Choose the Best Accelerated Technology

How to 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.

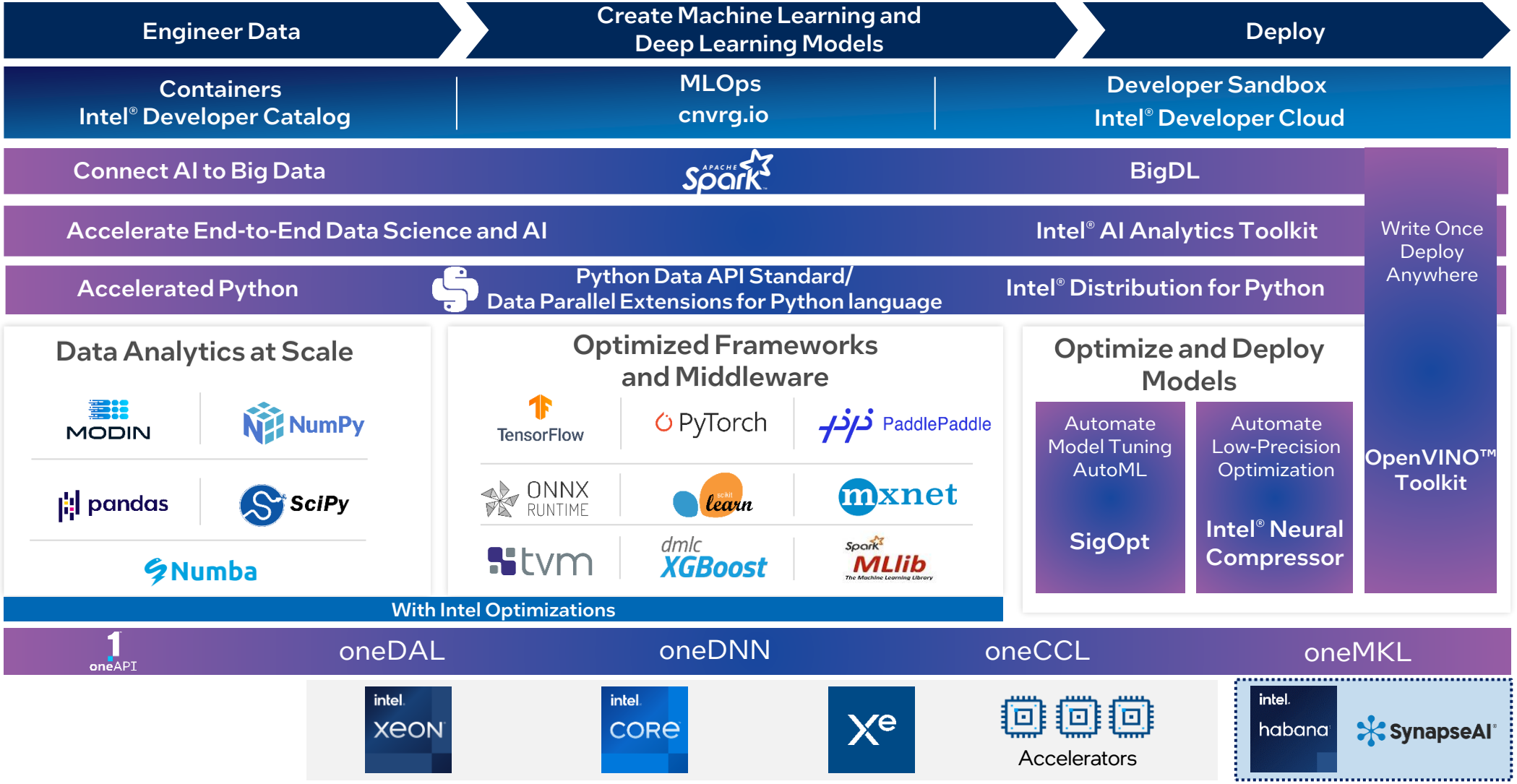
Testing Date: Performance results are based on testing by Intel® as of February 3, 2021 and may not reflect all publicly available updates. **Configuration Details and Workload Setup:** Intel® Optimization for PyTorch v1.5.0; Intel® Extension for PyTorch (IPEX) 1.1.0; oneDNN version: v1.5; DLRM: Training 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 and 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*
- Hands-on session with demos

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

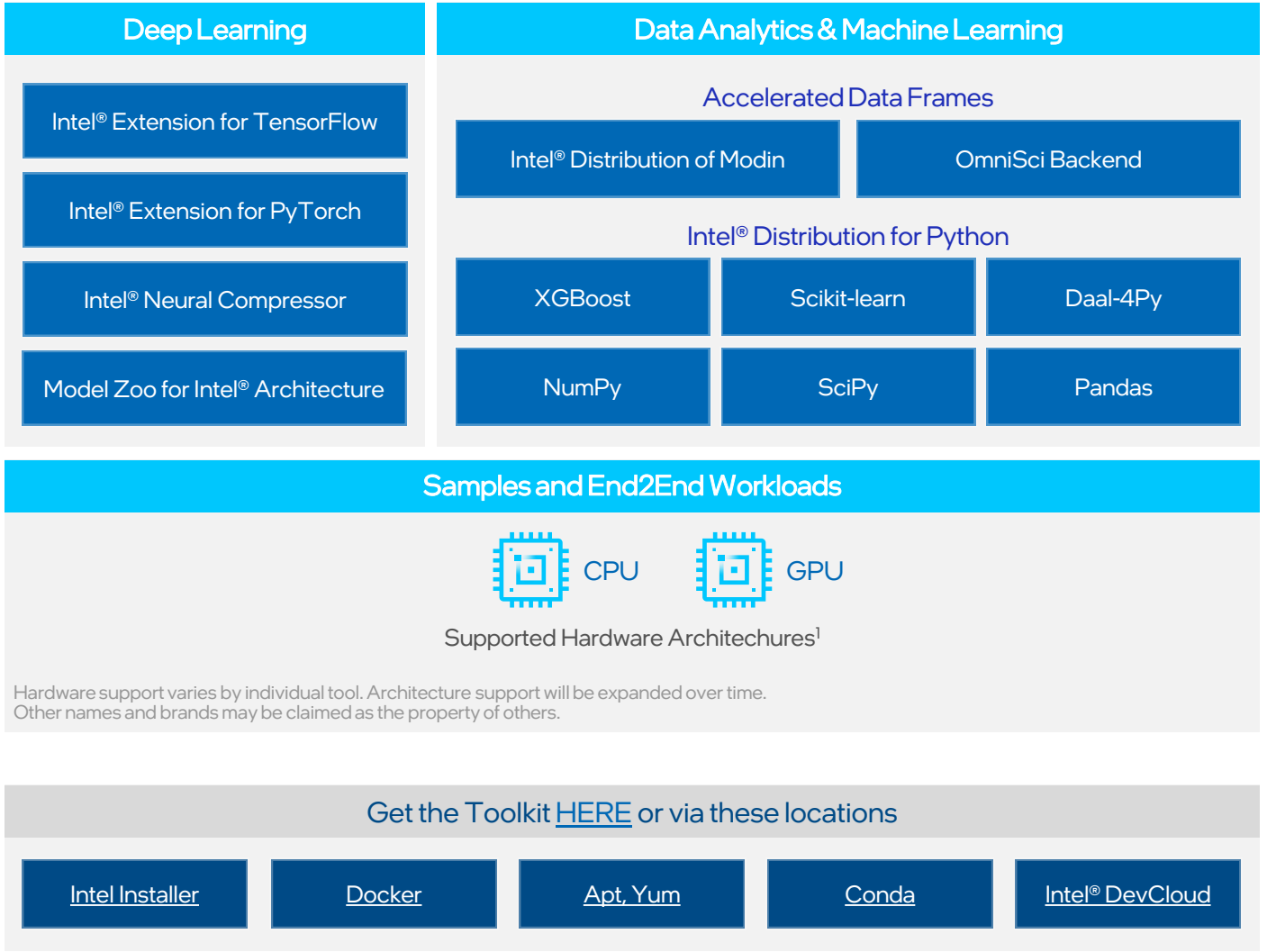
Accelerate end-to-end AI and data analytics pipelines with libraries optimized for Intel® architectures

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



Learn More: software.intel.com/oneapi/ai-kit

Back to Domain-specific Toolkits for Specialized Workloads

Maximize the Power of Intel® XPUs for Data Science, Machine Learning, and AI Workflows

Accelerate Performance

Maximize machine learning performance for multiple architectures (Intel® CPUs/GPUs) with tools and components built using oneAPI libraries

Streamline End-2-End Workflows

Get the latest AI Analytics optimizations in one place that work seamlessly together, scale end-to-end workflows fast

No need to download and integrate multiple external packages together

Improve Productivity

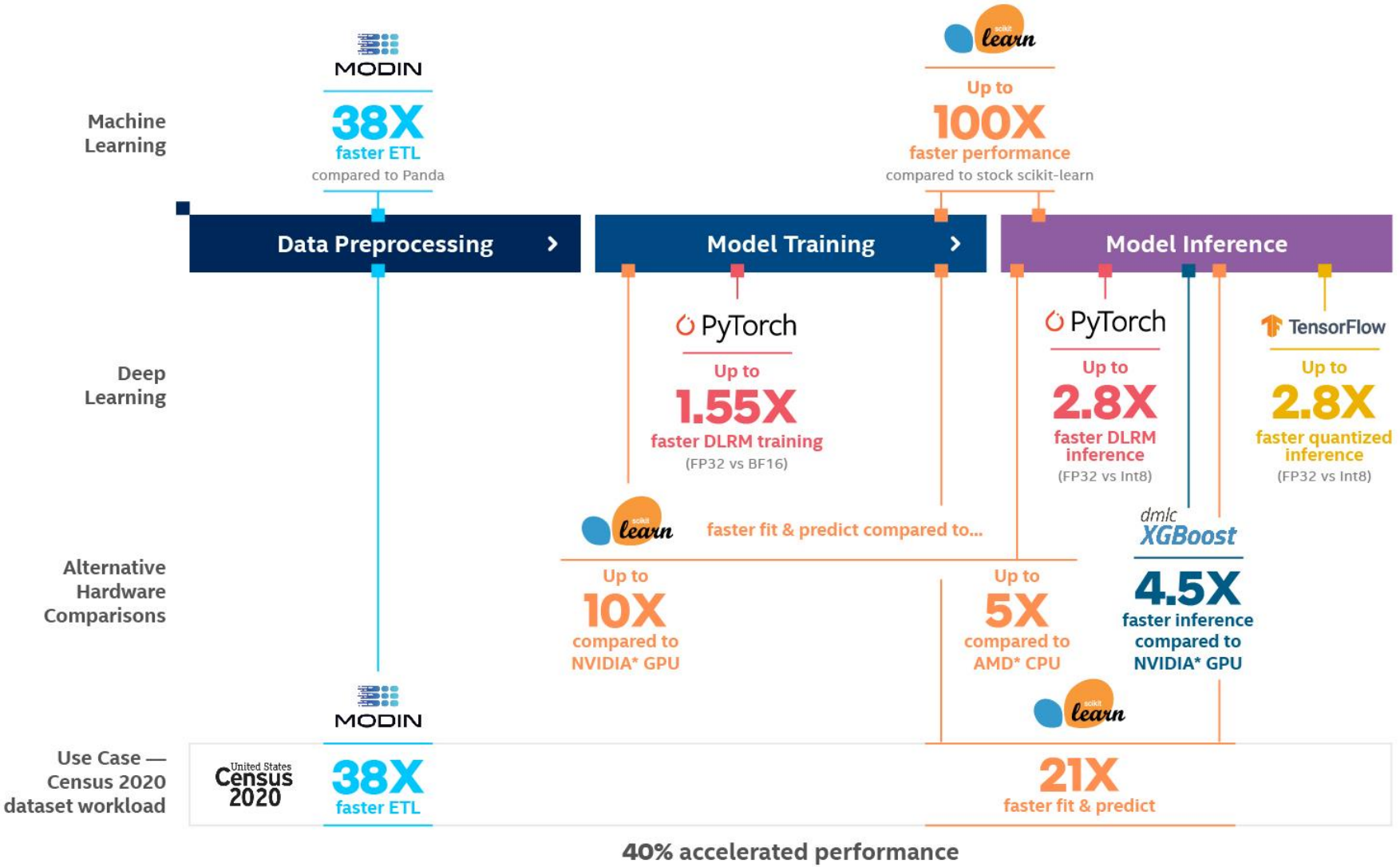
Alleviate the uncertainty associated with the Conda* package manager through a version-controlled binary installation

Speed Development

Reduce the learning curve with drop-in replacement for Python* packages with minimal to no code changes

Get started quickly with samples, pre-trained models, and end-to-end workloads

A Sneak Peek at the Performance Benefits

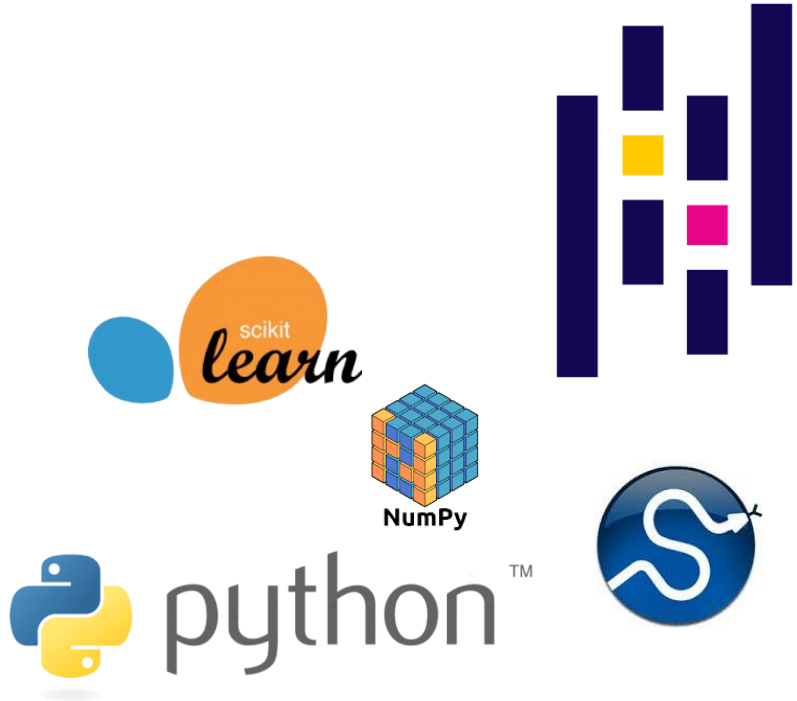


*Performance improvements shown here are based off hardware running on Intel Cascade Lake processors. This chart will be updated once data from Ice Lake is available. See backup for workloads and configurations. Results may vary.

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

Intel® Distribution for Python

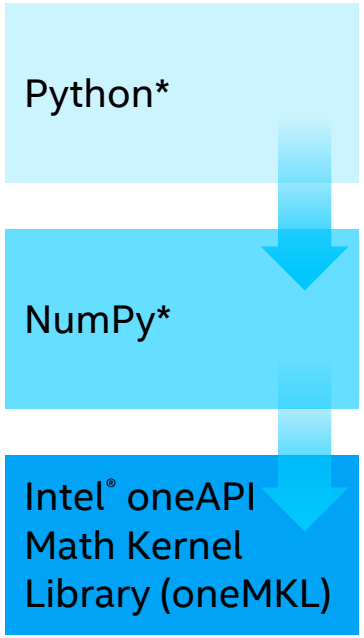
- Intel® Distribution for Python covers major usages in HPC and Data Science
- Achieve faster Python application performance — right out of the box — with minimal or no changes to a code
- Accelerate NumPy*, SciPy*, and scikit-learn* with integrated Intel® Performance Libraries such as Intel® oneMKL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

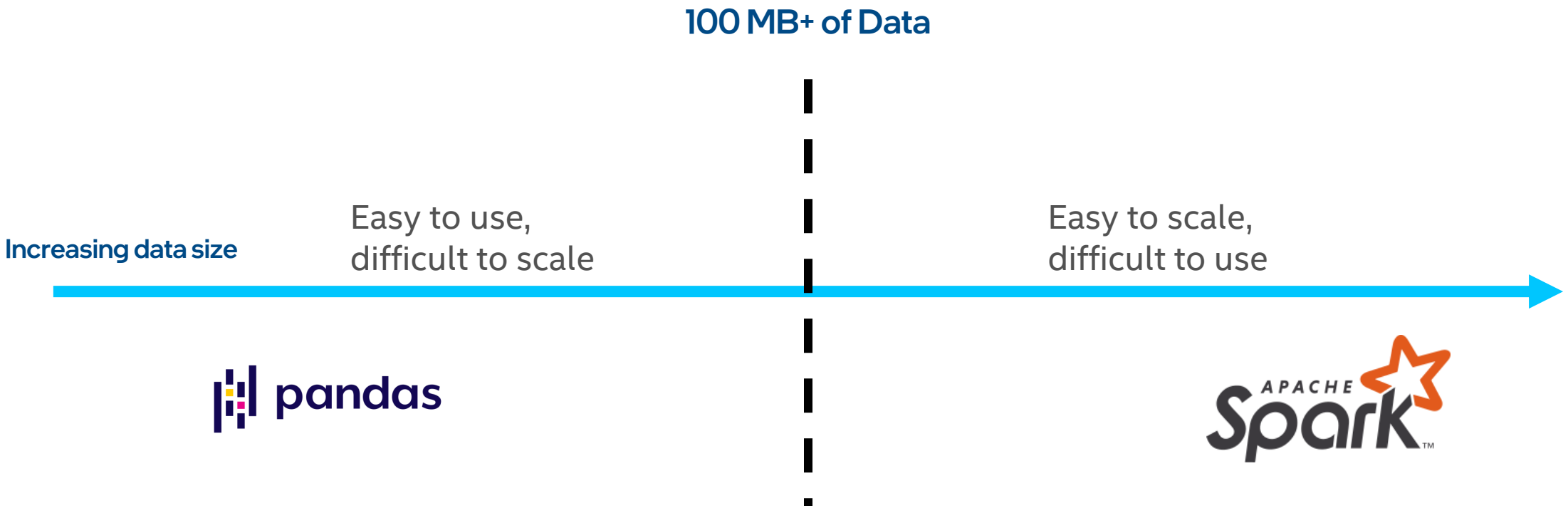
Gets around BLAS API bottleneck
Much stricter typing
Fastest performance level
Dispatches to hardware vectorization

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

Intel® 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
```

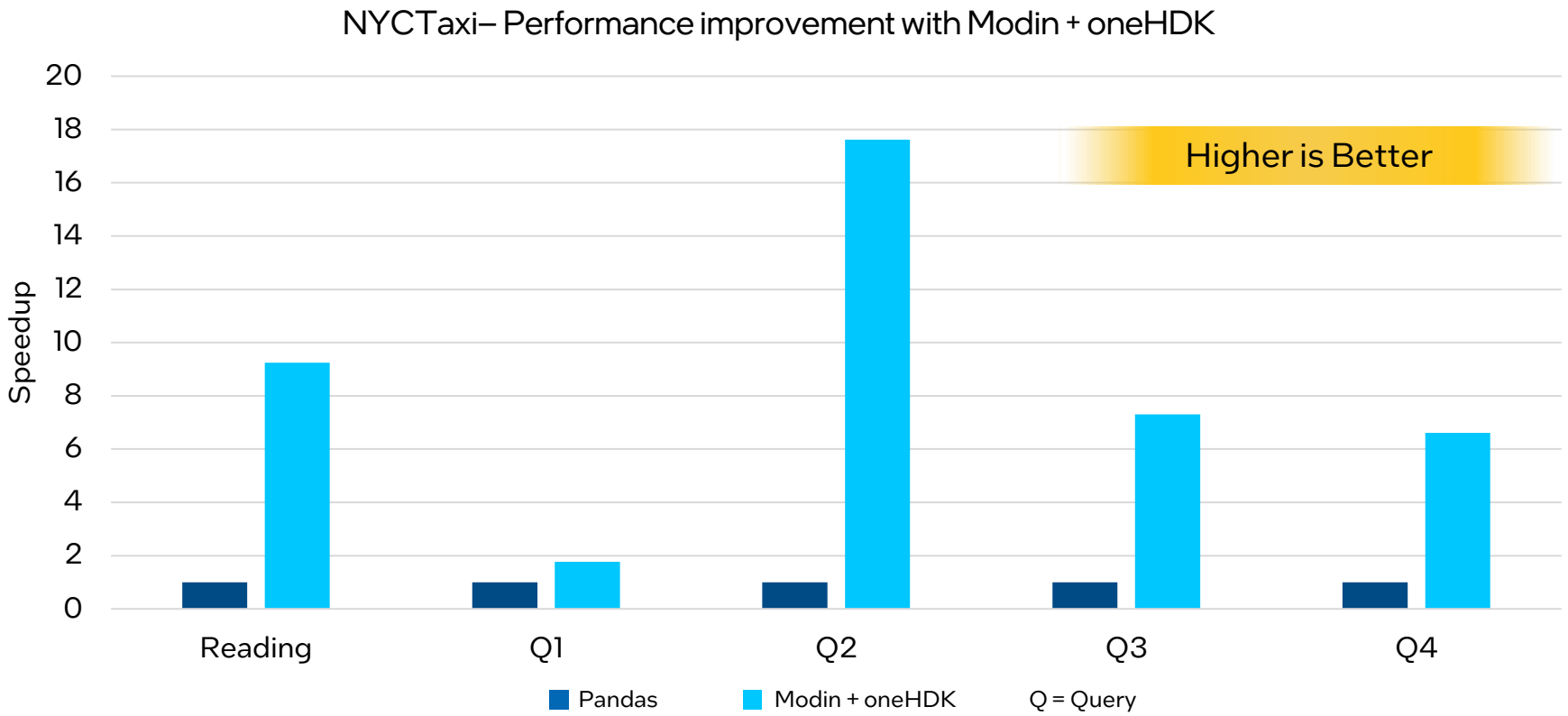
- Accelerate your Pandas* workloads across multiple cores and multiple nodes
- No upfront cost to learning a new API
  - import modin.pandas as pd
- Integration with the Python* ecosystem
- Integration with Ray/Dask clusters (run on what you have, even on a laptop!)
- Integration with Intel-built oneAPI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

NYCTaxi Workload Performance

- Pandas* vs. Modin*



Dataset source: https://github.com/toddwschneider/nyc-taxi-data

Configurations: For 20 million rows: Dual socket Intel(R) Xeon(R) Platinum 8280L CPUs (S2600WFT platform), 28 cores per socket, hyperthreading enabled, turbo mode enabled, NUMA nodes per socket=2, BIOS: SE5C620.86B.02.01.0013.121520200651, kernel: 5.4.0-65-generic, microcode: 0x4003003, OS: Ubuntu 20.04.1 LTS, CPU governor: performance, transparent huge pages: enabled, System DDR Mem Config: slots / cap / speed: 12 slots / 32GB / 2933MHz, total memory per node: 384 GB DDR RAM, boot drive: INTEL SSDSC2BB800G7. For 1 billion rows: Dual socket Intel Xeon Platinum 8260M CPU, 24 cores per socket, 2.40GHz base frequency, DRAM memory: 384 GB 12x32GB DDR4 Samsung @ 2666 MT/s 1.2V, Optane memory: 3TB 12x256GB Intel Optane @ 2666MT/s, kernel: 4.15.0-91-generic, OS: Ubuntu 20.04.4

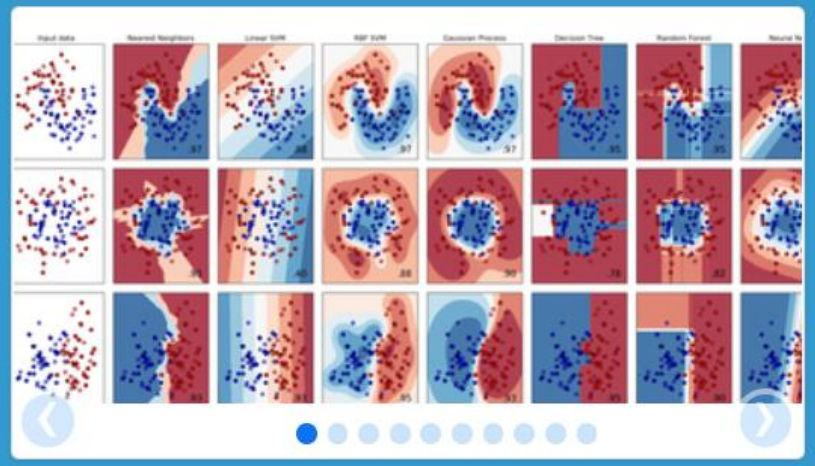
Intel® Extension for Scikit-learn* and XGBoost*

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 mainline

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

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

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

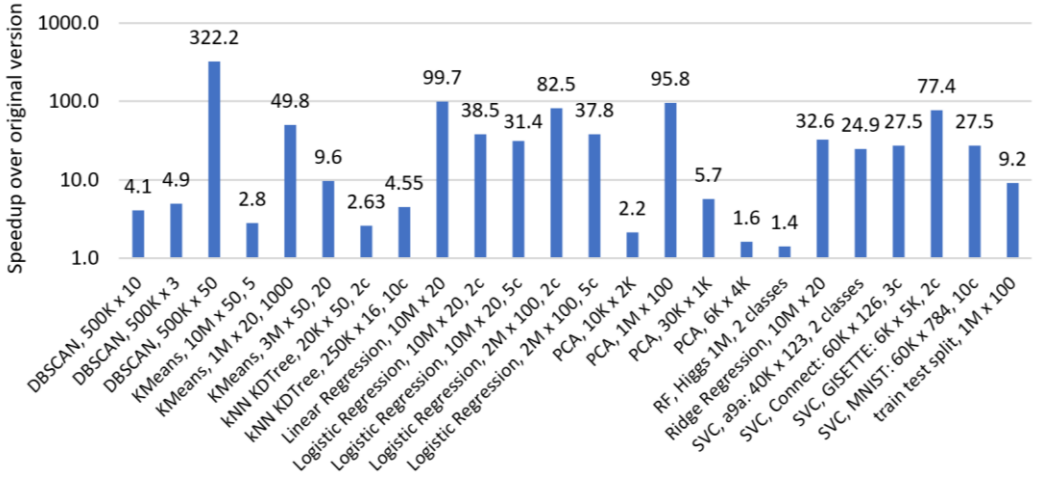
Available algorithms

- 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)

Intel® Extension for Scikit-learn*

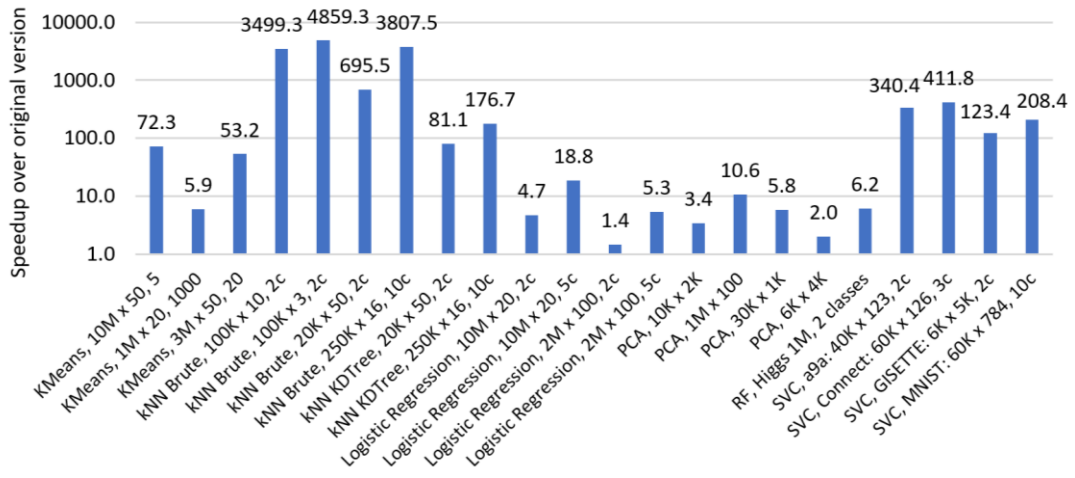
- Intel® Extension for Scikit-learn* Performance on CLX compared to original Scikit-learn* (Training and Inference)

Speedups of Intel® Extension for Scikit-learn training over the original Scikit-learn



Higher is Better

Speedups of Intel® Extension for Scikit-learn inference over the original Scikit-learn



Higher is Better

Testing Date: Performance results are based on testing by Intel as of June 8, 2021 and may not reflect all publicly available security updates.
Configuration Details and Workload Setup: c5.24xlarge AWS EC2 (3.0 GHz Intel Xeon Platinum 8275CL, two sockets, 24 cores per socket) Python 3.8, scikit-learn 0.24.2, scikit-learn-intelex 2021.2.3.
Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. 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

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, …

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

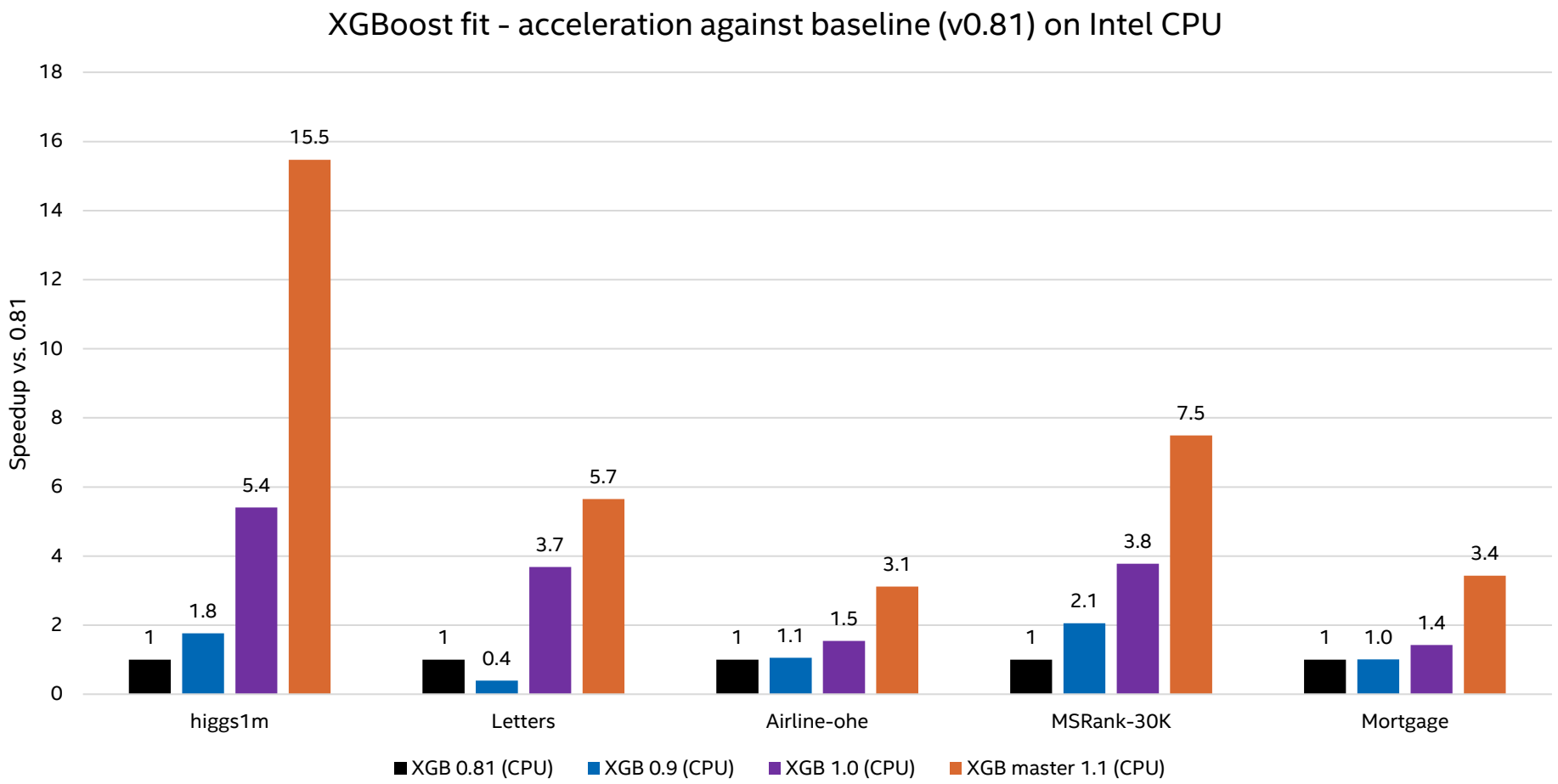
```
def ComputeHist(node):
  hist = []
  for i in samples:
    prefetch(bin_matrix[i + 10])
    for f in features:
      bin = bin_matrix[i][f]
      bin_value = load(hist[2*bin])
      bin_value = add(bin_value, gh[i])
      store(hist[2*bin], bin_value)
  return hist


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)

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'
```

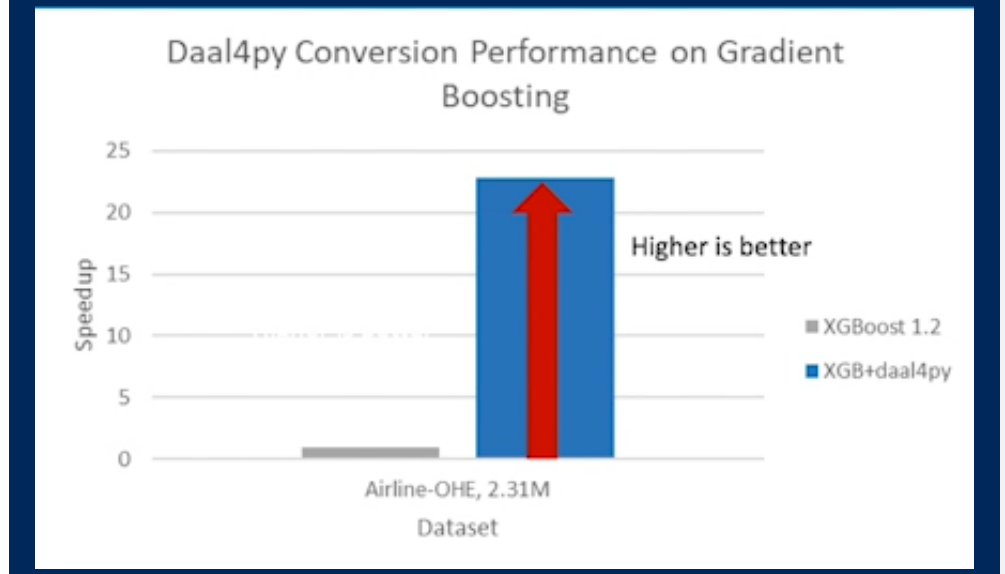
(https://www.intel.com/content/www/us/en/developer/articles/technical/improve-performance-xgboost-lightgbm-inference.html)

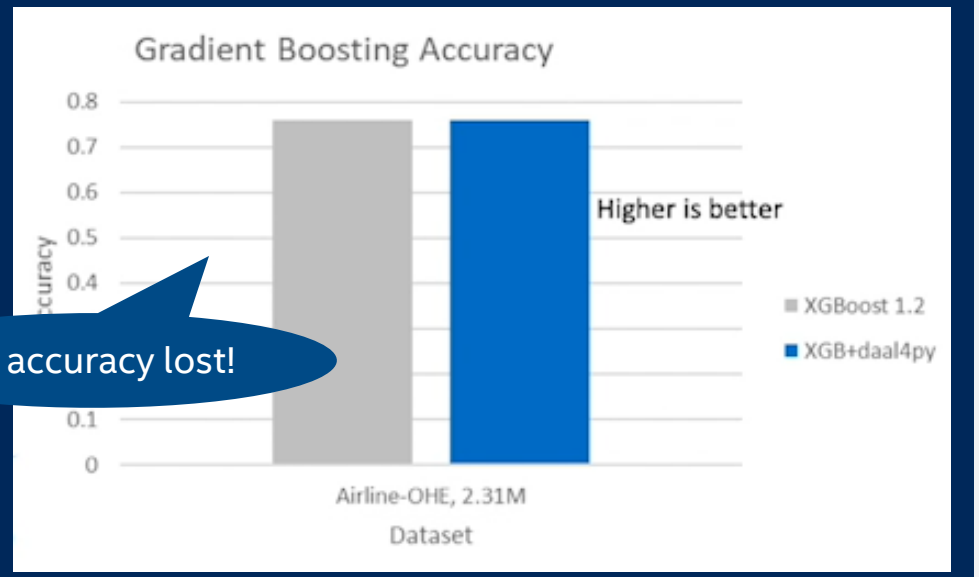
XGBoost* and LightGBM* Prediction Acceleration with Daal4Py

- 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

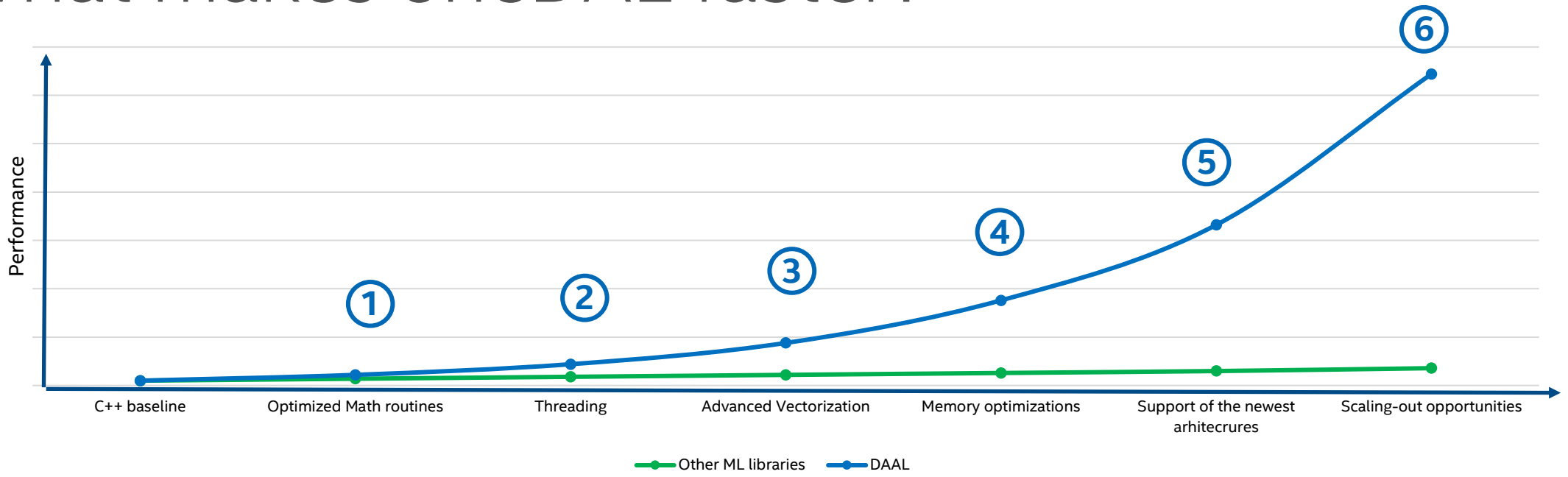




No accuracy lost!

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

What makes oneDAL faster?



1. The best performance on Intel Architectures with oneMKL vs. less performance OS BLAS/LAPACK libs

2. onDAL targets to many-core systems to achieve the best scalability on Intel® Xeon, other libs mostly target to client versions with small amount of cores

3. oneDAL uses the latest available vector-instructions on each architecture, enables them by compiler options, intrinsics. Usually other ML libs build application without vector-instructions support or sse4.2 only.

4. oneDAL's uses the most efficient memory optimization practices: minimally access memory, cache access optimizations, SW memory prefetching. Usually Other ML libs don't make low-level optimizations.

5. oneDAL enables new instruction sets and other HW features even before official HW lunch. Usually other ML libs do this with long delay.

6. oneDAL provides distributed algorithms which scale on many nodes

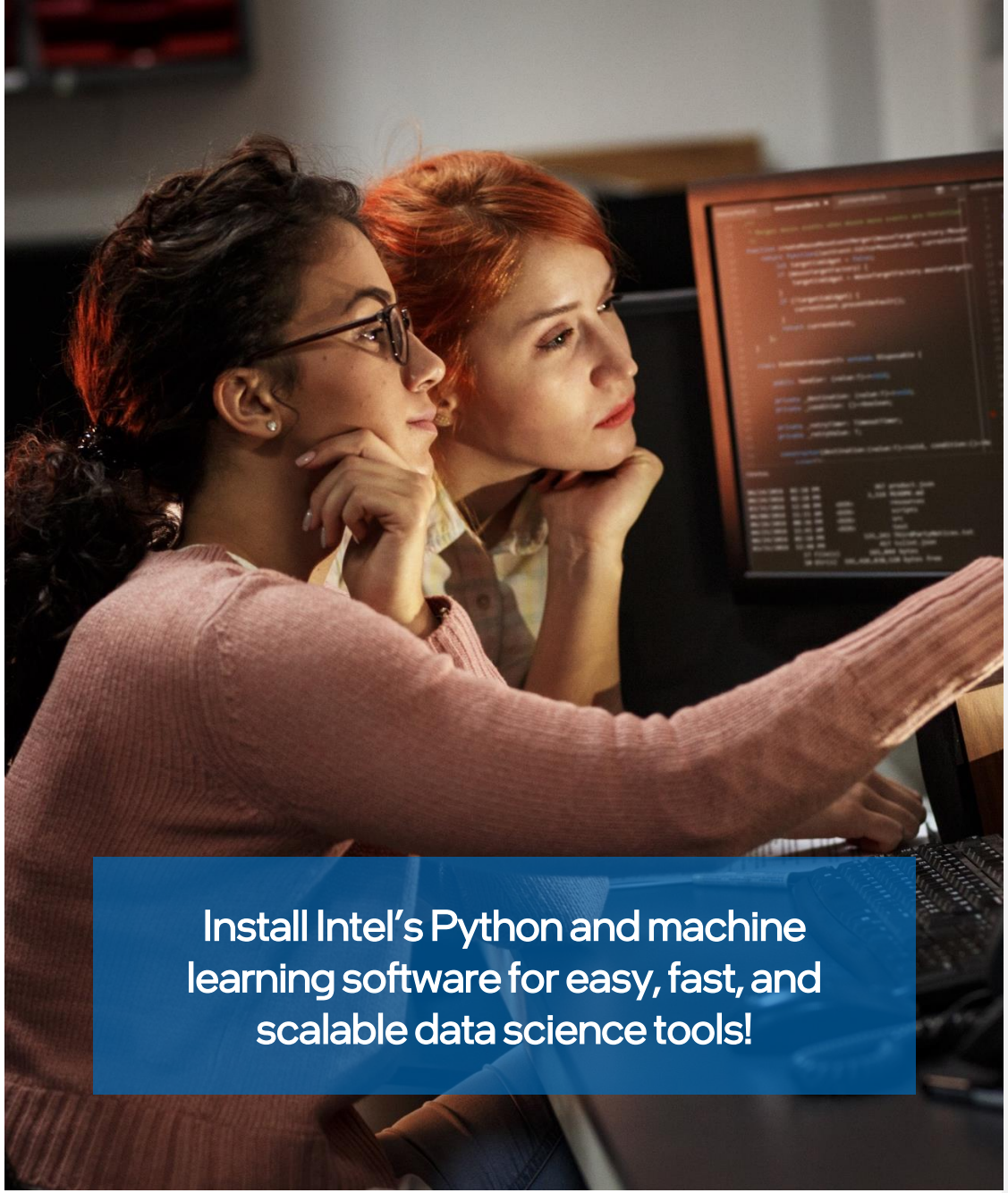
Call to Action

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



Demo time!

Questions?