Data Analytics & Classical ML

intel®

Intel® AI Analytics Toolkit

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

Who needs this product?

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

Intel® AI Analytics Toolkit

Deep Learning

- Intel® Optimization for TensorFlow
- Intel® Optimization for PyTorch
- Intel® Neural Compressor
- Model Zoo for Intel® Architecture

Machine Learning

- Intel® Extension for Scikit-learn
- Intel-optimized XGBoost

Data Analytics

- Intel® Distribution of Modin
- OmniSci Backend

Intel-optimized Python

- NumPy
- SciPy
- Numba
- Pandas
- Data Parallel Python

CPU GPU

Hardware support varies by individual tool. Architecture support will be expanded over time.

Get the Toolkit HERE or via these locations

- Intel Installer
- Docker
- Apt, Yum
- Conda
- Intel® DevCloud

Back to Domain-specific Toolkits for Specialized Workloads

Intel® Distribution for Python

Developer Benefits

Maximize Performance	Minimize Development Cost	Vast Ecosystem
Performance Libraries, Parallelism, Multithreading, Language Extensions	**Drop-in Python Replacement**	**Familiar usage and compatibility**
Near-native performance comes through acceleration of core Python numerical packages Accelerated NumPy/SciPy/scikit-learn with oneMKL & oneDAL Data analytics, machine learning & deep learning with scikit-learn, XGBoost, Modin, daal4py Scale with Numba*, Cython*, tbb4py, mpi4py, SDC Optimized for latest Intel® architectures	Prebuilt optimized packages for numerical computing, machine/deep learning, HPC, & data analytics Data-Parallel Python provides cross-architecture XPU support Conda build recipes included in packages Free download & free for all uses including commercial deployment	Supports Python 3 Supports conda & pip package managers Packages available via conda, pip YUM/APT, Docker image on DockerHub Commercial support through the Intel® oneAPI Base Toolkit

Operating Systems: Windows*, Linux*, MacOS[1]*

Intel® Architecture Platforms CPU GPU OTHER ACCEL.

Intel® oneAPI Data Analytics Library (oneDAL)

Deploy High-Performance Data Science on CPUs and GPUs

Machine Learning & Data Analytics Performance

- Helps applications deliver better predictions faster
- Optimizes data ingestion & algorithmic compute together for highest performance
- Supports offline, streaming & distributed usage models to meet a range of application needs
- Split analytics workloads between edge devices and cloud to optimize overall application throughput

GPU Support with oneDAL

The following algorithms are supported:

- **Statistical:** Correlation, Low-order moments*
- **Classification**: Linear Regression*, Logistic Regression*, KNN, SVM
- **Unsupervised Learning**: K-means clustering, DBSCAN
- **Classification & Regression**: Random Forest
- **Dimensionality Reduction**: PCA

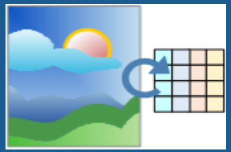
What's New: Full Support of scikit-learn[1] 1.2

*

Pre-processing

Decompression, Filtering, Normalization

Transformation



Aggregation, Dimension Reduction

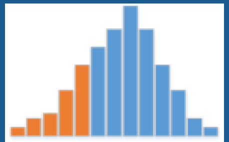
Analysis

Summary Statistics, Clustering, etc.

Modeling

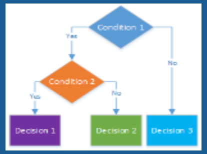
Machine Learning (Training) Parameter Estimation, Simulation

Validation



Hypothesis Testing, Model Errors

Decision Making



Forecasting, Decision Trees, etc.

[Learn More & Download]

* GPU implementation and existing oneDAL - oneAPI Initiative Specification represent a growing subset of CPU implementation.

[1] Other names and brands may be claimed as the property of others

Current Data Loading & ETL Landscape

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

100 MB+ of Data

Easy to use,
difficult to scale

Easy to scale,
difficult to use

Increasing data size

Single Line Code Change for Infinite Scalability

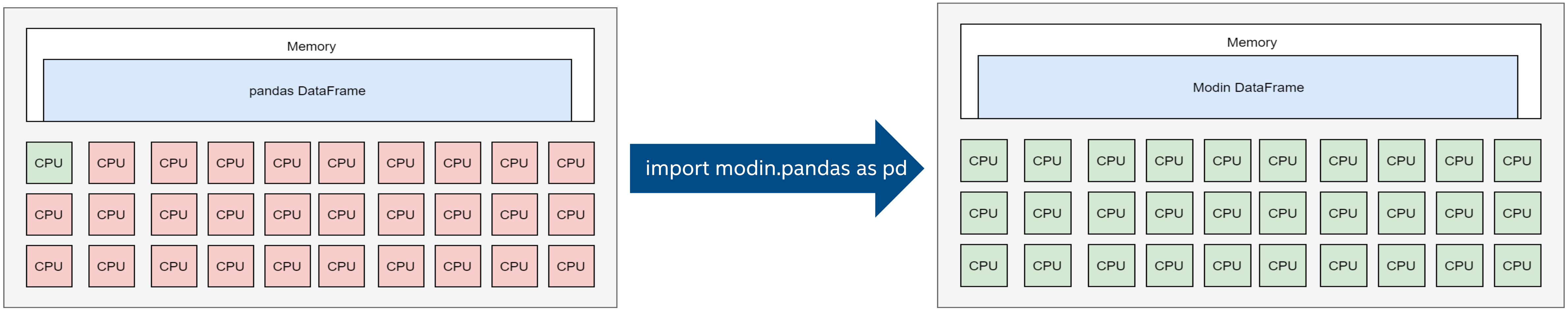
MODIN

- **No need to learn a new API to use Modin**

```
import pandas as pd
```

Pandas* on Big Machine

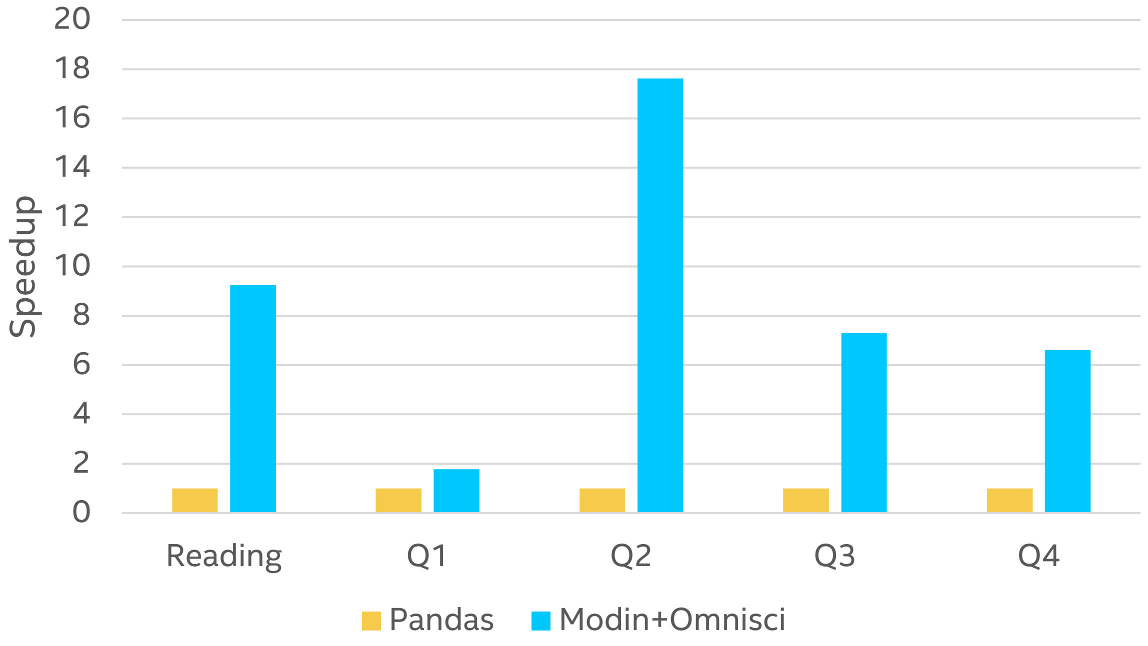
Modin on Big Machine



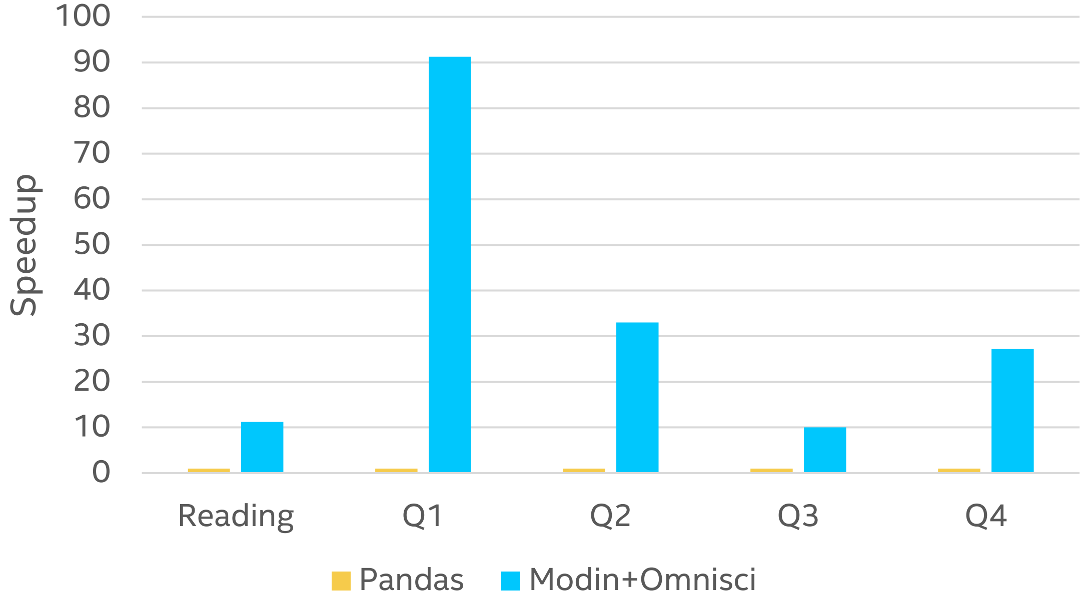
NYCTaxi Workload Performance

Pandas vs Modin – Higher is Better

NYCTaxi **(20 Million rows)** - Performance improvement with Modin+Omnisci



NYCTaxi **(1 Billion rows = 1.6 TB in mem)** - Performance improvement with Modin+Omnisci – using 3TB Optane



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

Common Scikit-learn

```python
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

```python
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)
```

Intel extension for sklearn

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

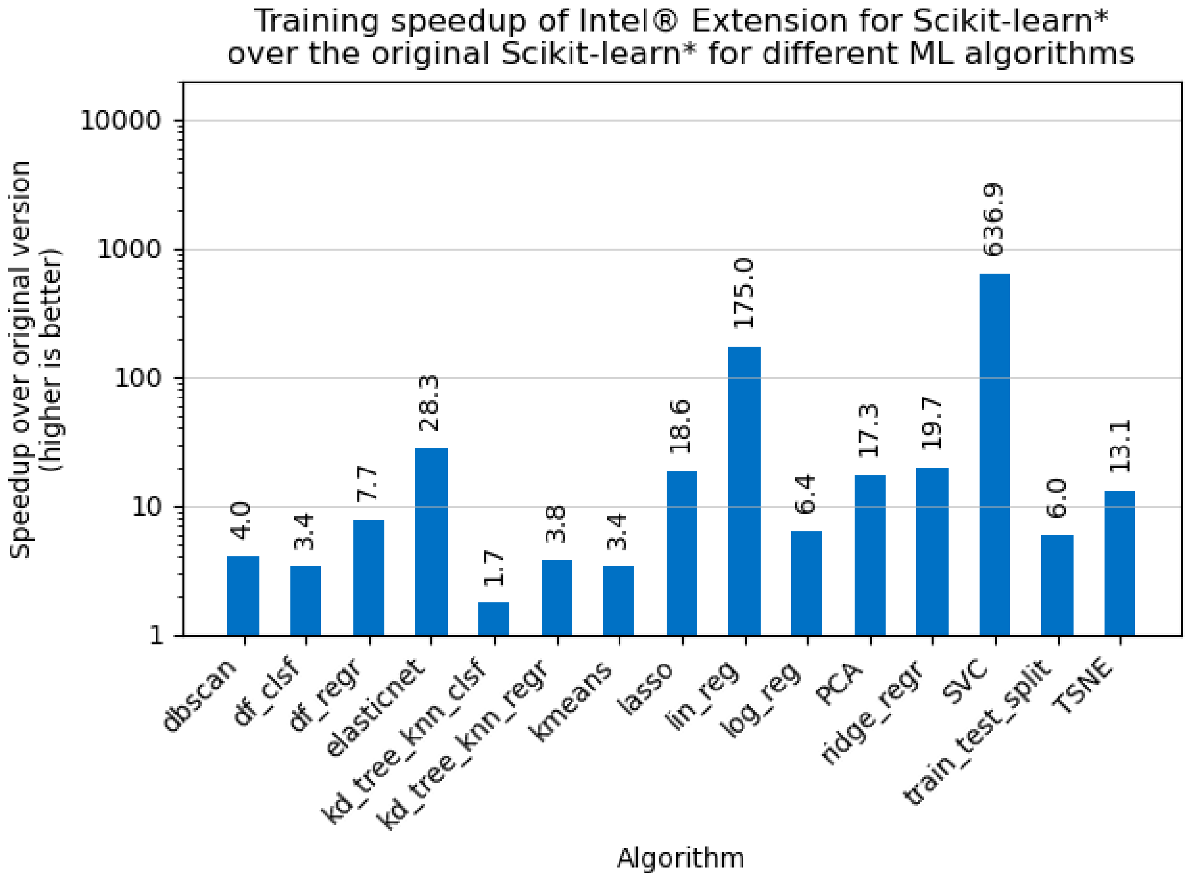
- Directly from the script:

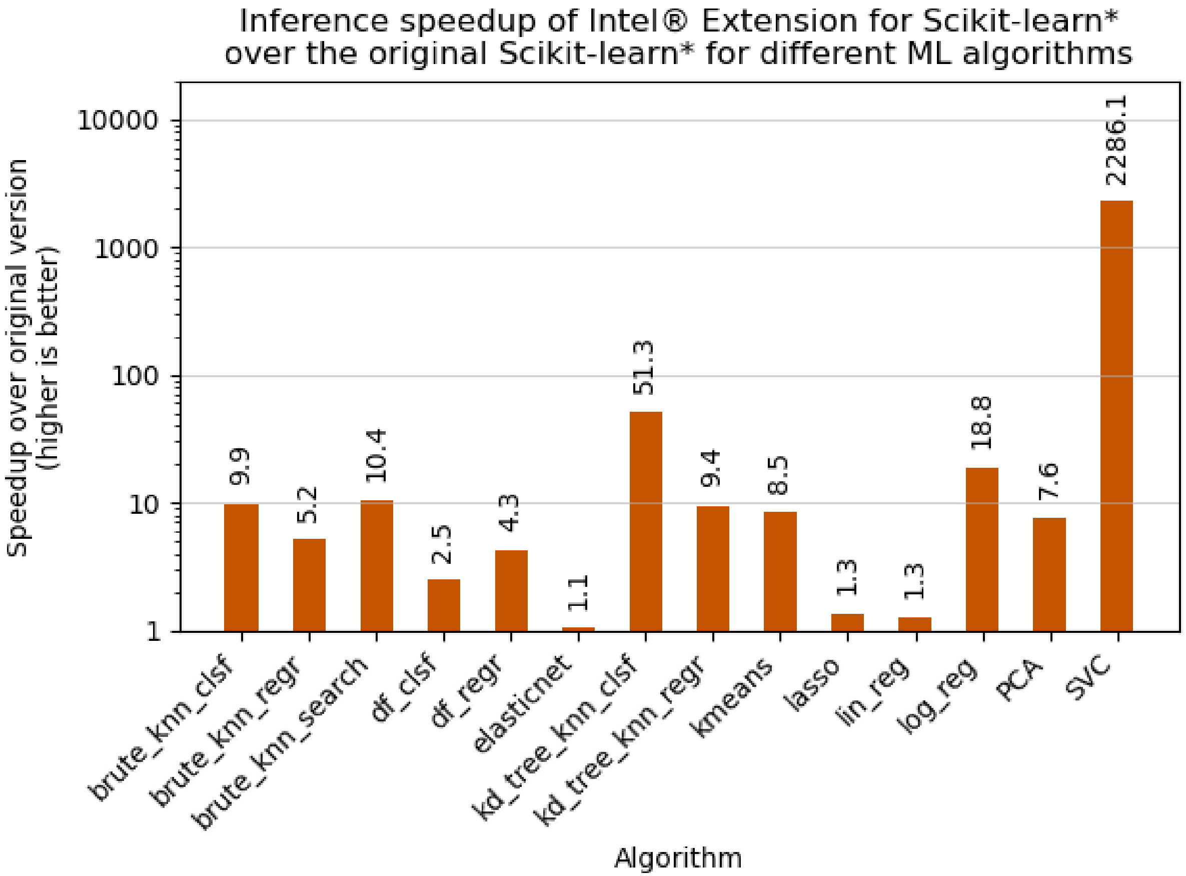
```python
from sklearnex import patch_sklearn
patch_sklearn()
```

- Through global patching to enable patching for your scikit-learn installation for all further runs:

```
python sklearnex.glob patch_sklearn
```

Intel® Extension for Scikit-learn* Performance





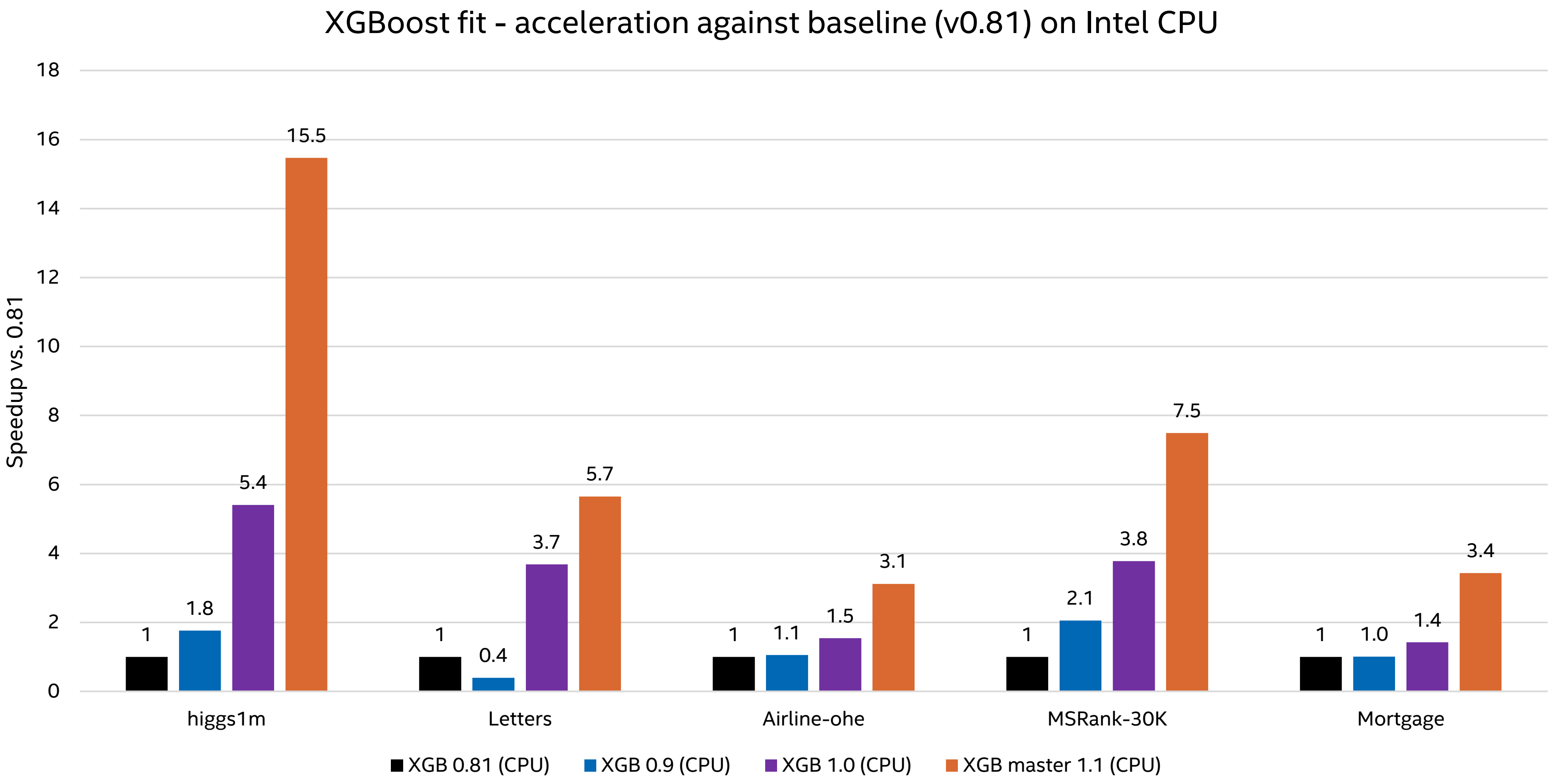
Testing Date: Performance results are based on testing by Intel as of March 21, 2023 and may not reflect all publicly available security updates.

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. Intel optimizations include use of multi-threading implementation for SKLearn algorithms (which are typically single-threaded), as well as other HW/SW optimizations.

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

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)

Installation: pip install xgboost

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
  - Avx512 instruction set usage
  - Better L1/L2 caches locality