Choose the Best Accelerated Technology

Accelerate Classical Machine Learning on Intel® Architecture

Vladimir Kilyazov

AI Software Solutions Engineer

Notices and Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

Workloads and Configurations

See all benchmarks and configurations: https://software.intel.com/content/www/us/en/develop/articles/blazing-fast-python-data-science-ai-performance.html. Each performance claim and configuration data is available in the body of the article listed under sections 1, 2, 3, 4, and 5. Please also visit this page for more details on all scores, and measurements derived.

Testing Date: Performance results are based on testing by Intel as of October 16, 2020 and may not reflect all publicly available updates. **Configurations details and Workload Setup:** 2 x Intel® Xeon® Platinum 8280 @ 28 cores, OS: Ubuntu 19.10.5.3.0-64-generic Mitigated 384GB RAM (192 GB RAM (12x 32GB 2933). SW: Modin 0.81. Scikit-learn 0.22.2. Pandas 1.01, Python 3.8.5, DAL(DAAL4Py) 2020.2, Census Data, (21721922.45) Dataset is from IPUMS USA, University of Minnesota, www.ipums.org [Steven Ruggles, Sarah Flood, Ronald Goeken, Josiah Grover, Erin Meyer, Jose Pacas and Matthew Sobek. IPUMS USA: Version 10.0 [dataset], Minneapolis, MN. IPUMS, 2020. https//doc.org/10.18128/D010.V10.0]

Testing Date: Performance results are based on testing by Intel® as of October 23, 2020 and may not reflect all publicly available updates. **Configuration Details and Workload Setup:** Intel® oneAPI Data Analytics Library 2021.1 (oneDAL). Scikit-learn 0.23.1, Intel® Distribution for Python 3.8; Intel® Xeon® Platinum 8280LCPU @ 270GHz, 2 sockets, 28 cores per socket, 10M samples, 10 features, 100 clusters, 100 iterations, float32.

Testing Date: Performance results are based on testing by Intel® as of October 23, 2020 and may not reflect all publicly available updates. **Configuration Details and Workload Setup:** Intel® oneAPI AI Analytics Toolkit v2021.1; Intel® oneAPI Data Analytics Library (oneDAL) beta10, Scikit-learn 0.23.1, Intel® Distribution for Python 3.7, Intel® Xeon® Platinum 8280 CPU @ 2.70GHz, 2 sockets, 28 cores per socket, microcode: 0x4003003, total available memory 376 GB, 12X32GB modules, DDR4. **AMD Configuration:** AMD Rome 7742 @2.25 GHz, 2 sockets, 64 cores per socket, microcode: 0x8301038, total available memory 512 GB, 16X32GB modules, DDR4, oneDAL beta10, Scikit-learn 0.23.1, Intel® Distribution for Python 3.7. **NVIDIA Configuration:** NVIDIA Tesla V100 – 16 Gb, total available memory 376 GB, 12X32GB modules, DDR4, Intel® Xeon Platinum 8280 CPU @ 2.70GHz, 2 sockets, 28 cores per socket, microcode: 0x5003003, cuDF 0.15, cuML 0.15, CUDA 10.2.89, driver 440.33.01, Operation System: CentOS Linux 7 (Core), Linux 4.19.36 kernel.

Testing Date: Performance results are based on testing by Intel® as of October 13, 2020 and may not reflect all publicly available updates. **Configurations details and Workload Setup:** CPU: c5.18xlarge AWS Instance (2 x Intel® Xeon® Platinum 8124M @ 18 cores. OS: Ubuntu 20.04.2 LTS, 193 GB RAM. GPU: p3.2xlarge AWS Instance (GPU: NVIDIA Tesla V100 16GB, 8 vCPUs, OS: Ubuntu 18.04.2LTS, 61 GB RAM. SW: XGBoost 1.1: build from sources compiler – G++ 7.4, nvcc 9.1 Intel® DAAL: 2019.4 version: Python env: Python 3.6, Numpy 1.16.4, Pandas 0.25 Scikit-learn 0.21.2.

Workloads and Configurations

Testing Date: Performance results are based on testing by Intel® as of October 26, 2020 and may not reflect all publicly available updates. **Configuration Details and Workload Setup:** Intel® Optimization for Tensorflow v2.2.0; oneDNN v1.2.0; Intel® Low Precision Optimization Tool v1.0; Platform; Intel® Xeon® Platinum 8280 CPU; #Nodes 1; #Sockets: 2; Cores/socket: 28; Threads/socket: 56; HT: On; Turbo: On; BIOS version:SE5C620.86B.02.01.0010.010620200716; System DDR Mem Config: 12 slots/16GB/2933; OS: CentOS Linux 7.8; Kernel: 4.4.240-1.el7.elrepo x86_64.

Testing Date: Performance results are based on testing by Intel® as of February 3, 2021 and may not reflect all publicly available updates. **Configuration Details and Workload Setup:** Intel® Optimization for PyTorch v1.5.0; Intel® Extension for PyTorch (IPEX) 1.1.0; oneDNN version: v1.5; DLRM: Training batch size (FP32/BF16): 2K/instance, 1 instance; DLRM dataset (FP32/BF16): Criteo Terabyte Dataset; BERT-Large: Training batch size (FP32/BF16): 24/Instance. 1 Instance on a CPU socket. Dataset (FP32/BF16): WikiText-2 [https://www.salesforce.com/products/einstein/ai-research/the-wiktext-dependency-language-modeling-dataset/]: ResNext101-32x4d: Training batch size (FP32/BF16): 128/Instance, 1 instance on a CPU socket, Dataset (FP32/BF16): ILSVRC2012; DLRM: Inference batch size (INT8): 16/instance, 28 instances, dummy data. Intel® Xeon® Platinum 8380H Processor, 4 socket, 28 cores HT On Turbo ON Total memory 768 GB (24 slots/32GB/3200 MHz), BIOS; WLYDCRBLSYS.0015.P96.2005070242 (ucode: OX 700001b), Ubuntu 20.04 LTS, kernel 5.4.0-29-genen: ResNet50: [https://github.com/Intel/optimized-models/tree/master/pytorch/ResNet50]: ResNext101 32x4d: [https://github.com/intel/optimized-models/tree/master/pytorch/ResNext101_32x4ct: DLRM: https//github.com/intel/optimized-models/tree/master/pytorch/dlrm].

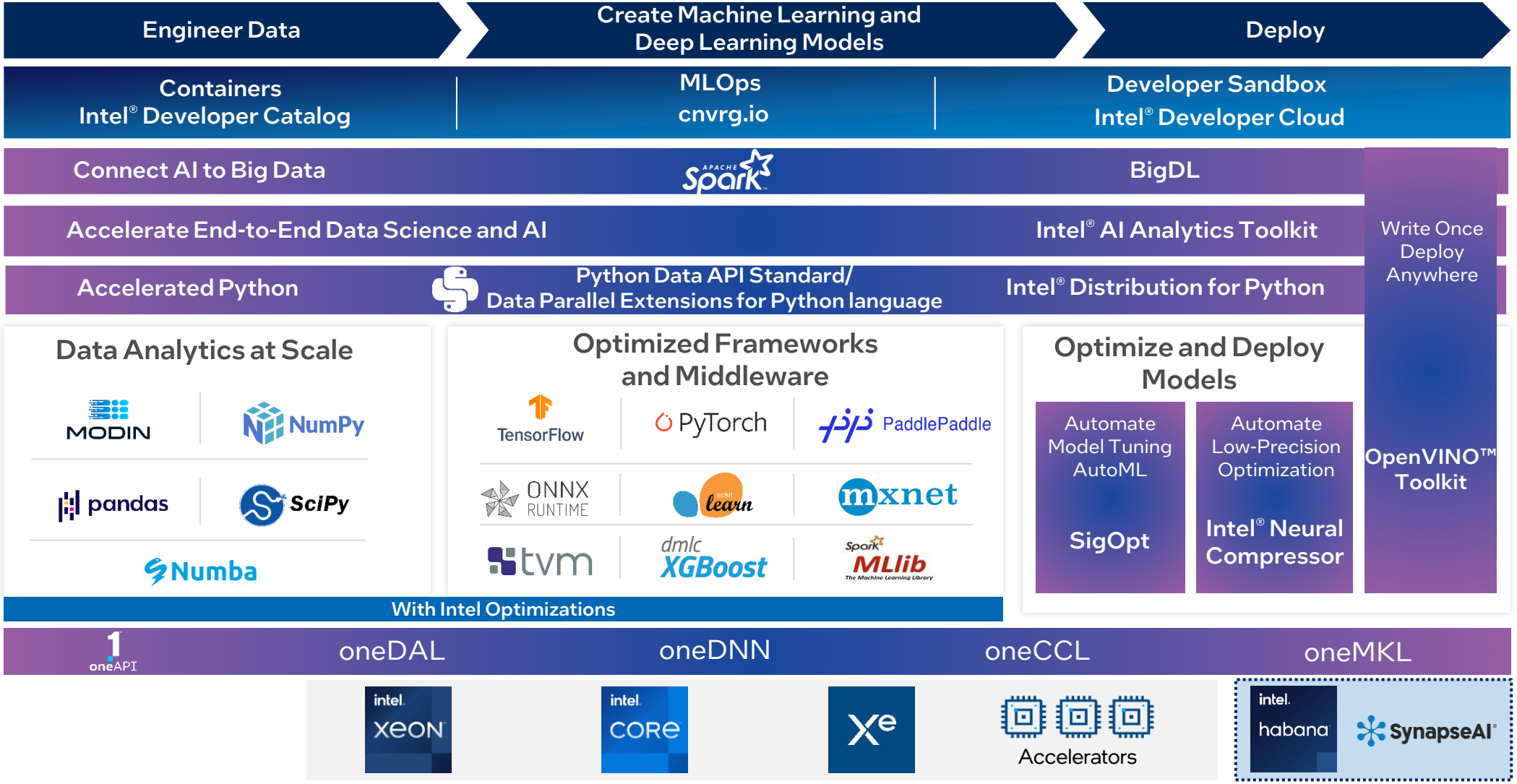
Agenda

- Intel® AI Analytics Toolkit Overview
- A Closer Look at:
  - Intel® Distribution for Python*: Intel® Optimizations for NumPy* and SciPy*
  - Intel® Distribution of Modin*
  - Intel® Extension for Scikit-learn* and XGBoost*
- Demos on Intel Developer Cloud

Intel® AI Analytics Toolkit Overview

A Brief Overview of Intel® AI Python Offerings

For larger scale and increased performance in data science workloads:



Intel® AI Analytics Toolkit

Powered by oneAPI

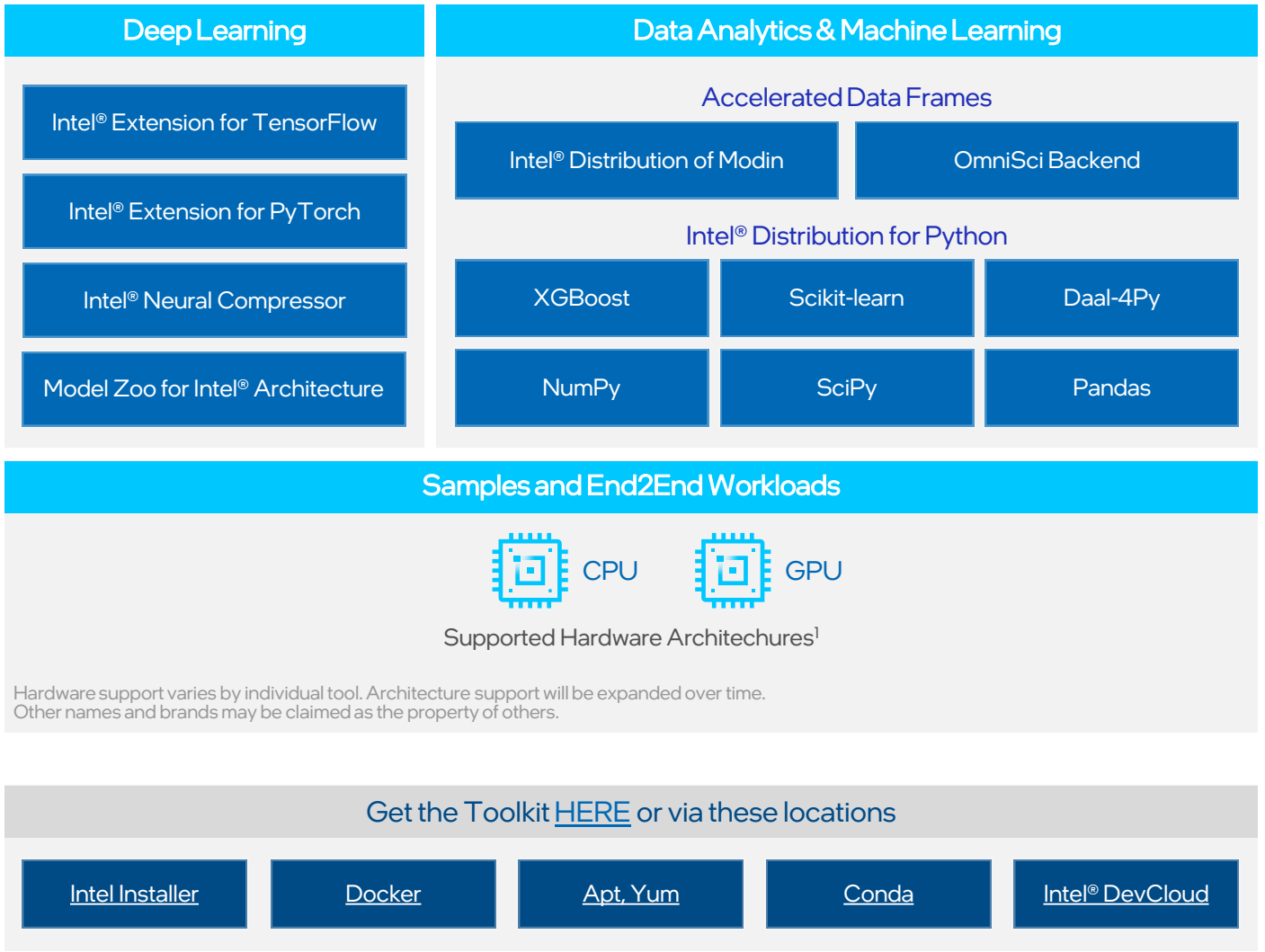
Accelerate end-to-end AI and data analytics pipelines with libraries optimized for Intel® architectures

Who Uses It?

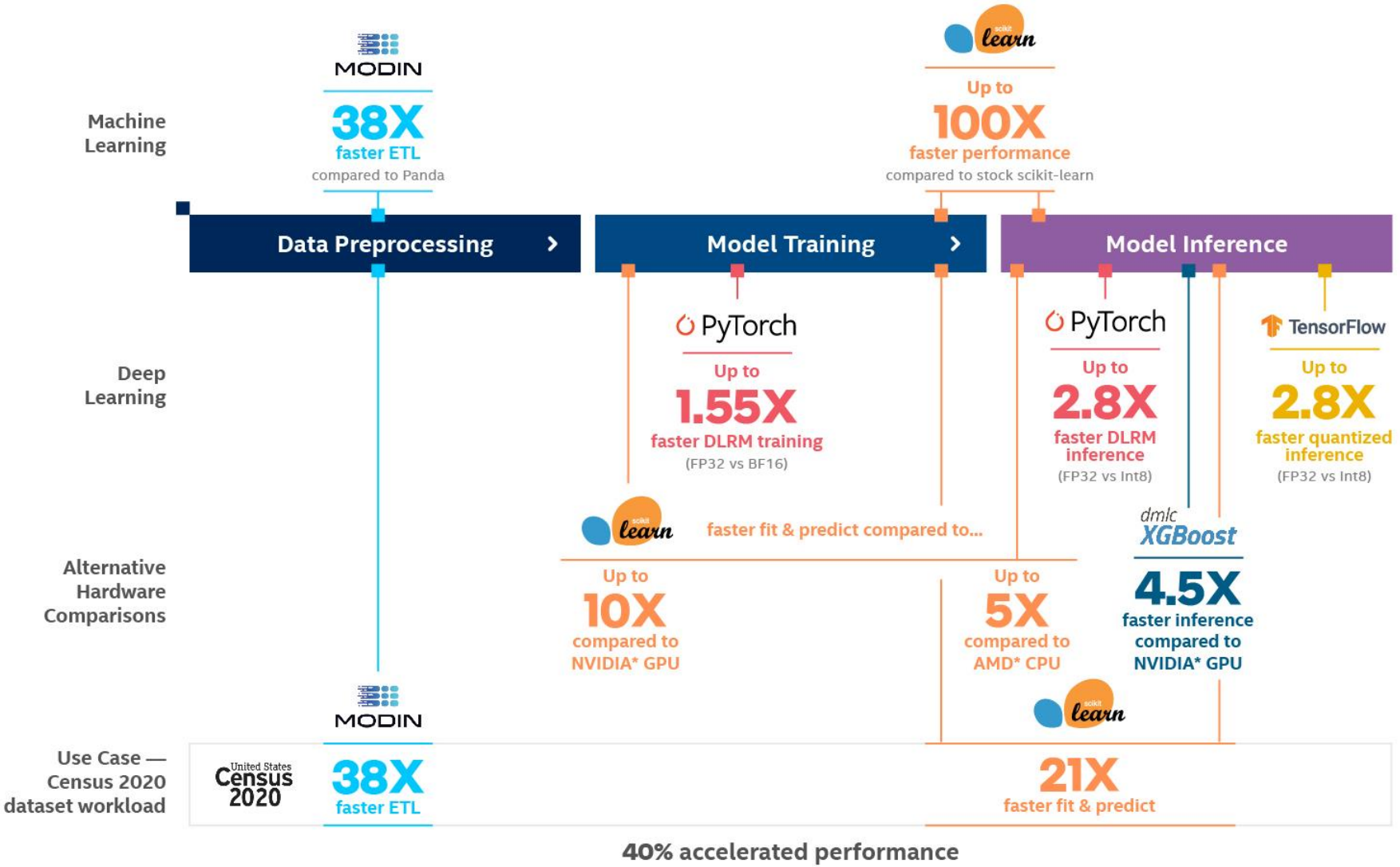
Data scientists, AI researchers, ML and DL developers, AI application developers

Top Features/Benefits

- Deep learning performance for training and inference with Intel optimized DL frameworks and tools
- Drop-in acceleration for data analytics and machine learning workflows with compute-intensive Python packages



A Sneak Peek at the Performance Benefits



*Performance improvements shown here are based off hardware running on Intel Cascade Lake processors. This chart will be updated once data from Ice Lake is available. See backup for workloads and configurations. Results may vary.

Intel® Distribution for Python*: Intel® Optimizations for NumPy* and SciPy*

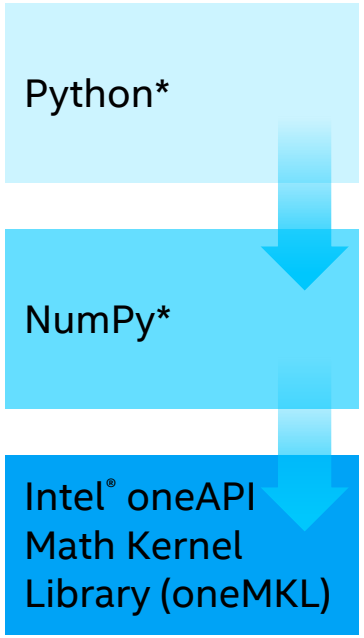
Intel® Distribution for Python

- Intel® Distribution for Python covers major usages in HPC and Data Science
- Achieve faster Python application performance — right out of the box — with minimal or no changes to a code
- Accelerate NumPy*, SciPy*, and scikit-learn* with integrated Intel® Performance Libraries such as Intel® oneMKL (Math Kernel Library) and Intel® oneDAL (Data Analytics Library)
- By default, already integrated in Anaconda

Intel® Performance Optimization with NumPy* and SciPy*

The layers of quantitative Python*

The Python* language is interpreted and has many type checks to make it flexible
Each level has various tradeoffs; NumPy* value proposition is immediately seen
For best performance, escaping the Python* layer early is best method



Enforces Global Interpreter Lock (GIL) and is single-threaded, abstraction overhead. No advanced types.

Gets around the GIL (multi-thread and multi-core)
BLAS* API can be the bottleneck

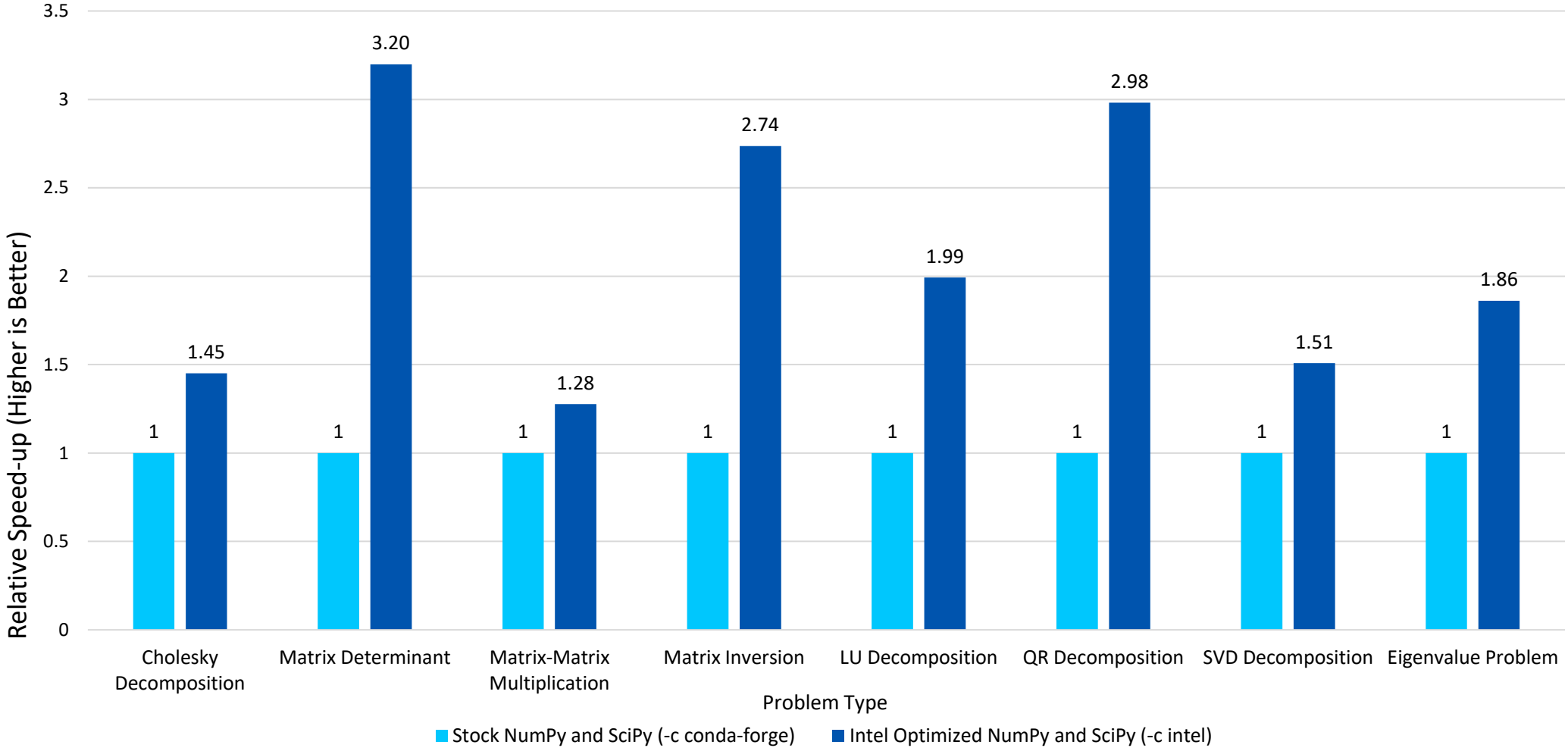
*Basic Linear Algebra Subprograms (BLAS) [CBLAS]

Gets around BLAS API bottleneck
Much stricter typing
Fastest performance level
Dispatches to hardware vectorization

Intel® oneMKL included with Anaconda standard bundle; is Free for Python

Intel Optimized NumPy* and SciPy* Linear Algebra Performance

Performance is Increased up to 3.2x with Intel Optimizations



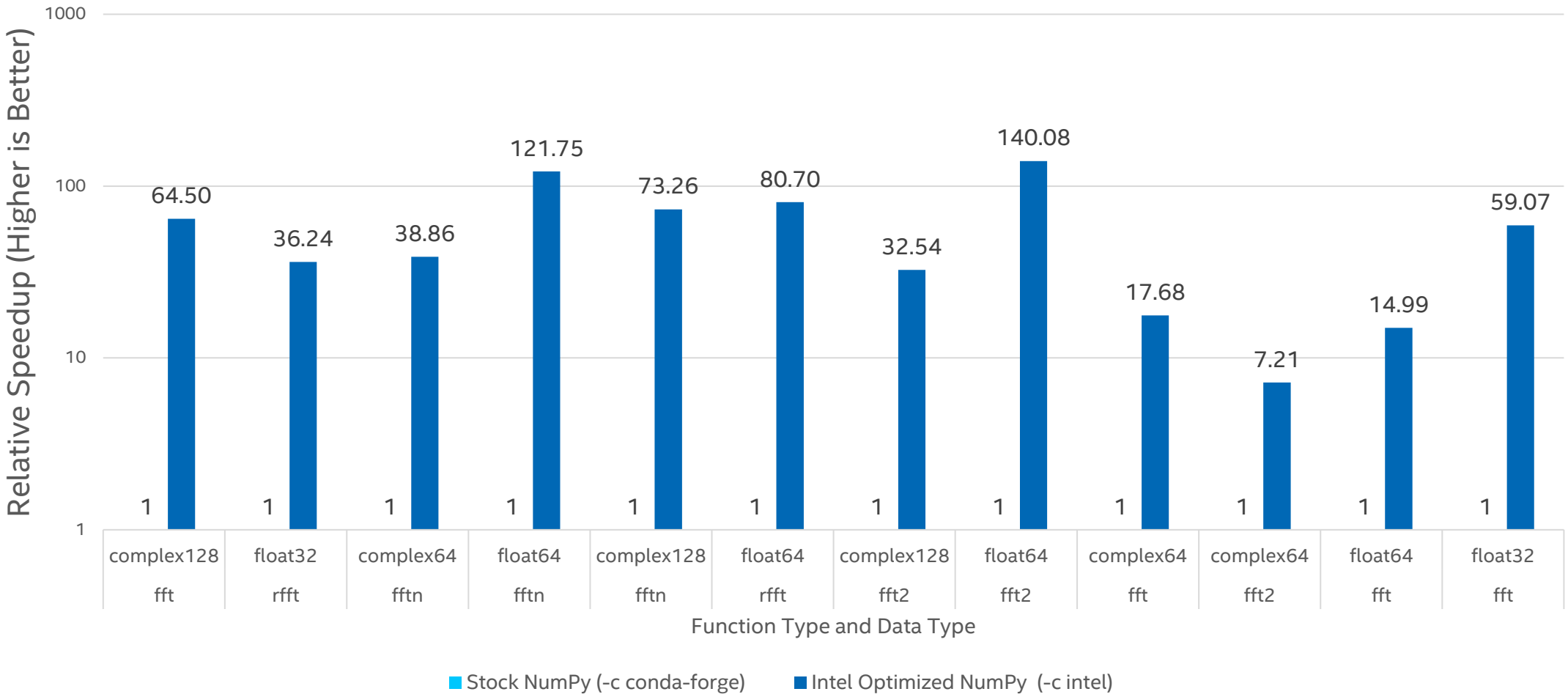
Intel Optimizations for NumPy* & SciPy* compared to conda-forge channel NumPy* & SciPy* Performance for Linear Algebra on Intel® Xeon® Platform 8480+

Testing Date: Performance results are based on testing by Intel as of July 15, 2023. **Configuration Details and Workload Setup: System:** cloud.intel.com, nodes=1:spr:ppn=2, Intel(R) Xeon(R) Platinum 8480+, 2 sockets, 56 cores per socket, HT On, Intel Turbo Boost On, Total Memory 528GB, RAM 33 MHz, Ubuntu 20.04.5 LTS, 5.18.15-051815-generic, Microcode: 0x2b000310, benchmarks https://github.com/IntelPython/ (ibench Linear Algebra), -c conda-forge environment versions: numpy 1.23.5, scipy 1.10.1, numba 0.56.4 modules installed, -c intel environment versions: numpy 1.21.4, scipy 1.7.3, numba 0.56.3, tbb4py 2021.8.0 modules installed

See backup for workloads and configurations. Results may vary.

Intel Optimized NumPy* Fast Fourier Transform Performance

Performance is Increased up to 140x with Intel Optimizations



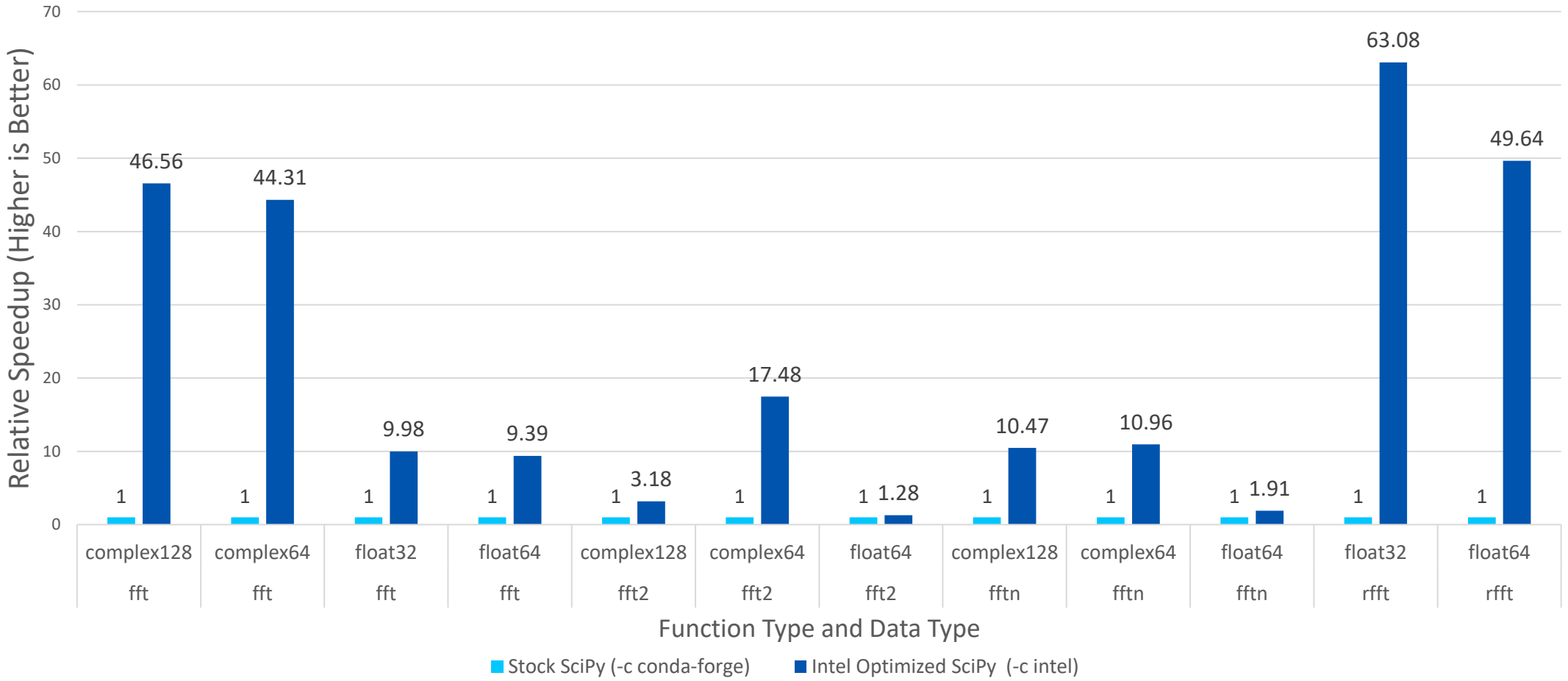
Fast Fourier Transform NumPy* performance intel vs. conda-forge on Intel® Xeon® Platform 8480+

Testing Date: Performance results are based on testing by Intel as of March 5, 2023. **Configuration Details and Workload Setup: System**: cloud.intel.com, nodes=1:spr:ppn=2, Intel(R) Xeon(R) Platinum 8480+, 2 sockets, 56 cores per socket, HT On, Intel Turbo Boost On, Total Memory 528GB, RAM 33 MHz, Ubuntu 20.04.5 LTS, 5.18.15-051815-generic, Microcode: 0x2b000310, benchmarks https://github.com/IntelPython/ (fft_benchmark, blackscholes_bench, composability_bench), -c conda-forge environment versions: numpy 1.23.5, scipy 1.10.1, numba 0.56.4 modules installed, -c intel environment versions: numpy 1.21.4, scipy 1.7.3, numba 0.56.3, tbb4py 2021.8.0 modules installed

See backup for workloads and configurations. Results may vary.

Intel Optimized SciPy* Fast Fourier Transform Performance

Out-of-place Memory Placement Performance is Increased up to 63x with Intel Optimizations



Fast Fourier Transform SciPy* performance intel vs. conda-forge on Intel® Xeon® Platform 8480+ for Out-of-place Memory Placement

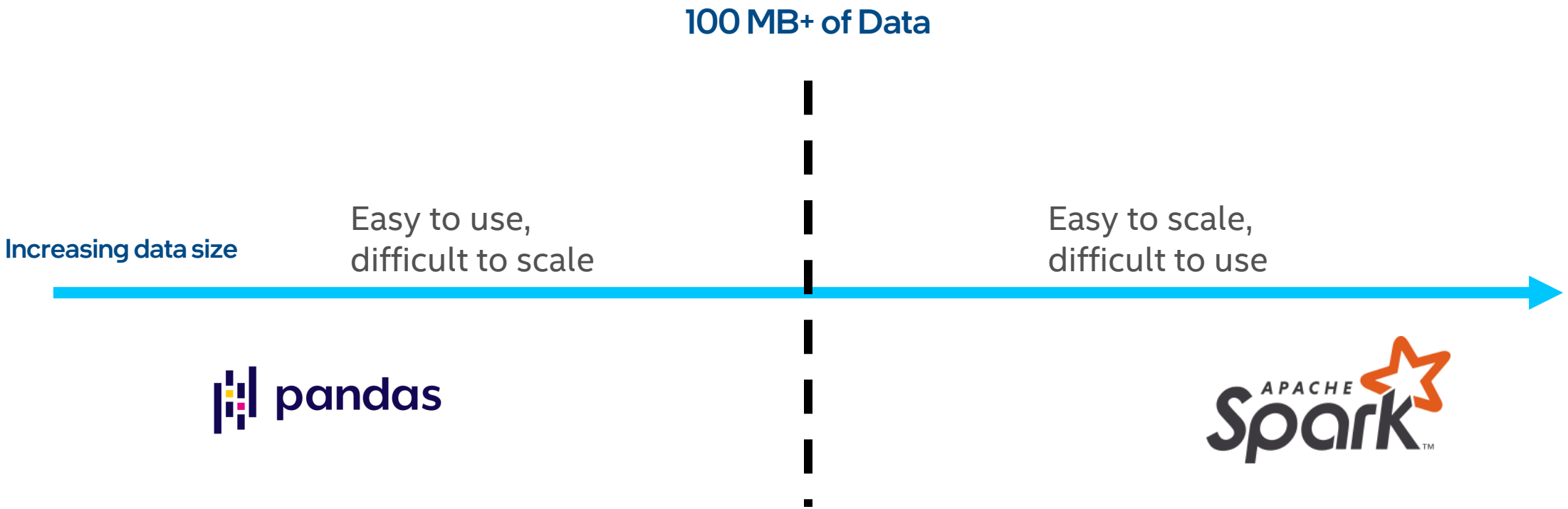
Testing Date: Performance results are based on testing by Intel as of March 5, 2023. **Configuration Details and Workload Setup: System**: cloud.intel.com, nodes=1:spr:ppn=2, Intel(R) Xeon(R) Platinum 8480+, 2 sockets, 56 cores per socket, HT On, Intel Turbo Boost On, Total Memory 528GB, RAM 33 MHz, Ubuntu 20.04.5 LTS, 5.18.15-051815-generic, Microcode: 0x2b000310, benchmarks https://github.com/IntelPython/ (fft_benchmark, blackscholes_bench, composability_bench), -c conda-forge environment versions: numpy 1.23.5, scipy 1.10.1, numba 0.56.4 modules installed, -c intel environment versions: numpy 1.21.4, scipy 1.7.3, numba 0.56.3, tbb4py 2021.8.0 modules installed

See backup for workloads and configurations. Results may vary.

Intel® Distribution of Modin*

Issue: Pandas Not Scaling to Larger Datasets

After a certain data size, need to change your API to handle more data



Solution: Modin Pandas Scales to Big Datasets

Spend the time that would be used to change the workload's API, and use it to improve your workload and analysis

Single Line Code Change for Infinite Scalability

No need to learn a new API to use Modin*

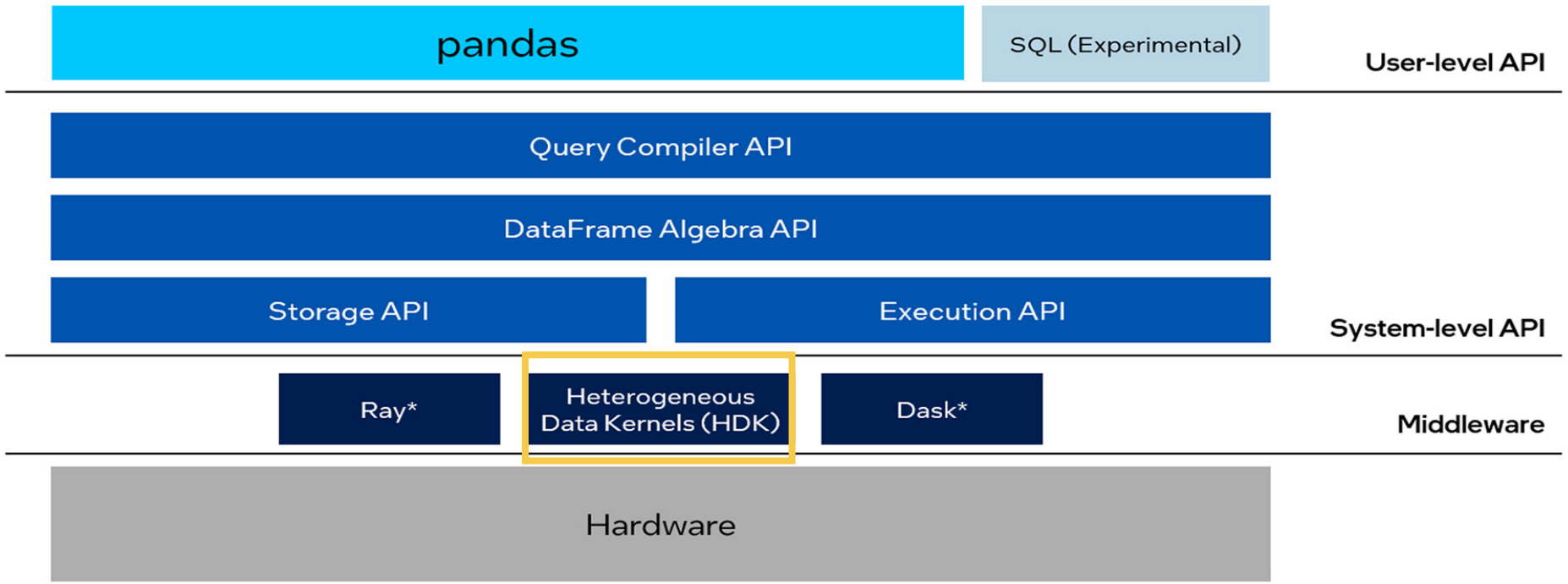
```
import pandas as pd
```

- Accelerate your Pandas* workloads across multiple cores and multiple nodes
- No upfront cost to learning a new API
  - import modin.pandas as pd
- Integration with the Python* ecosystem
- Integration with Ray/Dask clusters (run on what you have, even on a laptop!)
- Integration with Intel-built oneAPI Heterogeneous Data Kernels (oneHDK) backend
  - New experimental Modin backend based on HeavyDB* technology

Modin*: How it Works

- Modin* transparently distributes the data and computation across available cores, unlike Pandas which only uses one core at a time
- To use Modin, you do not need to know how many cores your system has, and you do not need to specify how to distribute the data

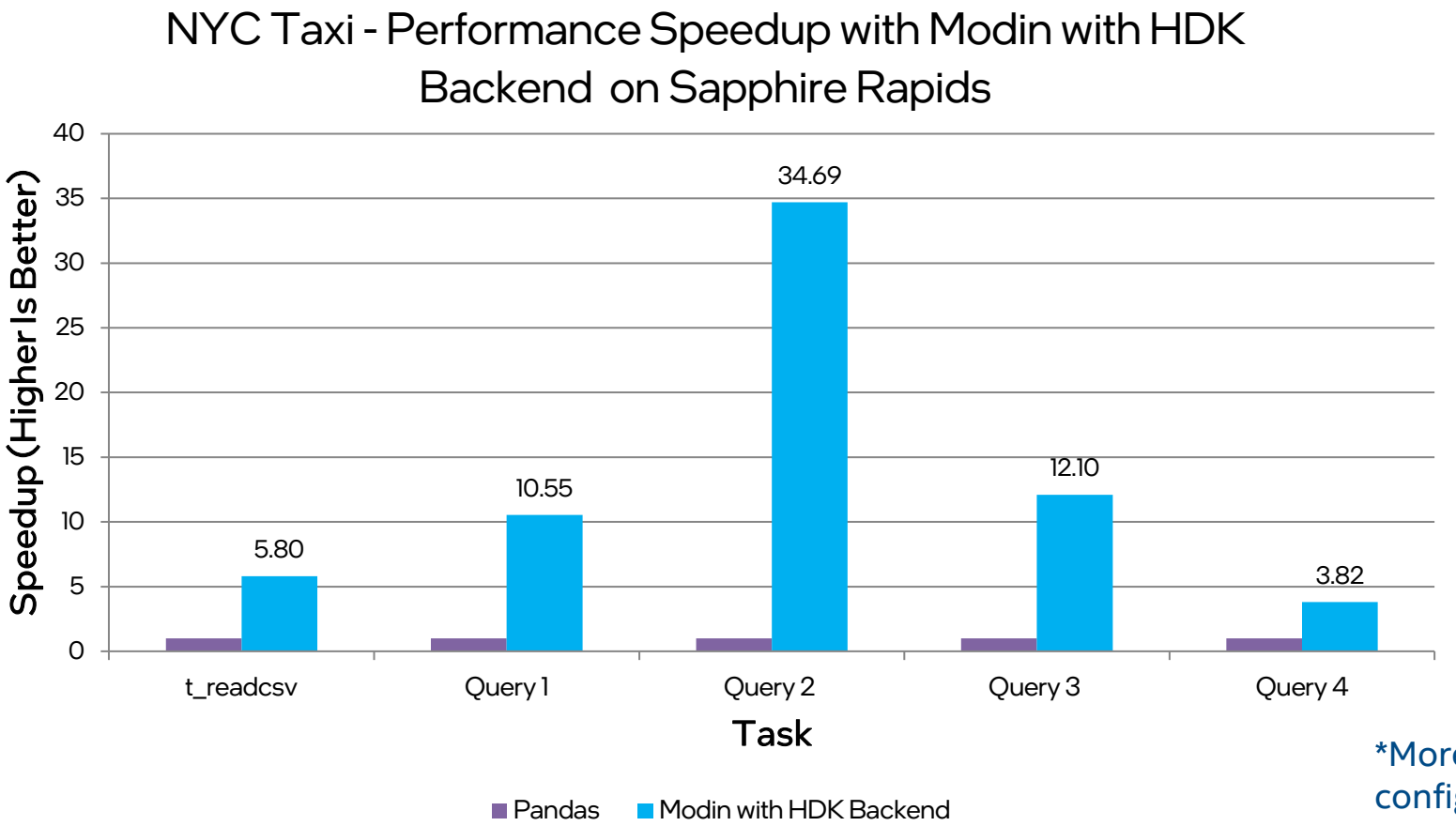
Modin – Layered API view



= High performance data analytics library developed by Intel

21

Intel® Distribution of Modin NYC Taxi Benchmark



*More detail on queries in configuration details

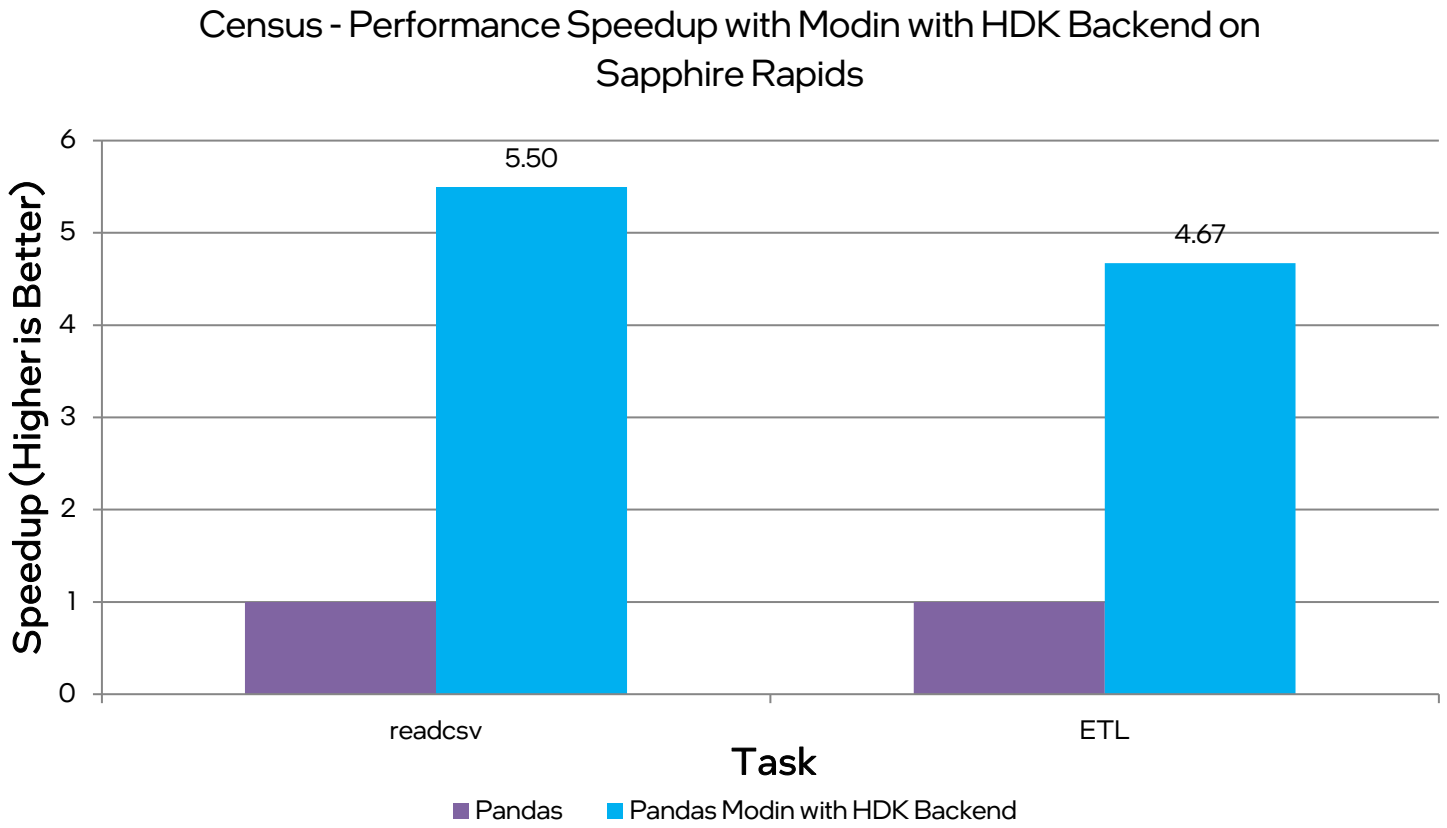
Testing Date: Performance results are based on testing by Intel as of March 03, 2023 and may not reflect all publicly available security updates

Configuration Details and Workload Setup: 2.0 GHz Intel Xeon Platinum 8468, two sockets, 48 cores per socket), 1024 GB DDR5 4800MT/s, 16x64GB DDR5 4800 MT/s, Hyperthreading enabled, Turbo mode enabled, NUMA nodes per socket=2, **BIOS:** SE5C7411.86B.9223.D04.2211291343, **Microcode:** 0x2b000111, **OS**: Ubuntu 22.04.2 LTS, **Kernel**: 5.15.0-60-generic, Python 3.9, Modin 0.18, pyhdk 0.3.1, pandas 1.5.3, Dataset [https://github.com/toddwschneider/nyc-taxi-data]. Query 1 operation utilizes select, count(), and groupby() operations on column cab_type(), Query 2 operation utilizes select, agg(), and groupby() operations on passenger_count and total_amount columns, Query 3 operation utilizes select, count, and groupby operations on passenger_count and pickup_datetime columns, Query 4 operation utilizes select, groupby(), count(), reset_index(), sort_values() operations on passenger_count, pickup_datetime, distance columns

Performance results are based on testing as of dates shown in configurations. See configuration disclosure for details. Not product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex. Your costs and results may vary

Intel® Distribution of Modin Census Benchmark on SPR



*More detail on queries in configuration details

Testing Date: Performance results are based on testing by Intel as of March 03, 2023 and may not reflect all publicly available security updates

Configuration Details and Workload Setup: 2.0 GHz Intel Xeon Platinum 8468, two sockets, 48 cores per socket), 1024 GB DDR5 4800MT/s, 16x64GB DDR5 4800 MT/s, Hyperthreading enabled, Turbo mode enabled, NUMA nodes per socket=2, **BIOS:** SE5C7411.86B.9223.D04.2211291343, **Microcode:** 0x2b000111, **OS**: Ubuntu 22.04.2 LTS, **Kernel**: 5.15.0-60-generic, Python 3.9, Modin 0.18, pyhdk 0.3.1, pandas 1.5.3, Census Data, Dataset is from IPUMS USA, University of Minnesota, www.ipums.org [Steven Ruggles, Sarah Flood, Ronald Goeken, Josiah Grover, Erin Meyer, Jose Pacas and Matthew Sobek. IPUMS USA: Version 10.0 [dataset], Minneapolis, MN. IPUMS, 2020. https//doc.org/10.18128/D010.V10.0]. ETL queries use df.shape, select, df.fillna(), df.drop() and df.astype operations

Performance results are based on testing as of dates shown in configurations. See configuration disclosure for details. Not product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex. Your costs and results may vary

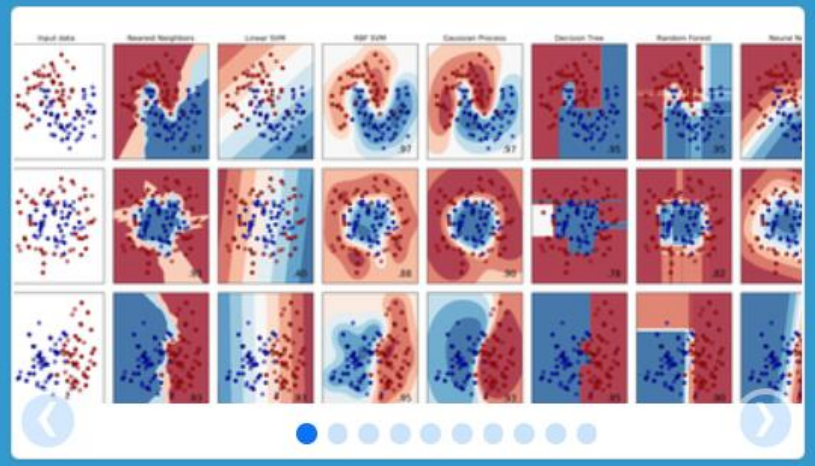
Intel® Extension for Scikit-learn* and XGBoost*

THE MOST POPULAR ML PACKAGE FOR PYTHON*

scikit learn

Home Installation Documentation Examples

Google Custom Search Search



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, … — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, … — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, … — Examples

Intel(R) Extension for Scikit-learn

Common Scikit-learn

```
from sklearn.svm import SVC
X, Y = get_dataset()

clf = SVC().fit(X, y)
res = clf.predict(X)
```

Scikit-learn with Intel CPU opts

```
from sklearnex import patch_sklearn
patch_sklearn()

from sklearn.svm import SVC

X, Y = get_dataset()

clf = SVC().fit(X, y)
res = clf.predict(X)
```

Same Code,
Same Behavior

PASSED

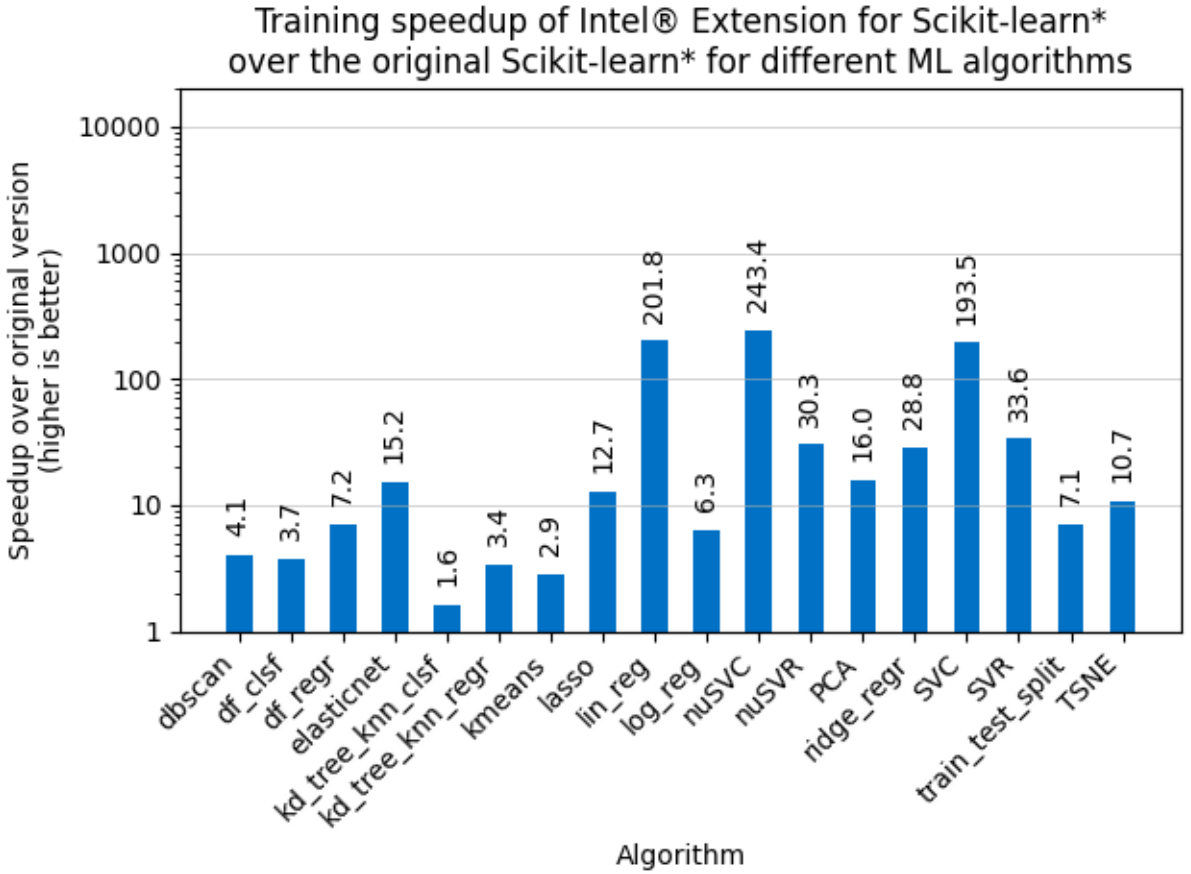
- Scikit-learn, not scikit-learn-*like*
- Scikit-learn conformance (mathematical equivalence) defined by Scikit-learn Consortium, continuously vetted by public CI

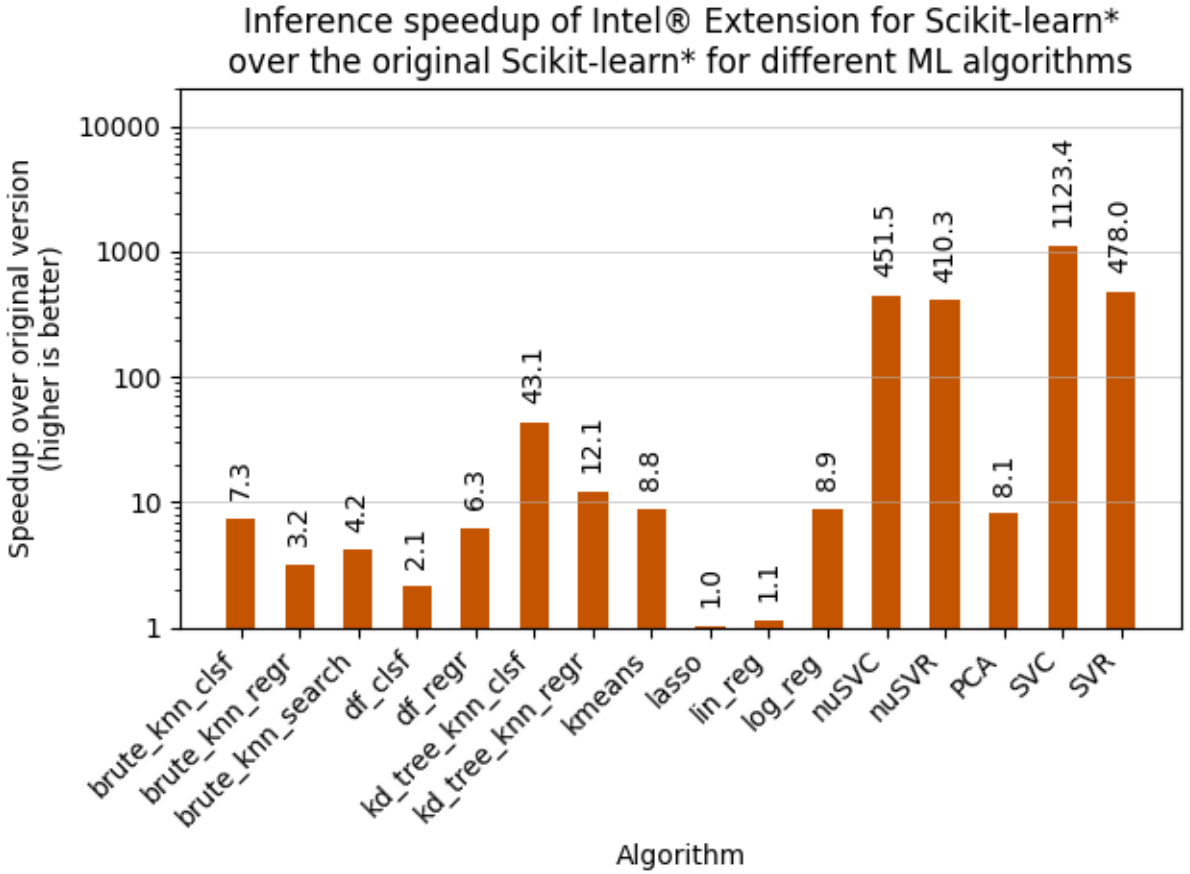
Scikit-learn mainline

Available through:

- conda install scikit-learn-intelex
- conda install –c intel scikit-learn-intelex
- conda install –c conda-forge scikit-learn-intelex
- pip install scikit-learn-intelex

Intel® Extension for Scikit-Learn Performance compared to original Scikit-Learn (Training & Inference)





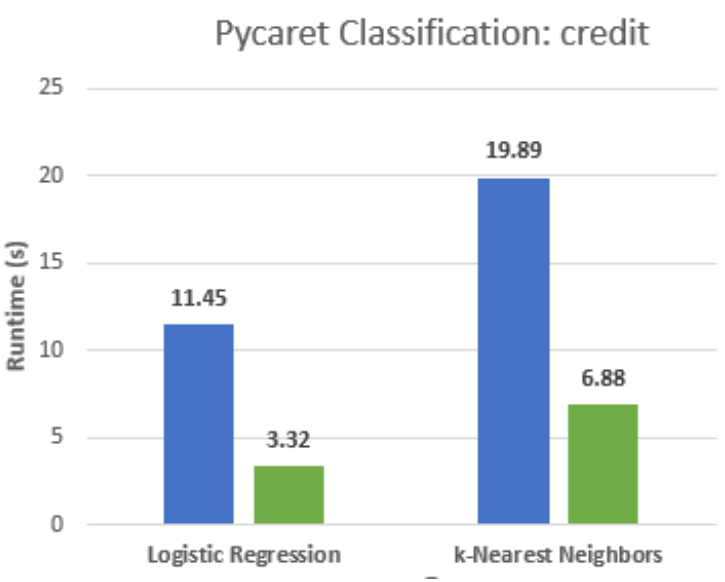
Configuration Details and Workload Setup: bare metal (2.0 GHz Intel Xeon Platinum 8480+, two sockets, 56 cores per socket), 512 GB DDR5 4800MT/s, Python 3.10, scikit-learn 1.2.0, scikit-learn-intelex 2023.0.1.

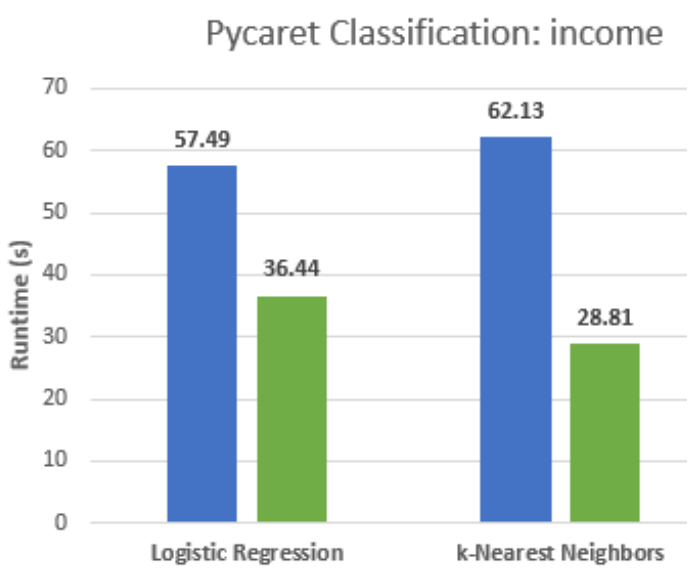
Available algorithms

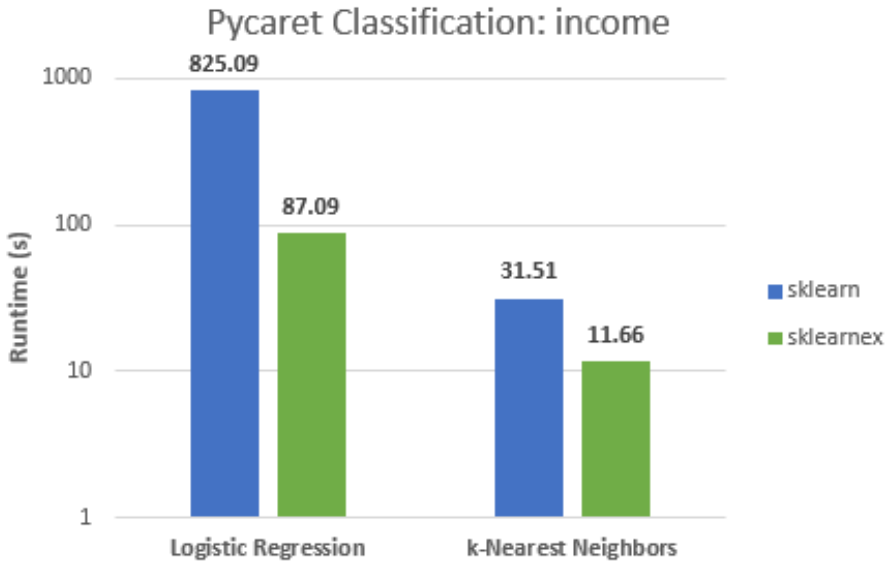
- Accelerated IDP Scikit-learn algorithms:
  - Linear/Ridge Regression
  - Logistic Regression
  - ElasticNet/LASSO
  - PCA
  - K-means
  - DBSCAN
  - SVC
  - train_test_split(), assume_all_finite()
  - Random Forest Regression/Classification
  - kNN (kd-tree and brute force)

PyCaret AutoML acceleration

Low-code auto ML with full sklearnex regression, classification, and clustering support





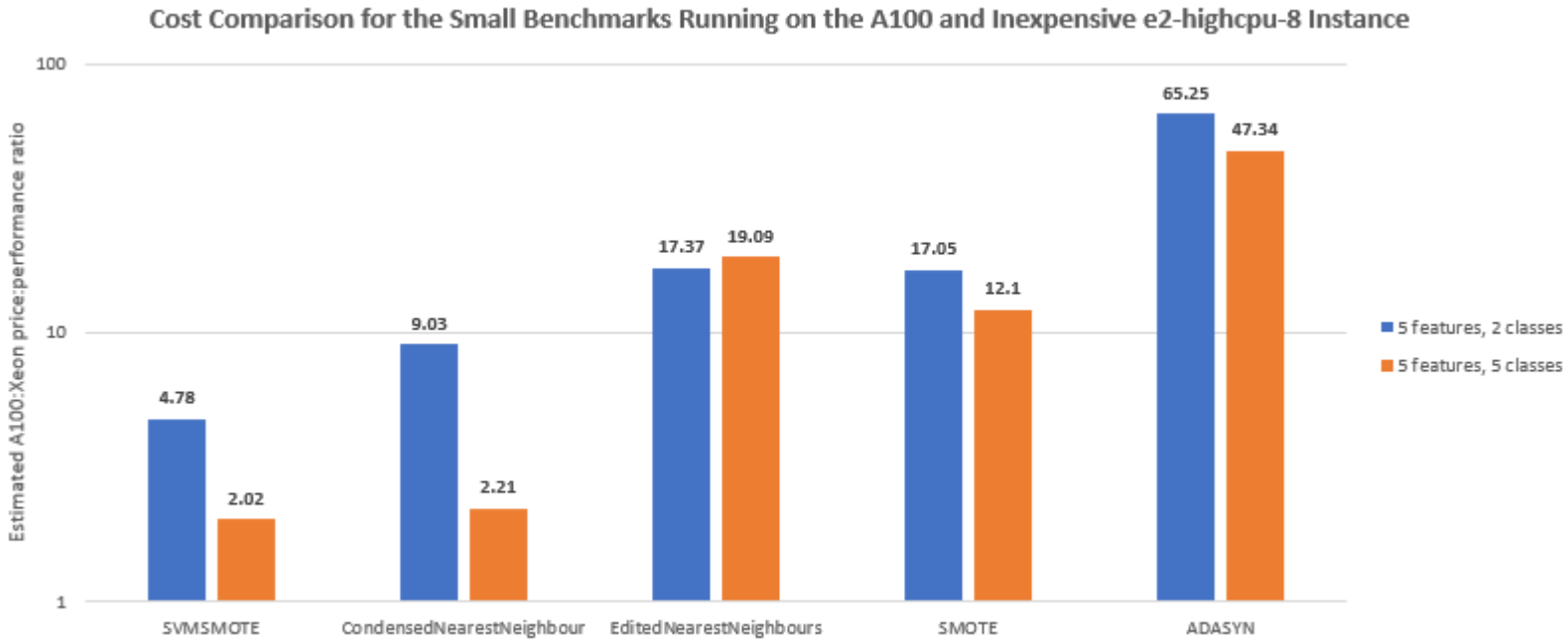


Scikit-learn CPU vs GPU TCO

Imbalanced learn acceleration study: better TCO than A100 (cuML)

[Why pay for mode for ML?]

https://medium.com/intel-analytics-software/why-pay-more-for-machine-learning-893683bd78e4

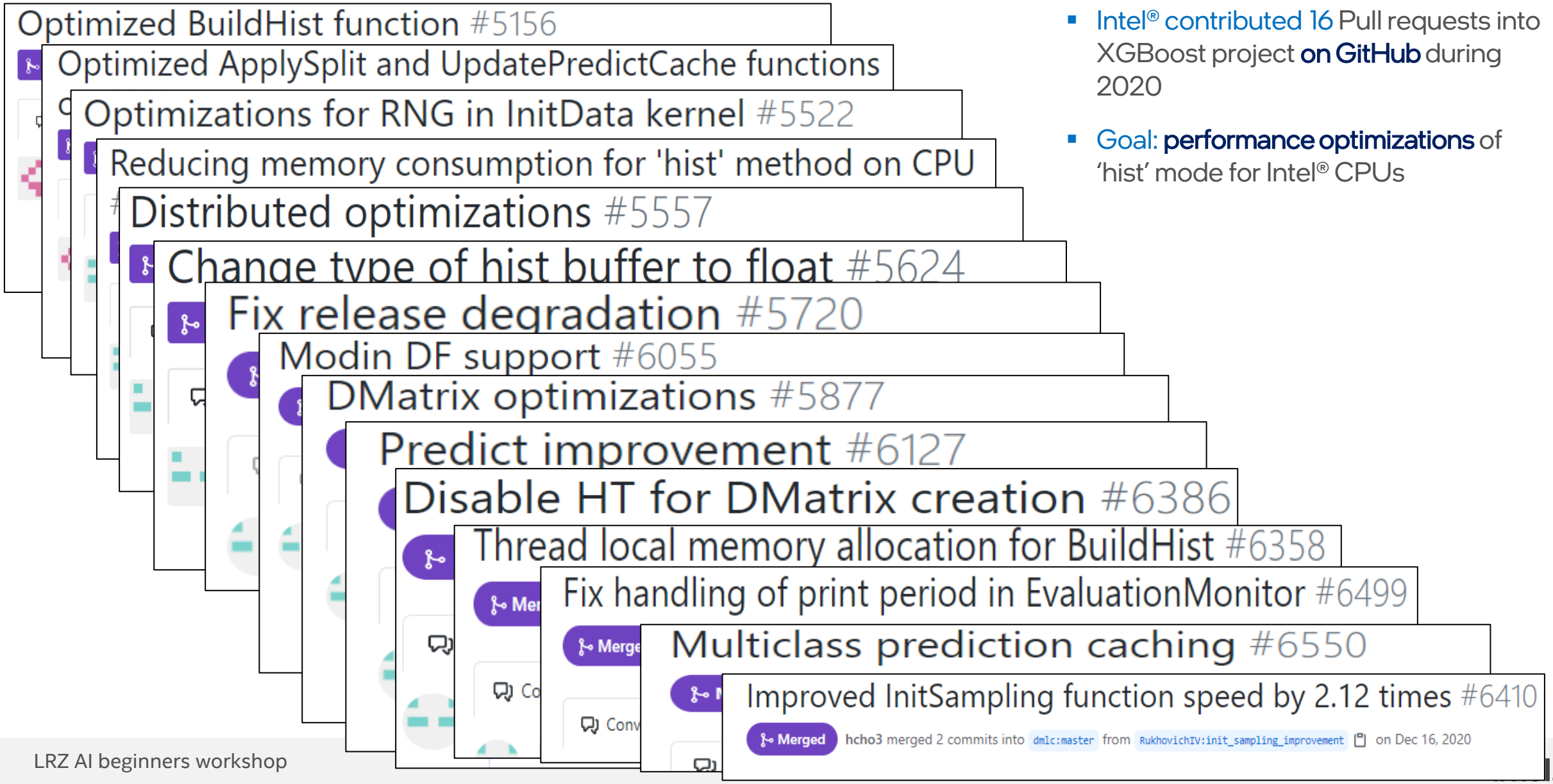


Gradient Boosting - Overview

Gradient Boosting:

- Boosting algorithm (Decision Trees - base learners)
- Solve many types of ML problems (classification, regression, learning to rank)
- Highly-accurate, widely used by Data Scientists
- Compute intensive workload
- Known implementations: XGBoost*, LightGBM*, CatBoost*, Intel® oneDAL, …

DMLC XGBoost Acceleration



- Intel® contributed 16 Pull requests into XGBoost project **on GitHub** during 2020
- Goal: **performance optimizations** of 'hist' mode for Intel® CPUs

Optimizations

- Xgboost Training -> upstreamed
- Xgboost Inference -> have to switch to oneDAL backend

- Installation guide: Improve the Performance of XGBoost and LightGBM Inference

```
conda install -c conda-forge daal4py'>=2020.3'
```

(https://www.intel.com/content/www/us/en/developer/articles/technical/improve-performance-xgboost-lightgbm-inference.html)

Gradient Boosting Acceleration – gain sources

Pseudocode for XGBoost* (0.81) implementation

```
def ComputeHist(node):
  hist = []
  for i in samples:
    for f in features:
      bin = bin_matrix[i][f]
      hist[bin].g += g[i]
      hist[bin].h += h[i]
  return hist

def BuildLvl:
  for node in nodes:
    ComputeHist(node)

  for node in nodes:
    for f in features:
      FindBestSplit(node, f)

  for node in nodes:
    SamplePartition(node)
```

Memory prefetching to mitigate irregular memory access

Usage uint8 instead of uint32

SIMD instructions instead of scalar code

Nested parallelism

Advanced parallelism, reducing seq loops

Usage of AVX-512, vcompress instruction (from Skylake)

Pseudocode for Intel® oneDAL implementation

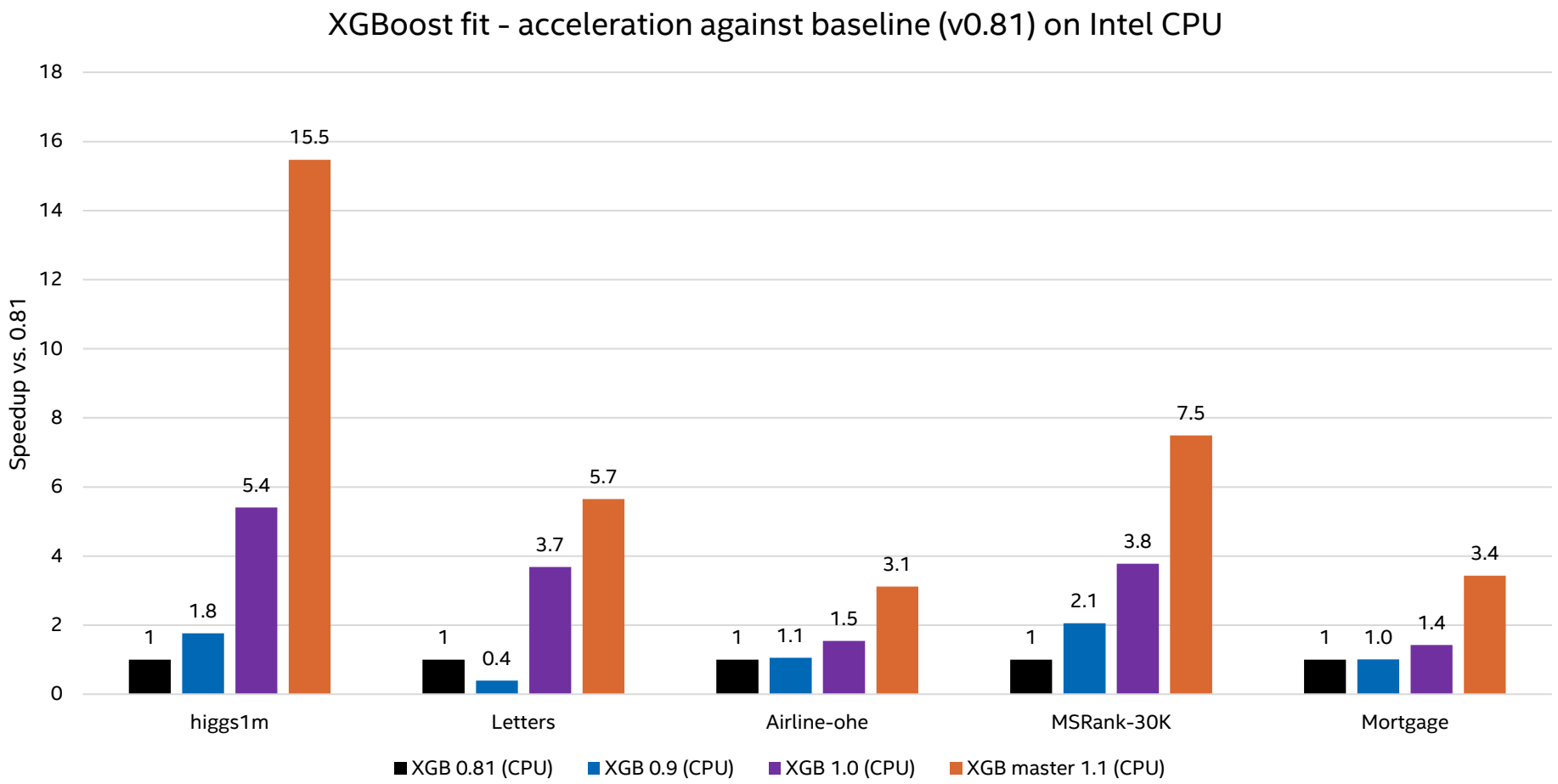
```
def ComputeHist(node):
  hist = []
  for i in samples:
    prefetch(bin_matrix[i + 10])
    for f in features:
      bin = bin_matrix[i][f]
      bin_value = load(hist[2*bin])
      bin_value = add(bin_value, gh[i])
      store(hist[2*bin], bin_value)
  return hist


def BuildLvl:
  parallel_for node in nodes:
    ComputeHist(node)

  parallel_for node in nodes:
    for f in features:
      FindBestSplit(node, f)

  parallel_for node in nodes:
    SamplePartition(node)
```

Training stage

Legend:

- Moved from Intel® oneDAL to XGBoost (v1.3)
- Already available in Intel® oneDAL, potential optimizations for XGBoost*

XGBoost* fit CPU acceleration (“hist” method)



+ Reducing memory consumption

memory, Kb	Airline	Higgs1m
Before	28311860	1907812
#5334	16218404	1155156
reduced:	1.75	1.65

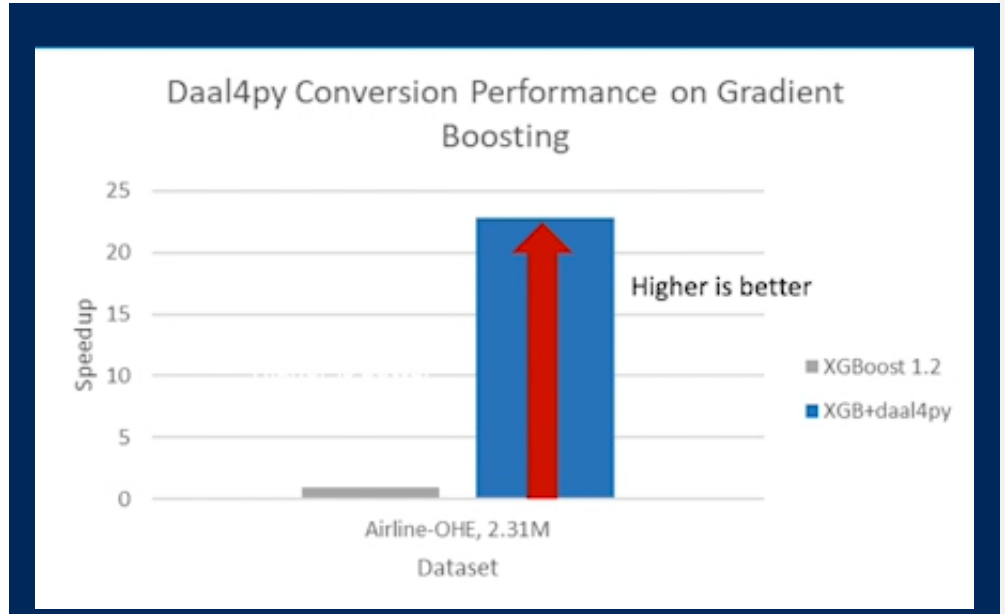
CPU configuration: c5.24xlarge AWS Instance, CLX 8275 @ 3.0GHz, 2 sockets, 24 cores per socket, HT:on, DRAM (12 slots / 32GB / 2933 MHz)

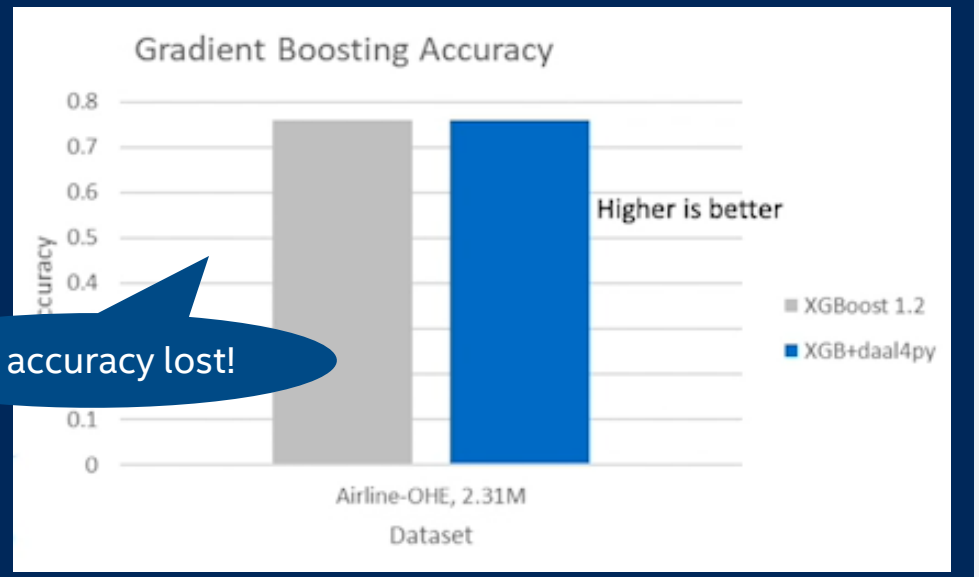
XGBoost* and LightGBM* Prediction Acceleration with Daal4Py

- Custom-trained XGBoost* and LightGBM* Models utilize Gradient Boosting Tree (GBT) from Daal4Py library for performance on CPUs
- No accuracy loss; 23x performance boost by simple model conversion into daal4py GBT:

```
# Train common XGBoost model as usual
xgb_model = xgb.train(params, X_train)

import daal4py as d4p

# XGBoost model to DAAL model
daal_model = d4p.get_gbt_model_from_xgboost(xgb_model)

# make fast prediction with DAAL
daal_prediction = d4p.gbt_classification_prediction(...).compute(X_test, daal_model)
```

- Advantages of daal4py GBT model:
  - More efficient model representation in memory
  - Intel® AVX512 instruction set usage
  - Better L1/L2 caches locality





No accuracy lost!

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.
See backup for configuration details.

CatBoost Performance Optimizations

CatBoost Latest 0.24.4 Version Note on GitHub:

"Major speedup asymmetric trees training time on CPU (2x speedup on Epsilon with 16 threads). We would like to recognize Intel software engineering team's contributions to Catboost project."

Speedup on several open-source datasets(larger is better):

Removed false-sharing issues detected by **Intel® VTune™ Profiler** (up to 1.47x speedup!)

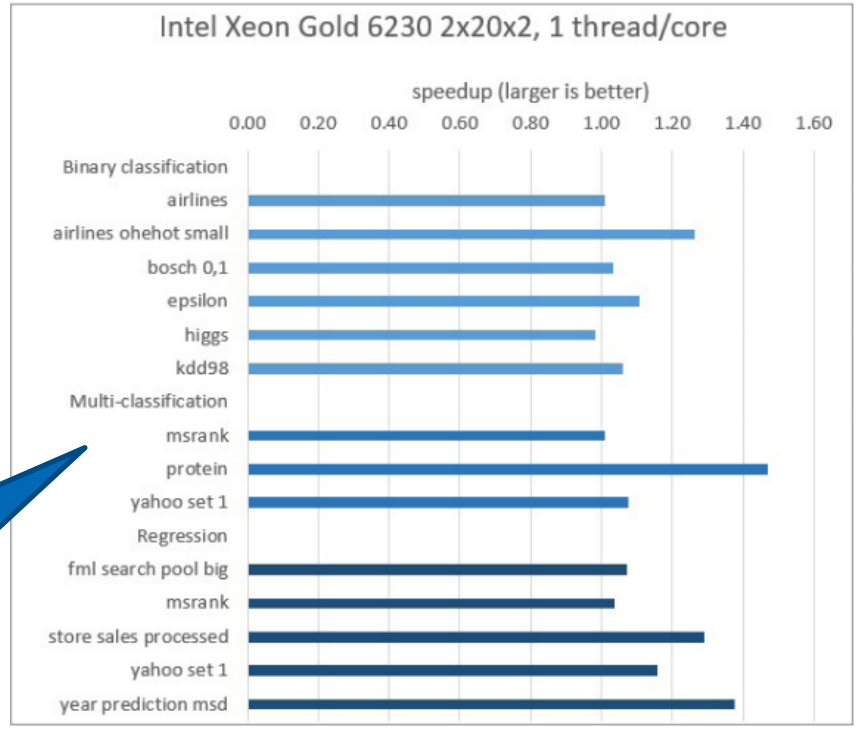


Figure 1. Intel Xeon processor 6230 used for training, 40 physical cores with 1 thread per physical core

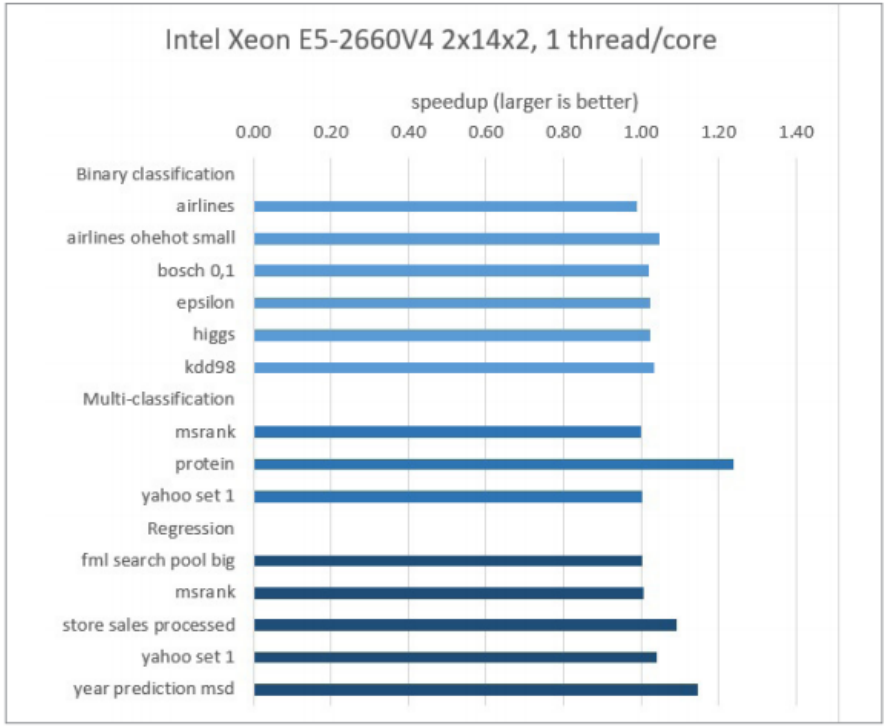


Figure 2. Intel Xeon processor E5-2660V4 with 2 sockets, 14 cores per socket, 2 HT per core, 1 thread per physical core

Source: https://software.intel.com/content/dam/develop/external/us/en/documents/yandex-case-study.pdf

Conclusion

Save Time with One-line Code Changes

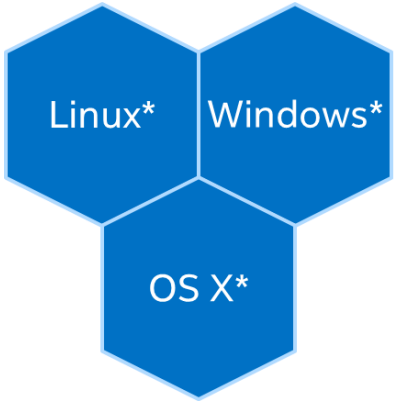
More model experimentation for higher accuracy

Engineer Data	Create Machine Learning & Deep Learning Models	Deploy
~90x	~38x	~3x
pandas MODIN	scikit learn	TensorFlow
import modin.pandas **as** pd	**from sklearnex import patch_sklearn** patch_sklearn()	TF_ENABLE_ONEDNN_OPTS=1

See link below for workloads and configurations. Results may vary
https://www.intel.com/content/www/us/en/developer/articles/technical/code-changes-boost-pandas-scikit-learn-tensorflow.html

Choose Your Download Option

Python Solutions	Download Options
Tools and frameworks to accelerate end-to-end data science and analytics pipelines	Intel® AI Analytics Toolkit 
Develop fast, performant Python code with essential computational packages	Intel® Distribution for Python 
Optimized Python packages from package managers and containers	Conda | YUM | APT | Docker 
Develop in the Cloud	Intel® DevCloud Intel® DevCloud



* Also available in the Intel® oneAPI Base Toolkit

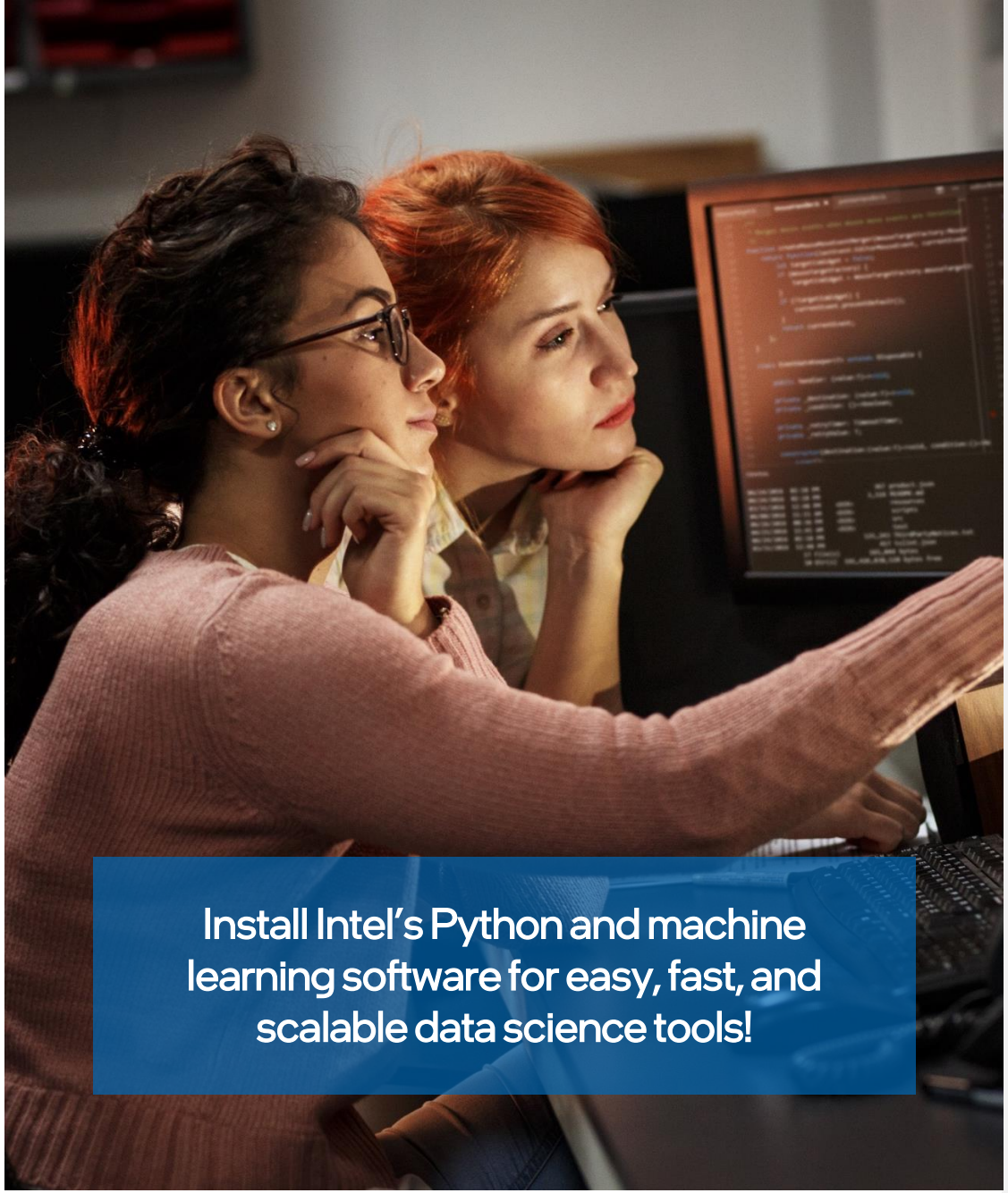
Call to Action

For more details on Intel® AI Analytics Toolkit and its software optimizations, please visit

- software.intel.com/en-us/oneAPI/ai-kit
- https://devcloud.intel.com/oneapi/
- AI Analytics Toolkit Support Forum

For more details on specific Intel's Python* Data Science software options, visit

- Intel oneContainer Portal
- Intel® AWS Containers
- Intel® oneAPI AI Analytics Toolkit Code Samples
- Intel® Distribution for Python Support Forum
- Machine Learning and Data Analytics Support Forum
- Intel AI Homepage



Questions?