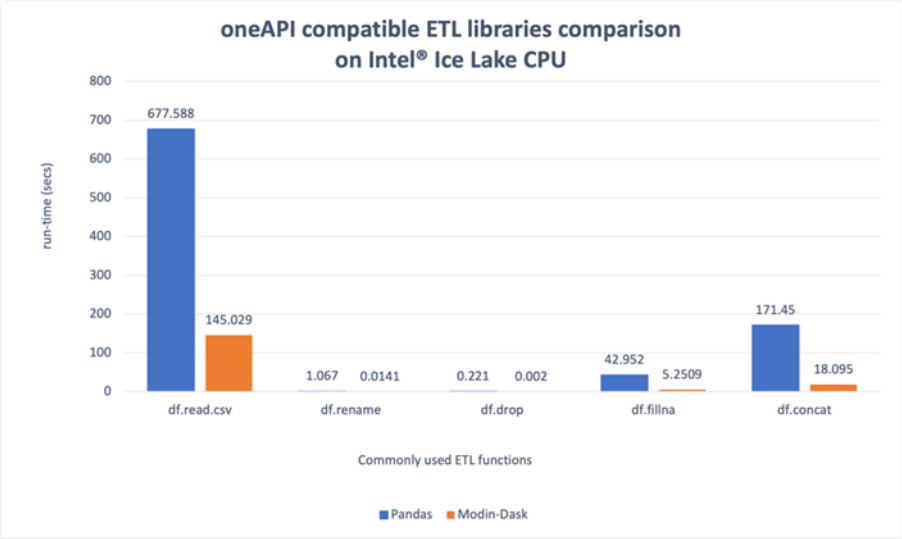

Figure 3. Run time of ETL functions: Modin-Dask vs. native pandas

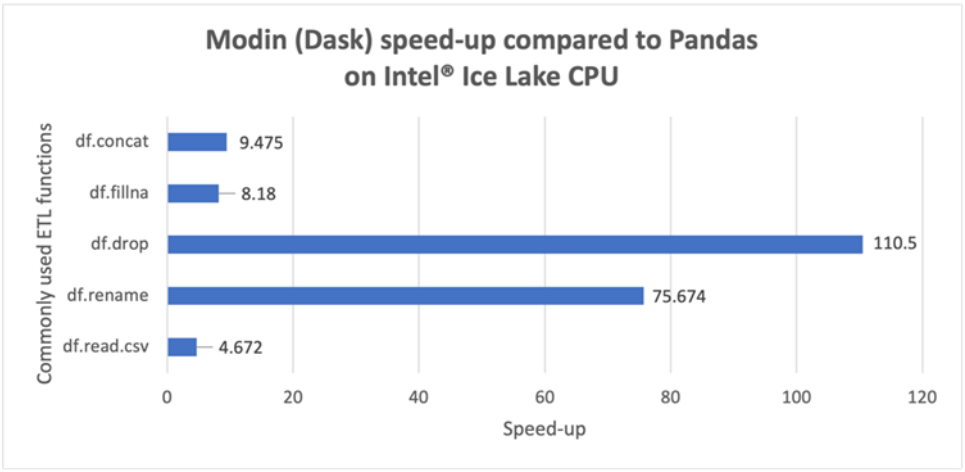

Figure 4. Speedup using Modin-Dask vs. native pandas

Based on these results, we optimized the ETL stage of our ML pipeline by using Modin-Dask. In the next section, the optimized implementation of the ML pipeline is compared to the performance of a serial baseline pandas implementation on an Intel® DevCloud node.

Optimized End-to-End ML Pipeline vs. Serial Pipeline

In this section, the performance of the optimized implementation (using Modin-Dask and oneDAL) is compared against the unoptimized version (using native panda and XGBoost [9] libraries), with a performance breakdown by the three stages in the pipeline. The comparison is performed for both the New York City (NYC) Taxi dataset and the Higgs dataset. The NYC dataset is used to study the performance of the regression function; the Higgs dataset is used to study the classification problem.

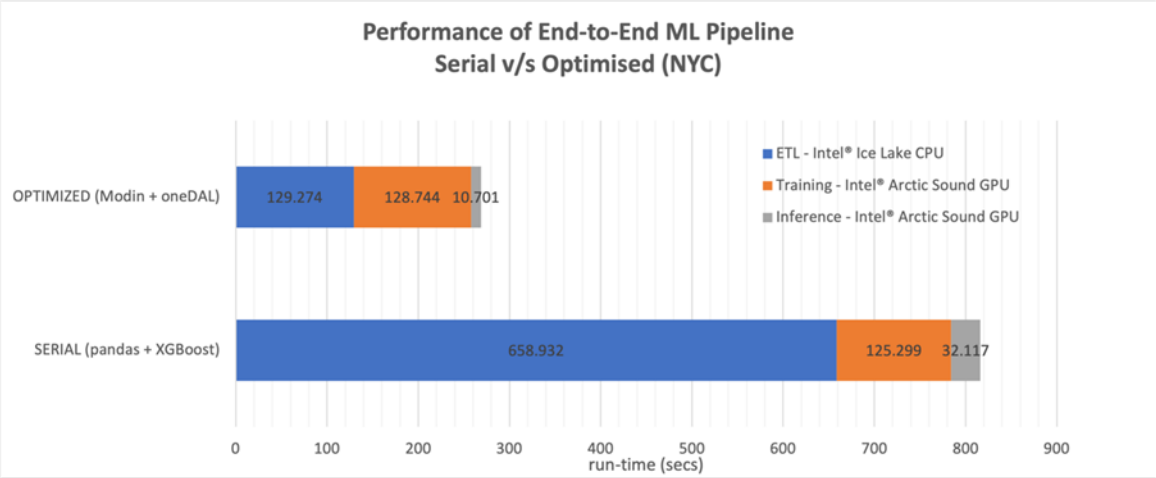

Figure 5. Performance of end-to-end ML Pipeline using the NYC Taxi dataset

The comparison results for the NYC dataset are shown in Figure 5. The overall speedup of the optimized pipeline is ~3x, with the following breakdown:

- The ETL stage shows a speedup of ~5x using Intel® oneAPI AI Analytics toolkit's distribution of Modin with Dask due to efficient use of CPU cores.
- For the Inference stage, a speedup of ~3x is seen as it uses oneDAL with XGBoost [10] to utilize all the capabilities of the Intel hardware by using Intel® Advanced Vector Extensions 512 (AVX-512) [11] vector instruction set to maximize the utilization of the Intel® Xeon® processors.
- In the Training stage, XGBoost model training is converted to oneDAL using daal4py (overhead), hence taking slightly more time than the native pandas and XGBoost. However, because the data size for the NYC dataset is large (64GB), the overhead is proportionally negligible by comparison. But still, there is a slight slowdown in this stage.

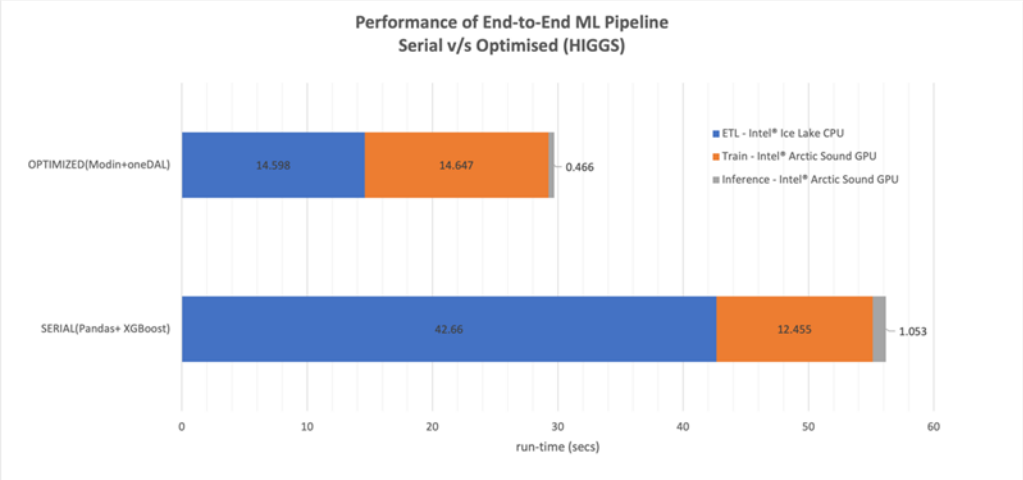

Figure 6. Performance of end-to-end ML Pipeline using the Higgs dataset

Shown in Figure 6 is the performance of the ML pipeline using the Higgs dataset. Again, the optimized implementation (using Modin-Dask and oneDAL) is compared against the native panda and XGBoost libraries, and the performance is also broken down by the

three stages in the ML pipeline. Note that only 20 computing cores were used for processing the Higgs dataset. Because the Higgs dataset is relatively small (7.5 GB vs 64GB for the NYC dataset), it was not large enough to be effectively partitioned beyond 20 cores. The results show a speedup of ~3x for the ETL stage and a speedup of ~2.25x for the Inference stage. Again, the XGBoost model training is converted to oneDAL using daal4py, and because the Higgs dataset size is smaller than the NYC dataset, the overhead becomes more of a factor, that is, the Training stage for the optimized version is taking more time than native pandas and XGBoost in the Training stage. As a result, the overall speedup of the optimized pipeline is ~1.9x.

Evaluation of ML Pipeline with Varying Dataset Sizes

In the section Optimized End-to-End ML Pipeline vs. Serial Pipeline, the performance of the optimized implementation (using Modin-Dask and oneDAL) was compared against the unoptimized version (using native pandas and XGBoost libraries), with a performance breakdown by the three stages in the pipeline. In this section, the optimized code is used to study the pipeline performance based on the varying dataset size of both datasets. In the section ML Pipeline Performance: NYC Dataset, Varying Dataset Sizes, the optimized code is used to study the pipeline performance based on the varying dataset size of the NYC dataset. The performance comparison is broken down for each of the three stages of the ML pipeline. The composite performance (all three pipeline stages) of the E2E ML pipeline for the NYC dataset is then summarized. Similarly, in the section ML Pipeline Performance: Higgs Dataset, Varying Dataset Sizes, the optimized code is used to study the pipeline performance based on the varying dataset size of the Higgs dataset.

ML Pipeline Performance: NYC Dataset, Varying Dataset Sizes

In this section, the optimized code is used to study the pipeline performance based on the varying dataset size of the NYC dataset. The performance comparison is broken down for each of the three stages of the ML pipeline. Figure 7, Figure 8, and Figure 9 show the performance comparison of the ETL, Training, and Inference stages, respectively, for varying the dataset size of the NYC dataset. Figure 10 presents the composite performance (all three pipeline stages) of the E2E ML pipeline for the NYC dataset, summarizing the data from Figure 7, Figure 8, and Figure 9. The dataset is varied from 2 GB to 64 GB (increment of power of 2).

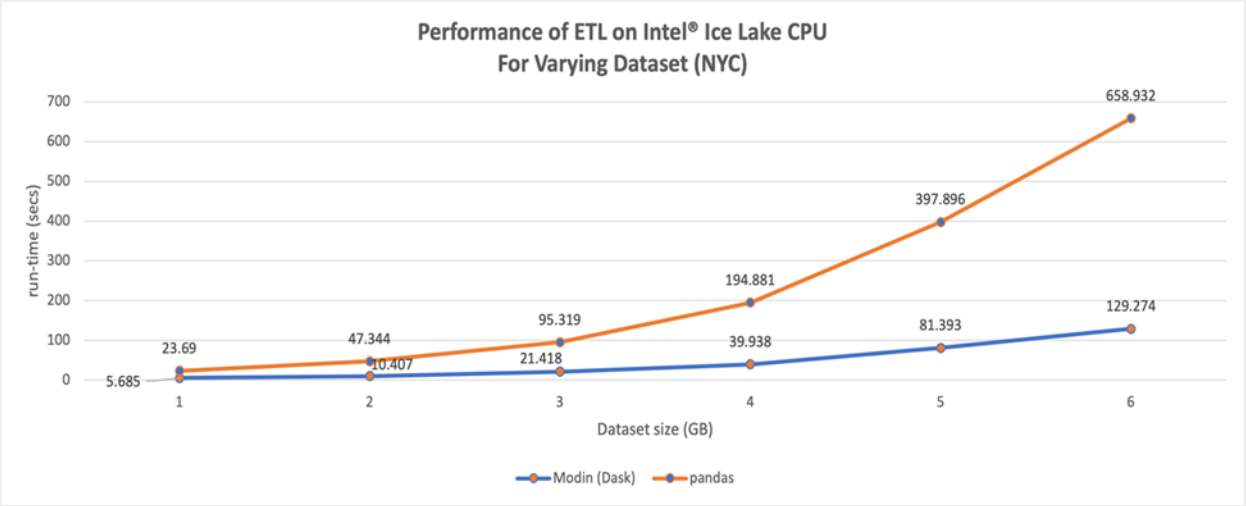

Figure 7. Performance of ML Pipeline ETL, varying NYC dataset size

Figure 7 shows a slight increase in speedup in the ETL stage of the ML pipeline as the size of the NYC dataset increases, ranging from ~4x at 2GB to ~4.8x at 16GB to ~5x at 64GB. As stated previously, the performance increase is due to Intel® AI Analytics toolkit's distribution of Modin with Dask which uses all the CPU cores efficiently.

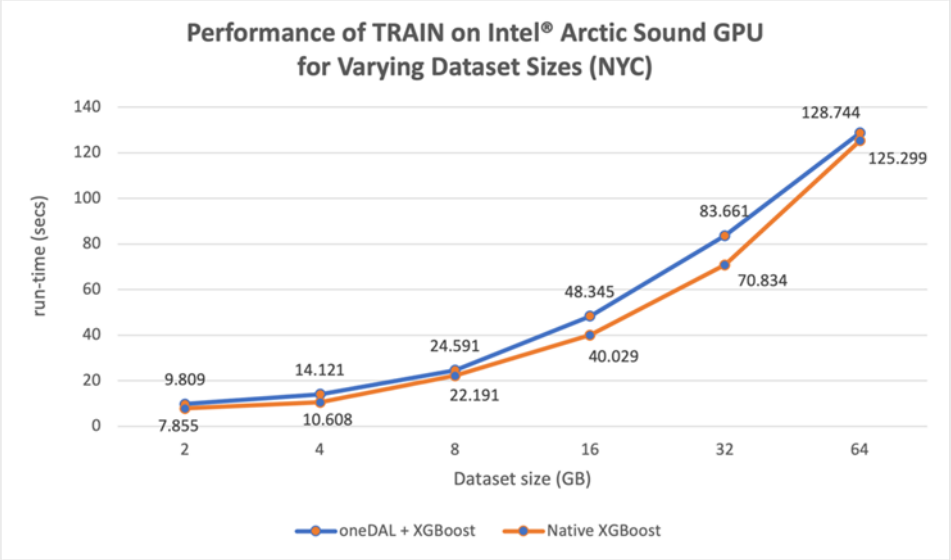

Figure 8. Performance of ML Pipeline Training, varying the NYC dataset size

Figure 8 shows the performance of the Training stage of the ML pipeline. As stated in Optimized End-to-End ML Pipeline vs. Serial Pipeline, in the Training stage, XGBoost model training is converted to oneDAL using daal4py (overhead). Hence, as confirmed in this figure, training in the optimized version takes slightly more time than the native pandas and XGBoost throughout the sizes of the NYC dataset. However, because the data size for the NYC dataset is large (64GB), the overhead is proportionally negligible by comparison.

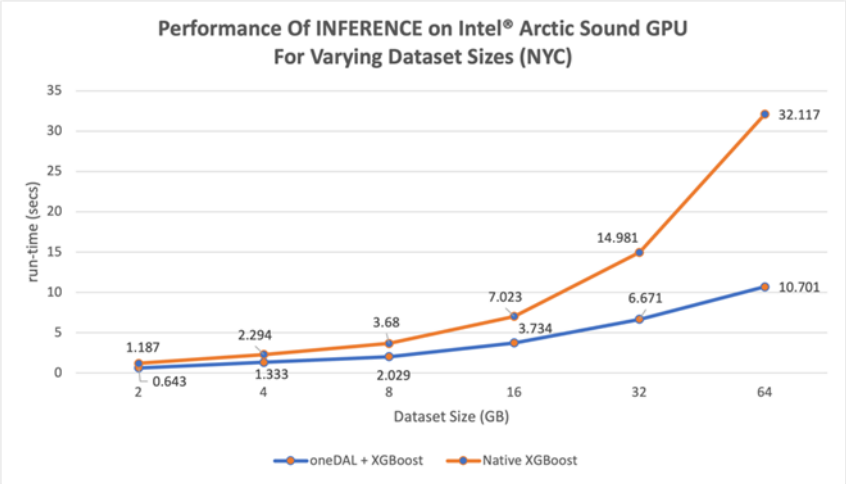

Figure 9. Performance of ML Pipeline Inference, varying NYC dataset size

For the Inference stage, shown in Figure 9, there is an increase in speedup as the dataset size increases, from ~1.8x at 2GB to the largest speedup of ~3x at 64GB. As stated previously, the performance boost is due to the Intel® Advanced Vector Extensions 512 (Intel® AVX-512) vector instruction set which maximizes the utilization of the Intel® Xeon® processors.

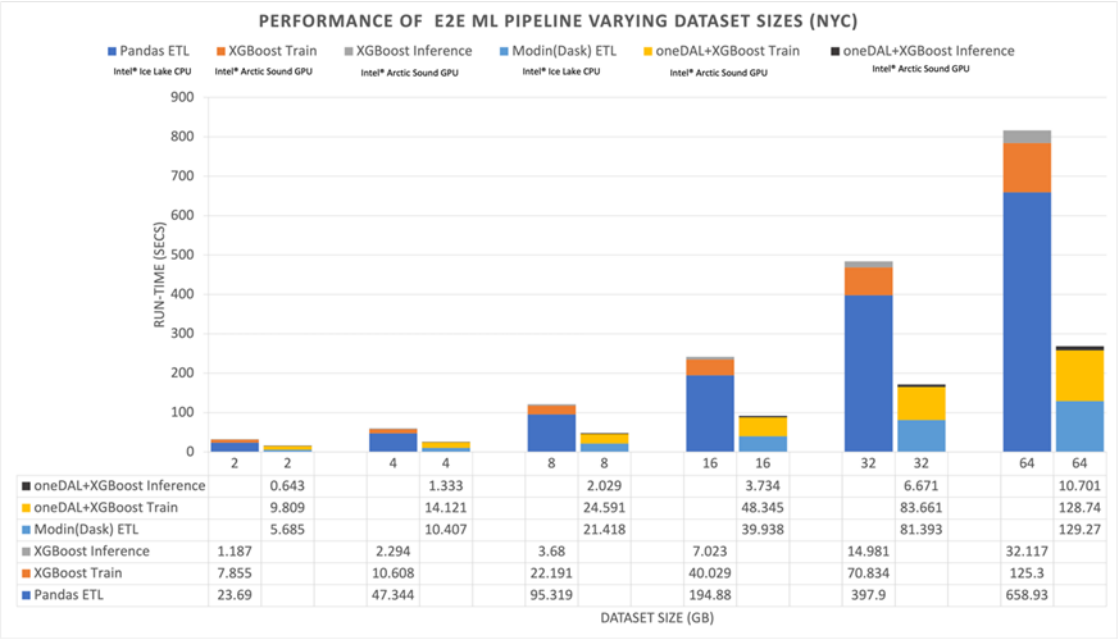

Figure 10. Composite E2E ML pipeline performance, varying NYC dataset size

Figure 10 shows the composite performance (all three pipeline stages) of the E2E ML pipeline for the NYC dataset, summarizing the data from Figure 7, Figure 8, and Figure 9. We can see (confirmed) that the ETL stage is the most time-consuming stage in the pipeline, for all dataset sizes. The Training stage cannot be accelerated at all by the AI Analytics Toolkit. In fact, at small dataset sizes, the overhead of the Training stage is especially noticeable. The overall speedup of the E2E pipeline increases steadily from ~2x at 2GB to ~2.6x at 16GB to the maximum speedup of ~3x for 64G.

ML Pipeline Performance: Higgs Dataset, Varying Dataset Sizes

In this section, the optimized code is used to study the pipeline performance based on the varying size of the Higgs dataset. The performance comparison is broken down for each of the three stages of the ML pipeline. Figure 11, Figure 12, and Figure 13 show the performance comparison of the ETL, Training, and Inference stages, respectively, for varying the size of the Higgs dataset. Figure 14 presents the composite performance (all three pipeline stages) of the E2E ML pipeline for the Higgs dataset, summarizing the data from Figure 11, Figure 12, and Figure 13.

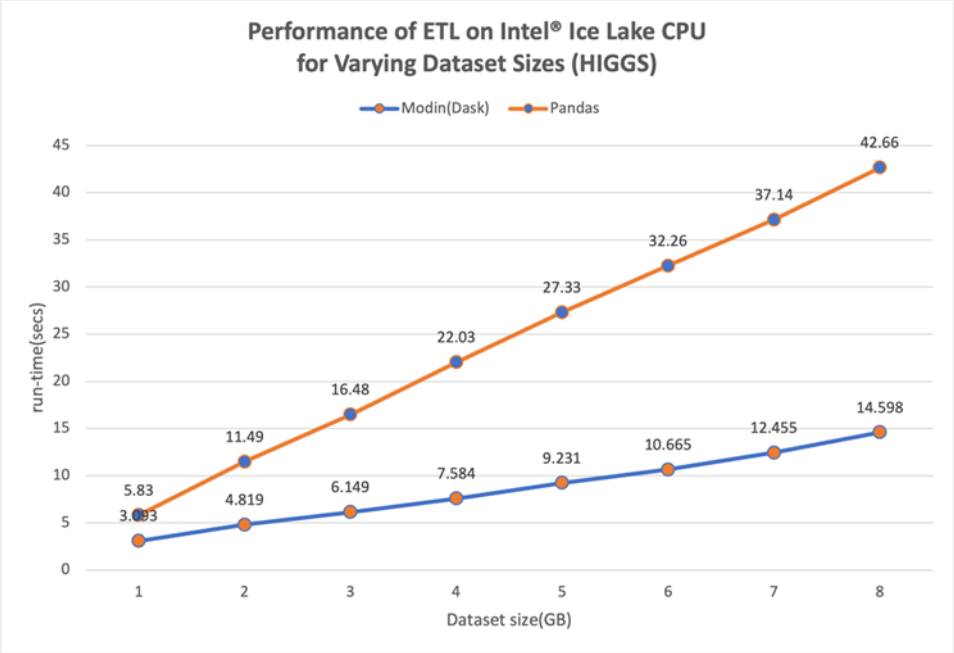

Figure 11. Performance of ML Pipeline ETL, varying Higgs dataset size

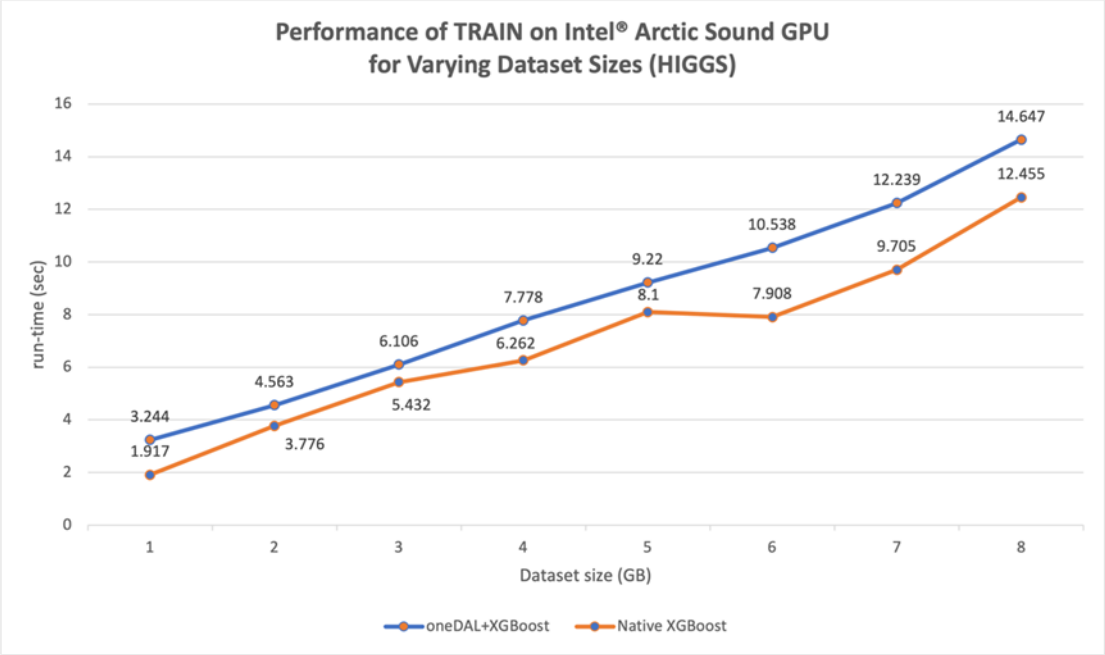

Figure 12. Performance of ML Pipeline Training, varying Higgs dataset size

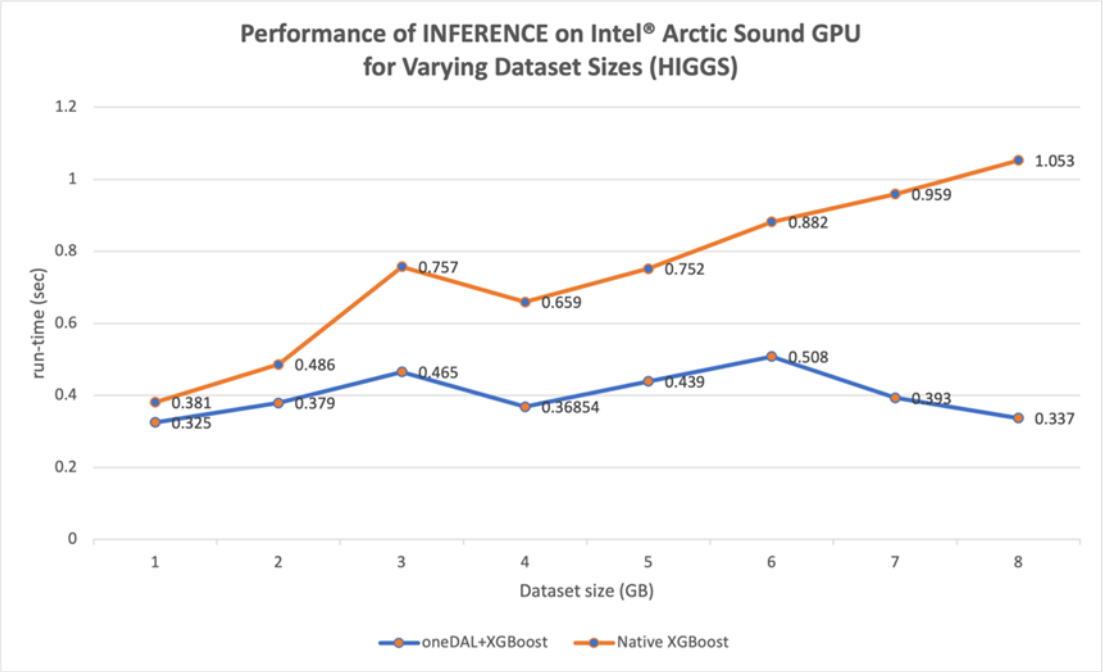

Figure 13. Performance of ML Pipeline Inference, varying Higgs dataset size

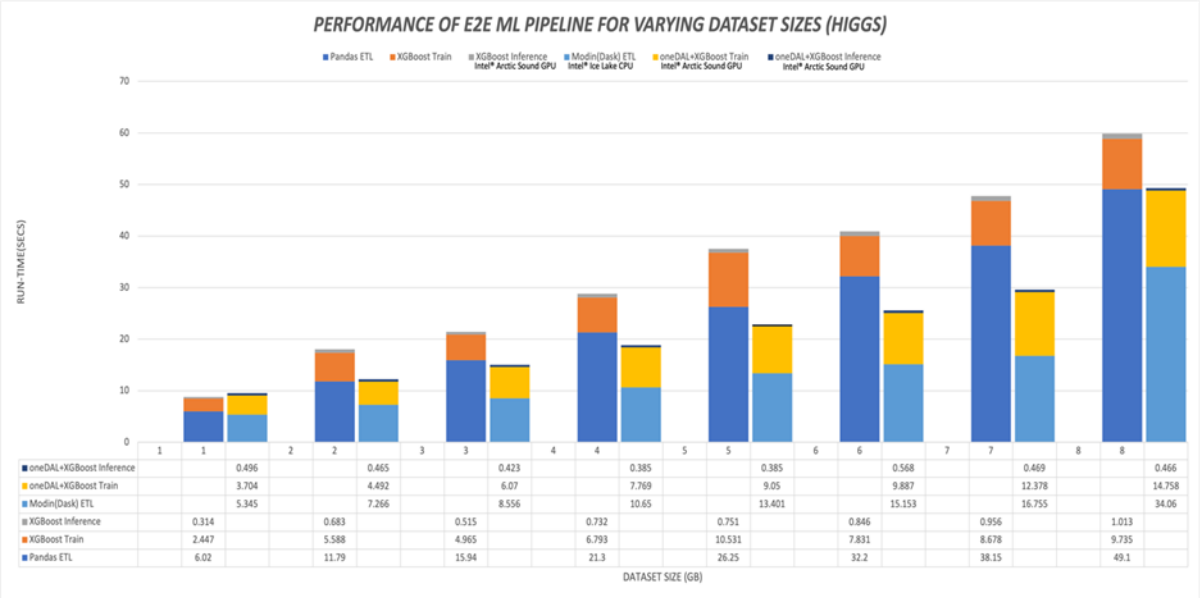

Figure 14. Composite E2E ML pipeline performance, varying Higgs dataset size

The Higgs dataset was varied from 1 GB to 7.5 GB (increment of 1 GB). The following is a summary of the observations:

- For the ETL stage (Figure 11), there is a steady increase in speedup, ranging from ~1.9x at 1GB to ~2.9x at 4GB to ~3x at 7.5GB. The performance boost is again because of Intel® AI Analytics toolkit's distribution of Modin with Dask which uses all the CPU cores efficiently.
- For the Training stage (Figure 12), the baseline pandas performed better (confirming this observation made in Figure 6). As stated previously, the XGBoost

model training is converted to oneDAL using daal4py, which added the overhead. Because the Higgs dataset size is smaller than the NYC dataset, the overhead becomes more of a factor.

- Finally, for the Inference stage (Figure 13), there is an increase in speedup as the dataset size increases (similar to the NYC dataset), from slightly >1x at 1GB to ~1.8x at 4GB to the largest speedup of >3x at 7.5GB. The performance boost is again due to the Intel® Advanced Vector Extensions 512 (Intel® AVX-512) vector instruction set which maximizes the utilization of the Intel® Xeon® processors.
- Figure 14 shows the composite performance (all three pipeline stages) of the E2E ML pipeline for the Higgs dataset, summarizing the data from Figure 11, Figure 12, and Figure 13. Similar to the observation of the NYC dataset, we can see that the ETL stage is the most time-consuming stage in the pipeline, for all dataset sizes and the Training cannot be accelerated at all by the AI Analytics Toolkit. At very small dataset sizes (1GB, 2GB), the overall E2E pipeline speedup is very modest. Beyond 2GB, the speedup stayed fairly constant at <2x, with a speedup of ~1.9x at 7.5GB. In summary, the Higgs dataset is too small to take full advantage of the parallel benefits offered by the AI Analytics Toolkit.

Conclusions

An end-to-end (E2E) Machine Learning (ML) pipeline consists of three general stages:

- Extraction, Transfer, and Loading (ETL) of data
- Model Training
- Inference and Visualization

These stages are generally computationally expensive, which impacts the performance and effectiveness of the resulting ML pipeline. The goal of this project is to accelerate the performance of an E2E ML pipeline using Intel's oneAPI AI Analytics Toolkit and compare its performance against a baseline pandas implementation.

In evaluating the ETL stage (the most time-consuming stage in the pipeline), we showed that all three Modin backends (Ray, Dask, and Omnisci) performed better than the baseline serial pandas library, with Modin-Dask the best performing (~5x speedup). This experiment verified that the Modin libraries can efficiently make use of all available CPU cores to process data in parallel, thus allowing Modin to support the pandas API efficiently. Modin Dask also performed the best since it supports more of the pandas API as compared to the Modin Ray and OmniSci, thus minimizing the need for operations to be defaulted to its original pandas implementations. Modin's fine-grained control over partitioning allows it to support various pandas operations that are otherwise very challenging to parallelize.

Based on this result, an optimized E2E ML pipeline was implemented and deployed on a computing node of the Intel® DevCloud (consisting of an Intel® Ice Lake CPU and a 150W Intel® Arctic Sound GPU), to test the acceleration of large-scale workloads using Intel's OneAPI toolkit. The comparison was performed using a NYC Taxi historical dataset (64GB) for ML regression analysis; and a Higgs dataset (7.5GB) for the classification problem. The optimized E2E ML pipeline was compared against an unoptimized version

(using native pandas and XGBoost libraries). Experiments were also performed for varying dataset sizes.

In summary, Intel's AI Analytics toolkit provides familiar Python tools and frameworks built with low-level optimizations to conveniently accelerate end-to-end ML algorithms. With oneAPI libraries built for low-level compute optimizations, it is possible to achieve highly efficient multithreading, vectorization, and memory management, and scale scientific computations across a cluster.

The optimized implementation uses Modin Dask to accelerate the ETL stage of the E2E ML pipeline. For the ETL stage, there is a steady speedup increase for each of the datasets as the size of the dataset increases: the NYC dataset ranges from ~4x at 2GB to ~4.8x at 16GB to ~5x at 64GB; the Higgs dataset ranges from ~1.9x at 1GB to ~2.9x at 4GB to ~3x at 7.5GB. The performance increase of the ETL stage can be attributed to Intel® AI Analytics toolkit's distribution of Modin with Dask which uses all the CPU cores efficiently.

The optimized implementation uses oneDAL to accelerate the Inference stage of the E2E ML pipeline. For the NYC dataset, there is an increase in speedup of the Inference stage as the dataset size increases, from ~1.8x at 2GB to the largest speedup of ~3x at 64GB. For the Higgs dataset, the increase in speedup is from slightly >1x at 1GB to ~1.8x at 4GB to the largest speedup of >3x at 7.5GB. The performance boost for the Inference stage is the result of the Intel® Advanced Vector Extensions 512 (Intel® AVX-512) vector instruction set which maximizes the utilization of the Intel® Xeon® processors.

For the Training stage, the baseline pandas performed better for both datasets. This is expected because the XGBoost model training is converted to oneDAL using daal4py, which added some overhead. For the NYC dataset, training in the optimized version takes slightly more time than the native pandas for all sizes of the NYC dataset. For the smaller Higgs dataset, the overhead is even more of a factor.

Finally, as shown in the section Evaluation of ML Pipeline with Varying Dataset Sizes, the composite performance (all three pipeline stages) of the E2E ML pipeline summarizes the results. The ETL stage is the most time-consuming stage in the pipeline, for all dataset sizes. Both the ETL and Inference stages showed a steady increase in speedup, but the Training stage cannot be accelerated at all by the AI Analytics Toolkit. In fact, at small dataset sizes, the overhead of the Training stage is especially noticeable. For the NYC dataset, the overall speedup of the E2E pipeline increases steadily from ~2x at 2GB to ~2.6x at 16GB to the maximum speedup of ~3x for 64G. For the Higgs dataset, the overall speedup is very modest at very small dataset sizes (1GB, 2GB). Beyond 2GB, the speedup stayed fairly constant at <2x, with a speedup of ~1.9x at 7.5GB. Thus, small datasets, such as the Higgs dataset, are too small to take full advantage of the parallel benefits offered by the AI Analytics Toolkit

Appendix: Reproducibility

Software
• Python 3.7 • pandas 1.2.3 • XGBoost 1.4.2 • Intel oneAPI AI Analytics Toolkit • Intel Modin 0.9.1 • Intel daal4py 2021.4.0
Data
Dataset 1: NYC Taxi • Size: 72 GB • Link: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page Dataset 2: Higgs • Size: 7.5 GB • Link: https://archive.ics.uci.edu/ml/datasets/HIGGS
Code Repository
Models: • NYC Taxi: XGBoost regression • Higgs: XGBoost classification E2EMLpipeline code repository: • (Link to be determined)
Hardware
Platform: Intel DevCloud • CPU model name: Intel® Xeon® Ice Lake-SP • CPU cores: 96 • CPU Sockets: 2 GPU: ATS-M • GPU Model: 150W

References

1. https://www.intel.com/content/www/us/en/developer/tools/oneapi/ai-analytics-toolkit.html#gs.djbqeh
2. https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html#gs.dkgl45
3. https://www.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/api-based-programming/intel-oneapi-data-analytics-library-onedal.html
4. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page
5. https://archive.ics.uci.edu/ml/datasets/HIGGS
6. https://www.intel.com/content/www/us/en/developer/tools/oneapi/distribution-of-modin.html#gs.djbww7
7. https://devcloud.intel.com/oneapi/home/
8. https://towardsdatascience.com/modin-accelerating-your-pandas-functions-by-changing-one-line-of-code-504c39b5ddbc
9. https://www.intel.com/content/www/us/en/developer/articles/technical/improve-performance-xgboost-lightgbm-inference.html#gs.dkgrcf
10. https://medium.com/intel-analytics-software/fast-gradient-boosting-tree-inference-for-intel-xeon-processors-35756f174f55
11. https://www.intel.com/content/www/us/en/architecture-and-technology/avx-512-overview.html