Choose the Best Accelerated Technology

Accelerate Machine Learning Workloads with the Intel AI Analytics Toolkit

EUROCC AI Workshop

Roy Allela – AI Software Engineer

October 25th 2022

Notices and Disclaimers

Agenda

- Intel® AI Analytics Toolkit
- Intel® Distribution for Python
- Intel® Distribution of Modin
- Intel® Extension for Scikit-learn
- Intel® Optimization for XGBoost

Intel® AI Analytics Toolkit

Powered by oneAPI

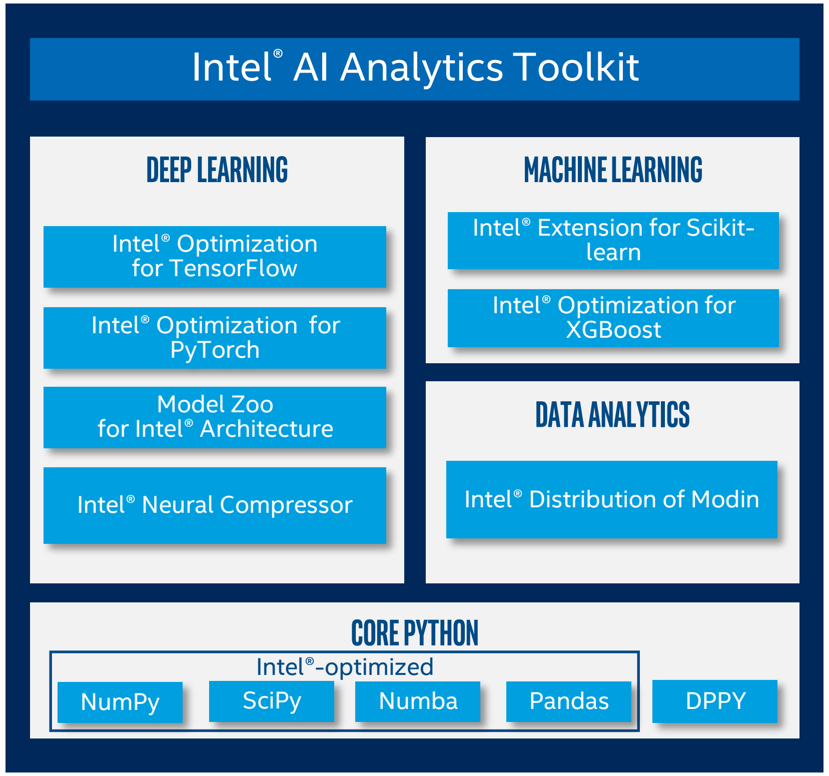
Accelerate end-to-end AI and data analytics pipelines with libraries optimized for Intel® architectures

Who Uses It?

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

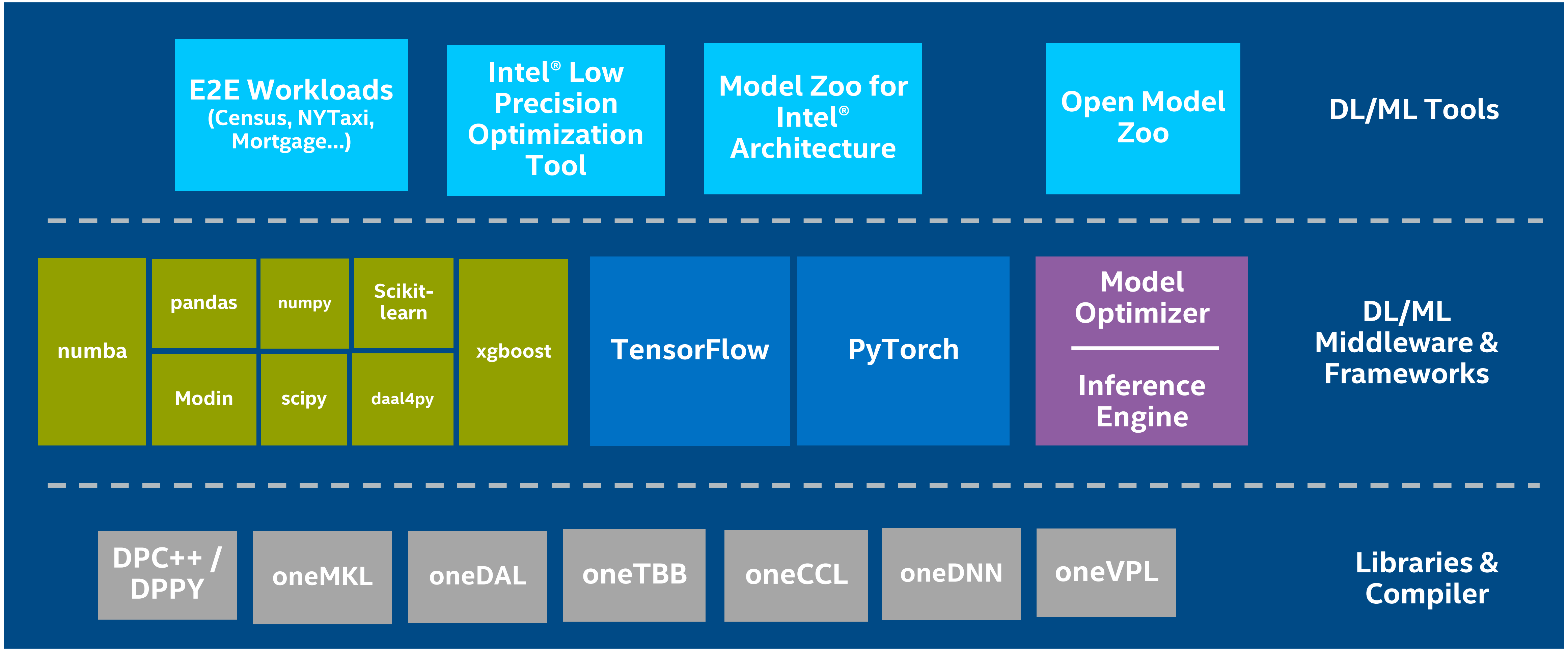
Top Features/Benefits

- 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


Intel® AI Analytics Toolkit
DEEP LEARNING
Intel® Optimization for TensorFlow
Intel® Optimization for PyTorch
Model Zoo for Intel® Architecture
Intel® Neural Compressor
MACHINE LEARNING
Intel® Extension for Scikit-learn
Intel® Optimization for XGBoost
DATA ANALYTICS
Intel® Distribution of Modin
CORE PYTHON
Intel®-optimized
NumPy
SciPy
Numba
Pandas
DPPY

AI Software Stack for Intel® XPUs

Intel offers a robust software stack to maximize performance of diverse workloads



Why Use the Intel® AI Analytics Toolkit?

Accelerate Performance

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

Streamline End2End 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

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

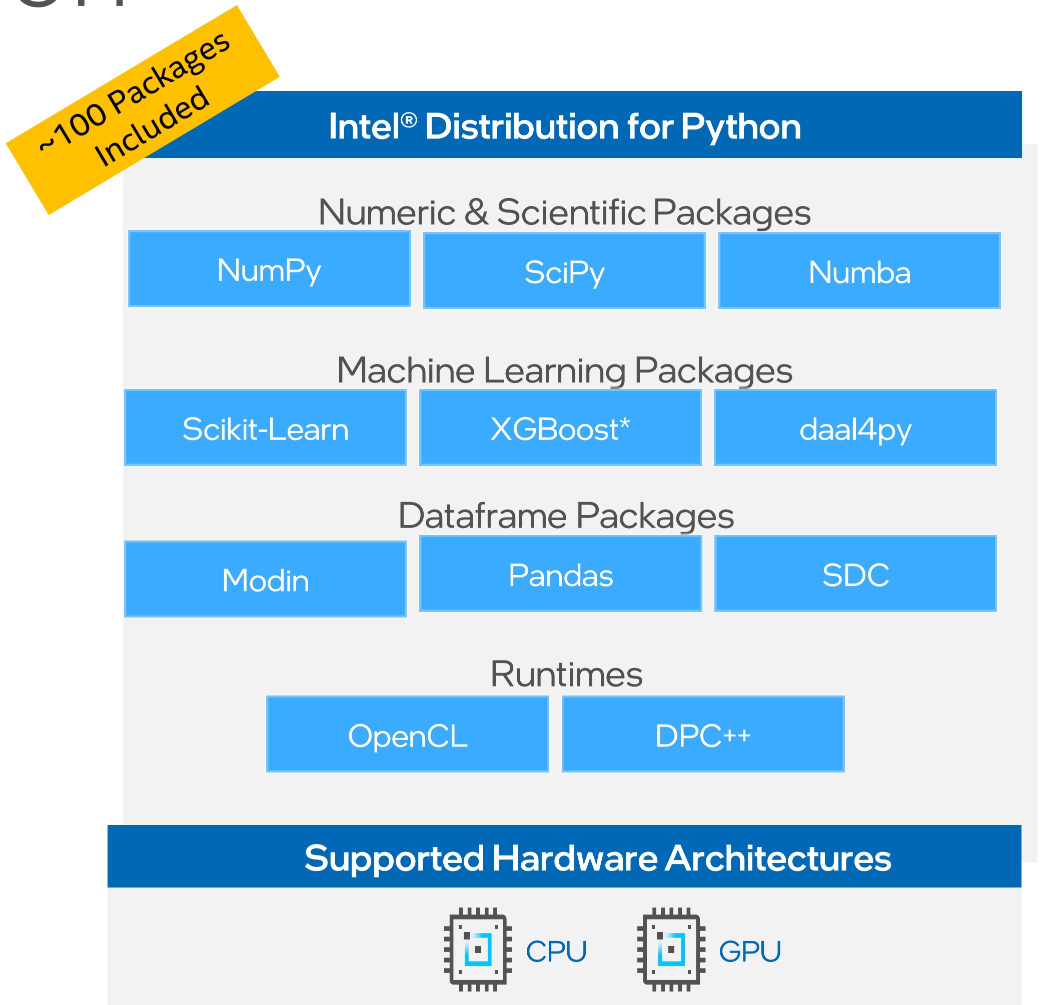
Intel® Distribution for Python

oneAPI Powered

- Develop fast, performant Python code with this set of essential computational packages

Who Uses It?

- Machine Learning Developers, Data Scientists, and Analysts can implement performance-packed, production-ready scikit-learn algorithms
- Numerical and Scientific Computing Developers can accelerate and scale the compute-intensive Python packages NumPy, SciPy, and mpi4py
- High-Performance Computing (HPC) Developers can unlock the power of modern hardware to speed up your Python applications



Hardware support varies by individual tool. Architecture support will be expanded over time.
Other names and brands may be claimed as the property of others.

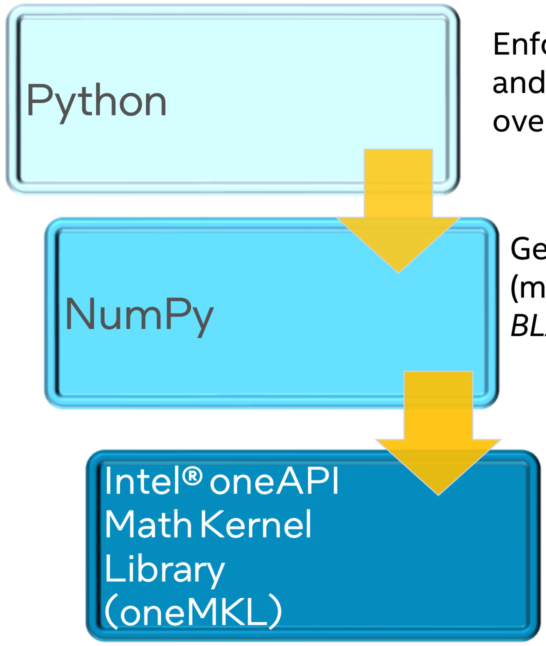
Intel Performance Optimization with NumPy and SciPy

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

Optimizations

- BLAS/LAPACK using oneMKL
- oneMKL-based FFT functionality
- Vectorized, threaded universal functions
- Use of Intel® C Compiler, and Intel® Fortran Compiler
- Aligned memory allocation
- Threaded memory copying



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

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

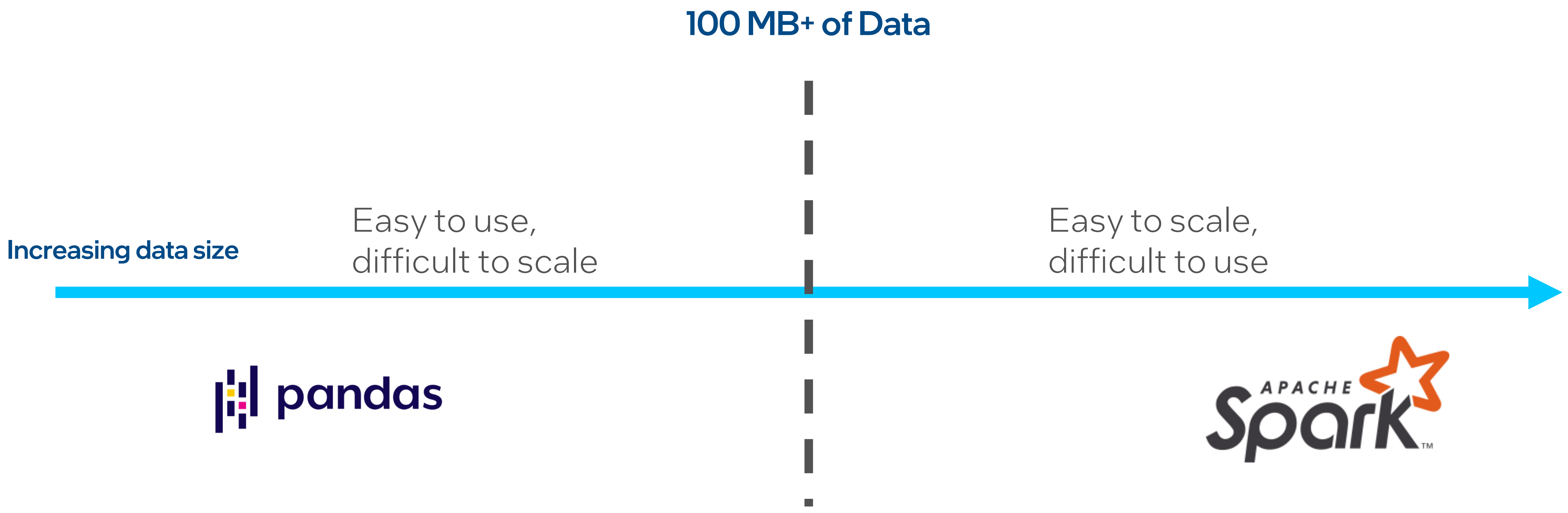
Gets around BLAS API bottleneck
Much stricter typing
Fastest performance level
Dispatches to hardware vectorization

oneMKL is 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

MODIN

Easy to use,
Easy to scale

Increasing data size

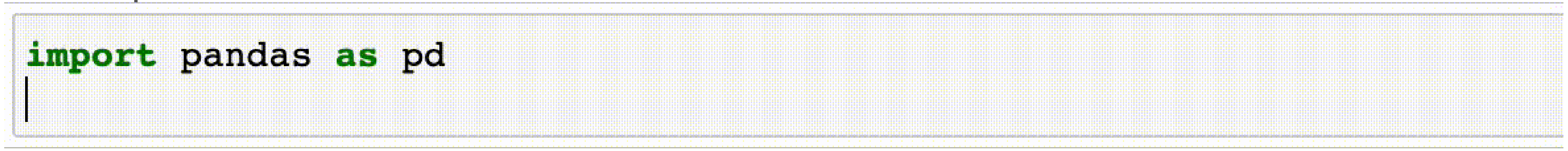
0-1TB+ of Data

pandas

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

Intel Distribution of Modin

- Accelerate your Pandas workloads across multiple cores and multiple nodes

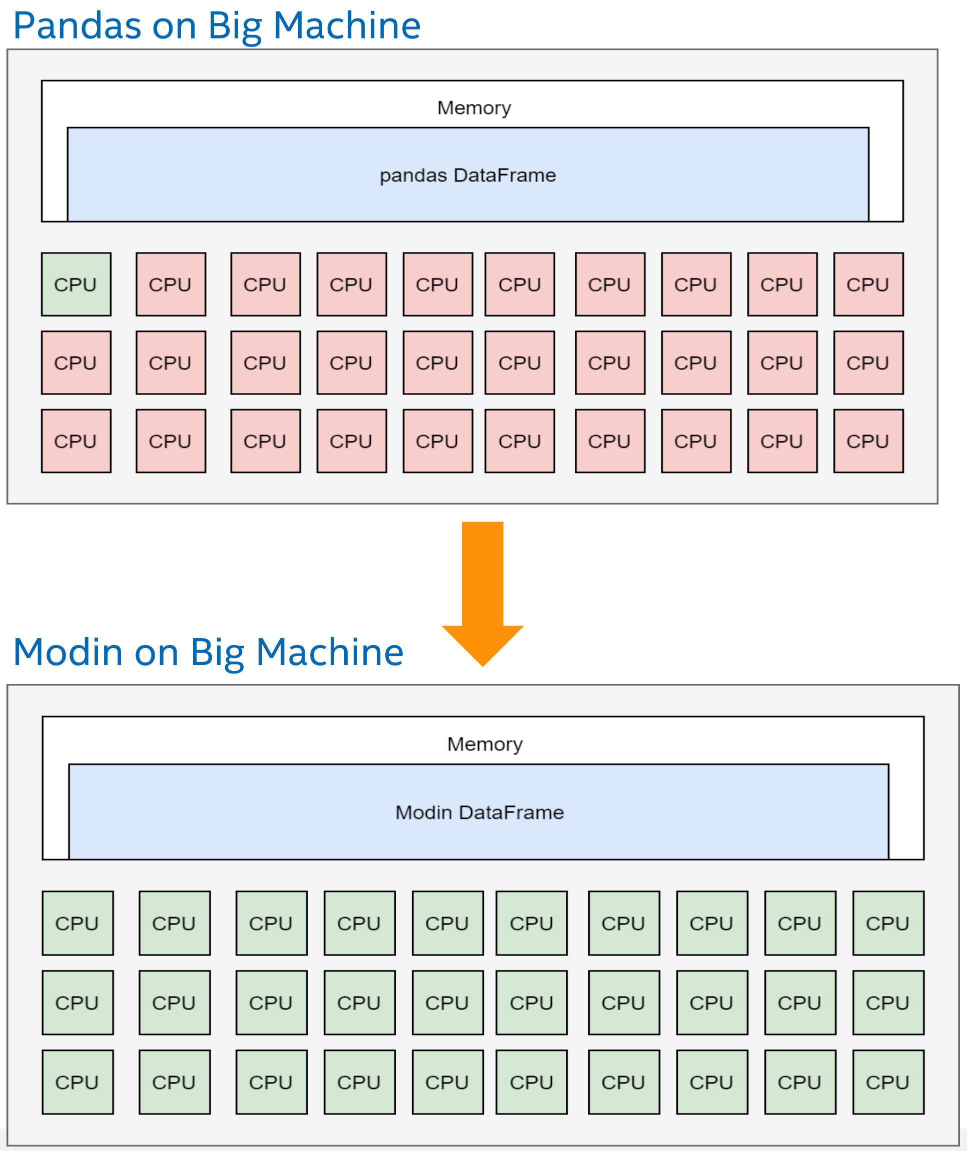


```
import pandas as pd
```

- 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!)

Intel Distribution of Modin

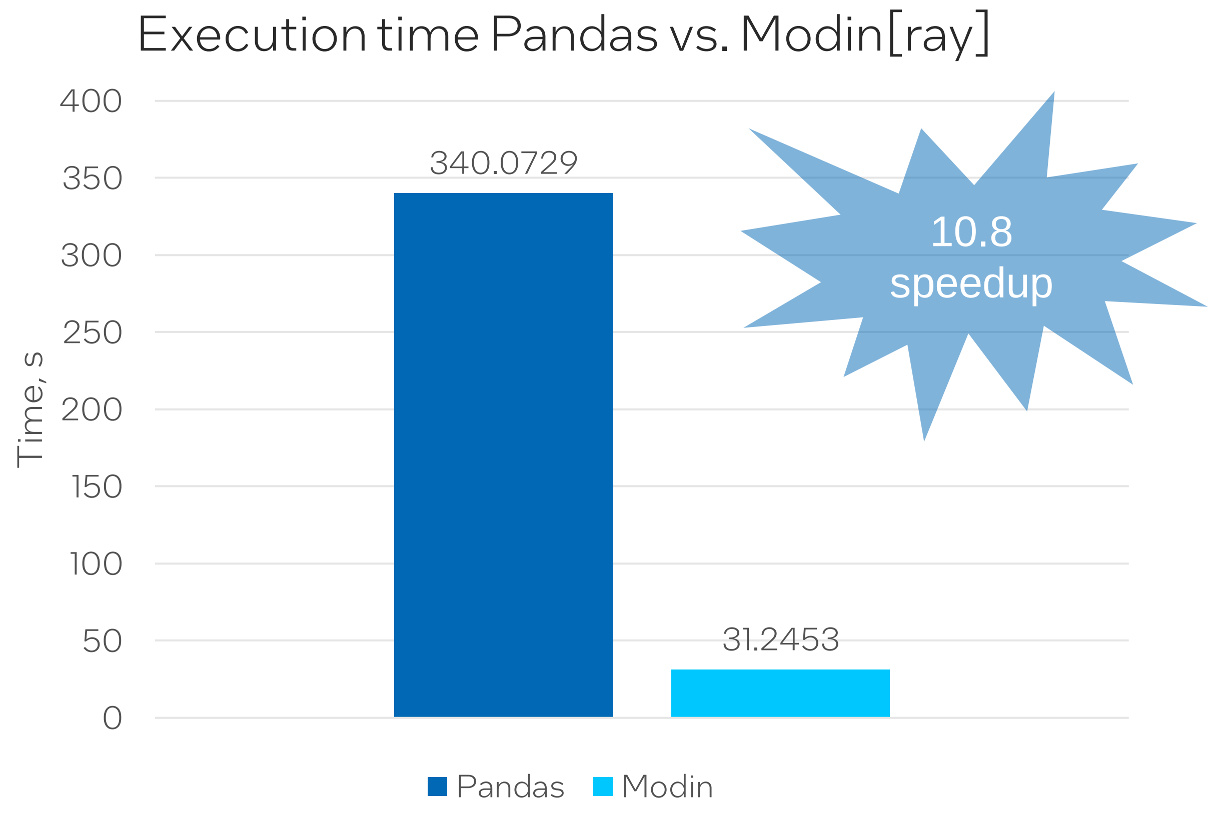
- Modin transparently **distributes the data and computation across available cores**, unlike Pandas which only uses one core at a time
- To use Modin, **you do not need to know how many cores your system** has, and you do not need to specify how to distribute the data



Modin

```
import modin.pandas as pd
import numpy as np


def run_etl():

    def cat_converter(x):
        if x is '':
            return np.int32(0)
        else:
            return np.int32(int(x, 16))

    names = [f"column_{i}" for i in range(40)]
    converter= {names[i]: cat_converter for i in range(14, 40)}

    df = pd.read_csv('data.csv', delimiter='\t', names=names,
                     converters=converter)

    count_y = df.groupby("column_0")["0"].count()

    return df, count_y


df, count_y = run_etl()
```

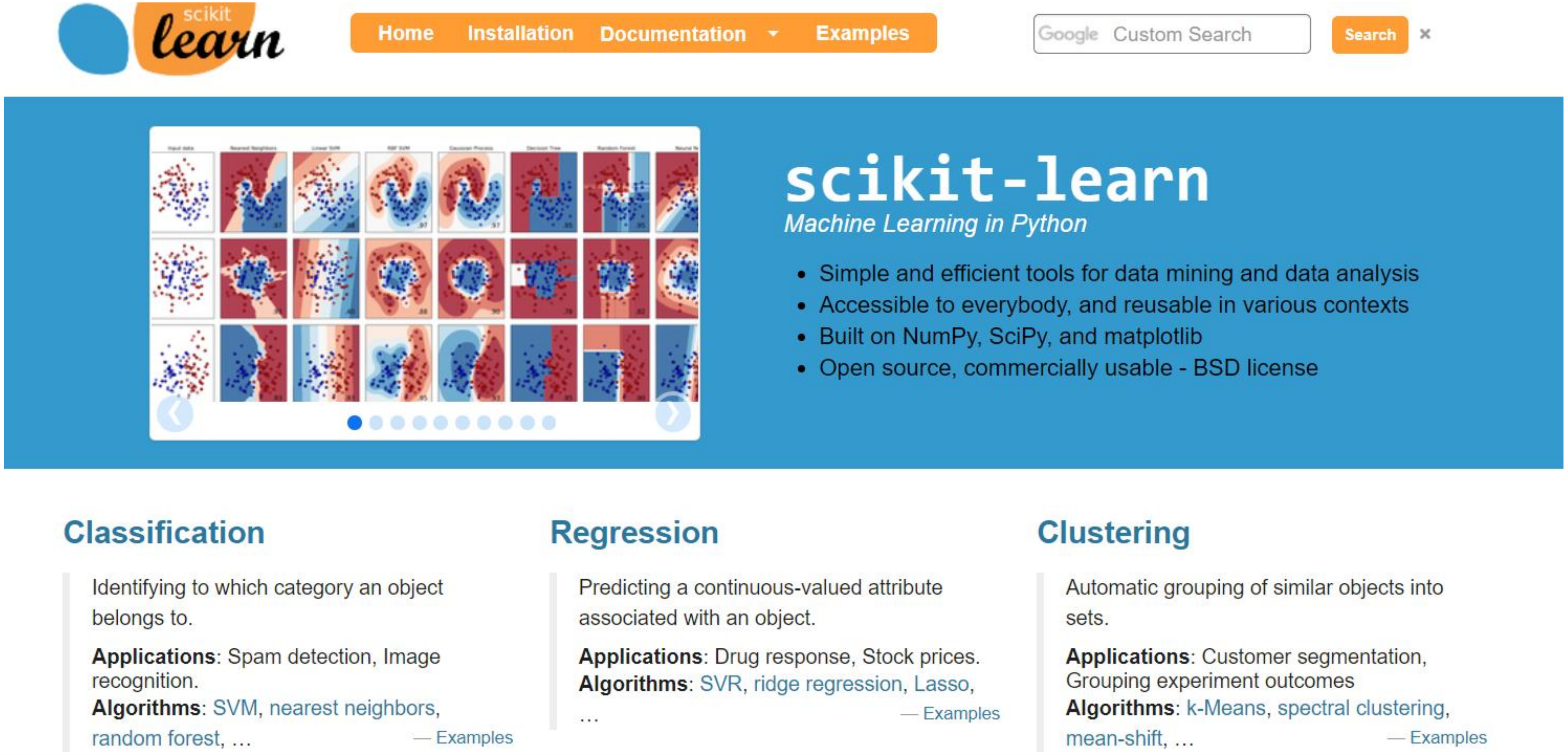
- Dataset size: 2.4GB



Intel® Xeon™ Gold 6248 CPU @ 2.50GHz, 2x20 cores

Intel® Extension for Scikit-Learn

The Most Popular Package for Python



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 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)
```

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

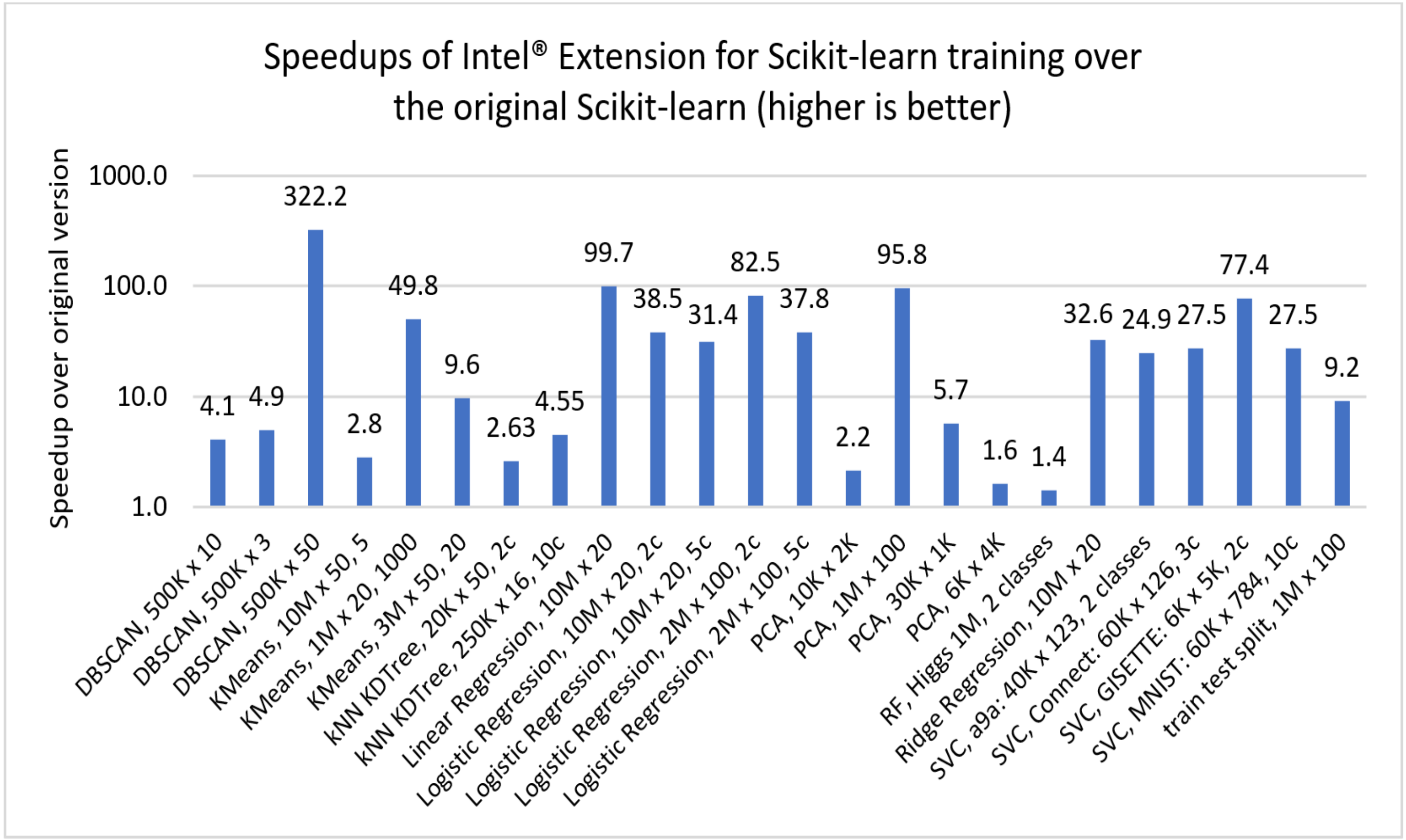
Same Code,
Same Behavior

✓ PASSED

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

Intel Extension for Scikit-Learn

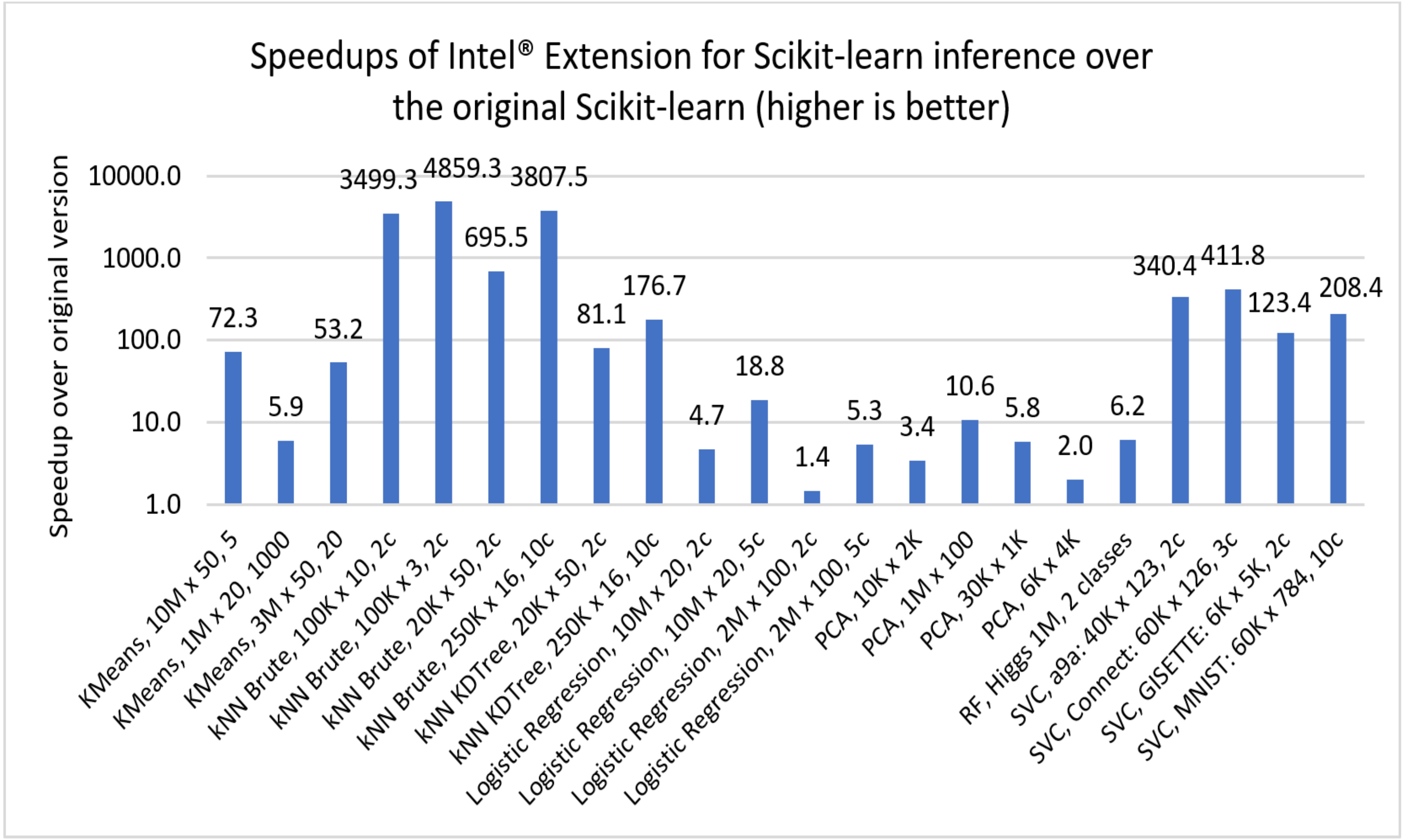
Performance on CLX compared to original Scikit-Learn : Training



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

Intel Extension for Scikit-Learn

Performance on CLX compared to original Scikit-Learn : Inference



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

XGBoost Optimizations

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® oneAPI Data Analytics Library (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 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

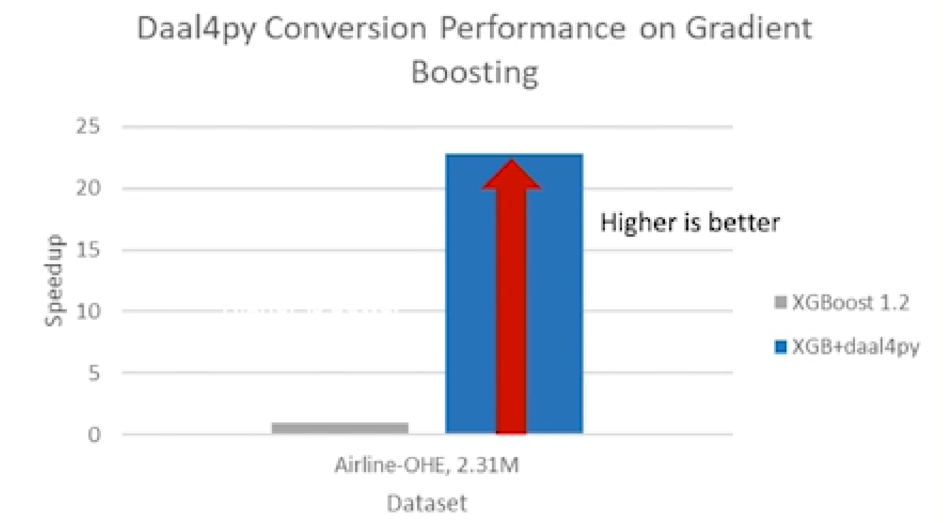
Legend:	Moved from oneDAL to XGBoost (v1.3)	Already available in oneDAL, potential optimizations for XGBoost

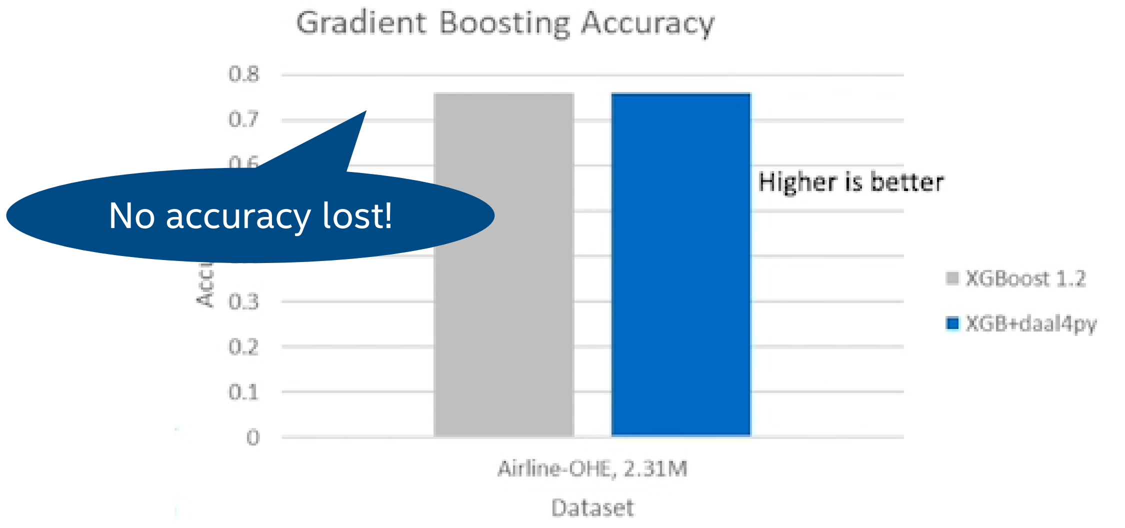
XGBoost and LightGBM Prediction Acceleration with Daal4Py

- 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





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

Demos

github.com/oneapi-src/oneAPI-samples/tree/master/AI-and-Analytics

Takeaways

- Intel AI Analytics Toolkit offers tools to accelerate each stage of your pipeline: IDP, Modin, Intel Extension for Scikit-learn and XGBoost.
- Drop-in replacements with minimal code changes to speed up your development.
- Packages easily available through Conda, pip, Docker.

Q n A