Choose the Best Accelerated Technology

Intel AI Analytics Toolkit – Classical ML

LRZ AI Workshop

Roy Allela – 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.
You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.
The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.
No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that a) you may publish an unmodified copy and b) code included in this document is licensed subject to the Zero-Clause BSD open source license (0BSD), https://opensource.org/licenses/0BSD. You may create software implementations based on this document and in compliance with the foregoing that are intended to execute on the Intel product(s) referenced in this document. No rights are granted to create modifications or derivatives of this document.
No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that code included in this document is licensed subject to the Zero-Clause BSD open source license (OBSD), http://opensource.org/licenses/0BSD.
No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.
Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.

Agenda

- Intel AI Analytics Toolkit
- Intel Distribution for Python
- Intel Distribution of Modin
- Intel(R) Extension for Scikit-learn
- XGBoost Optimizations

Intel® AI Analytics Toolkit

Powered by oneAPI

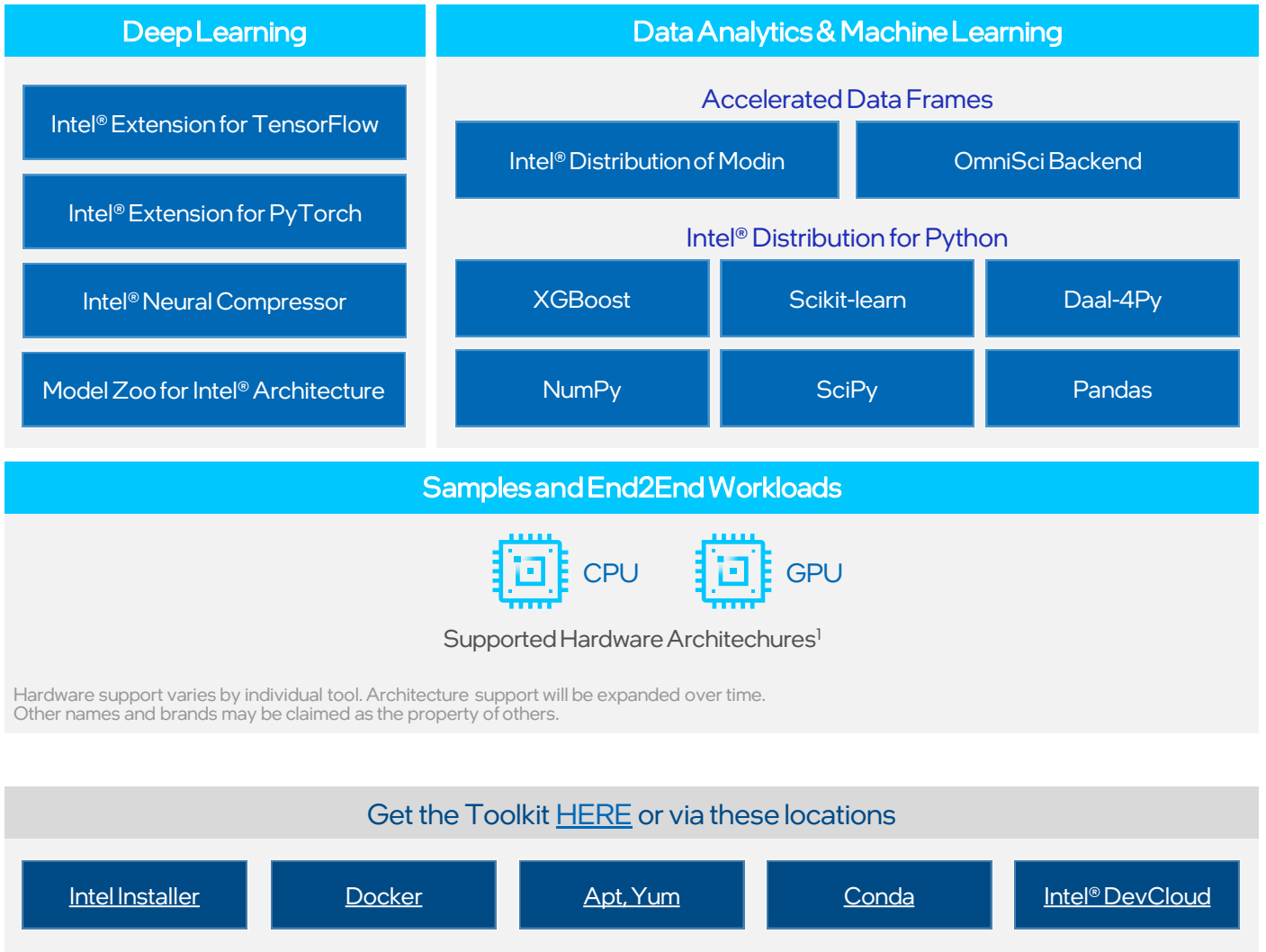
Accelerate end-to-end AI and data analytics pipelines with libraries optimized for Intel® architectures

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

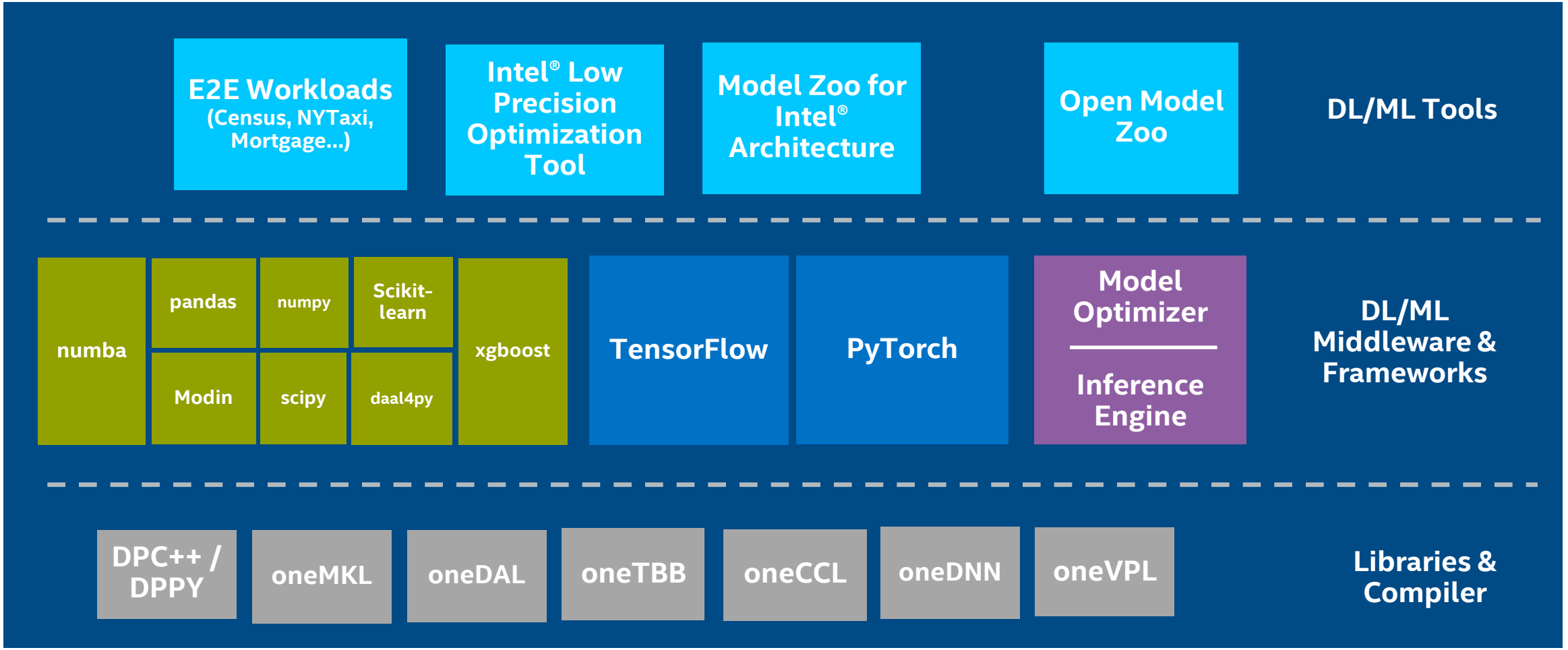


Learn More: software.intel.com/oneapi/ai-kit

Back to Domain-specific Toolkits for Specialized Workloads

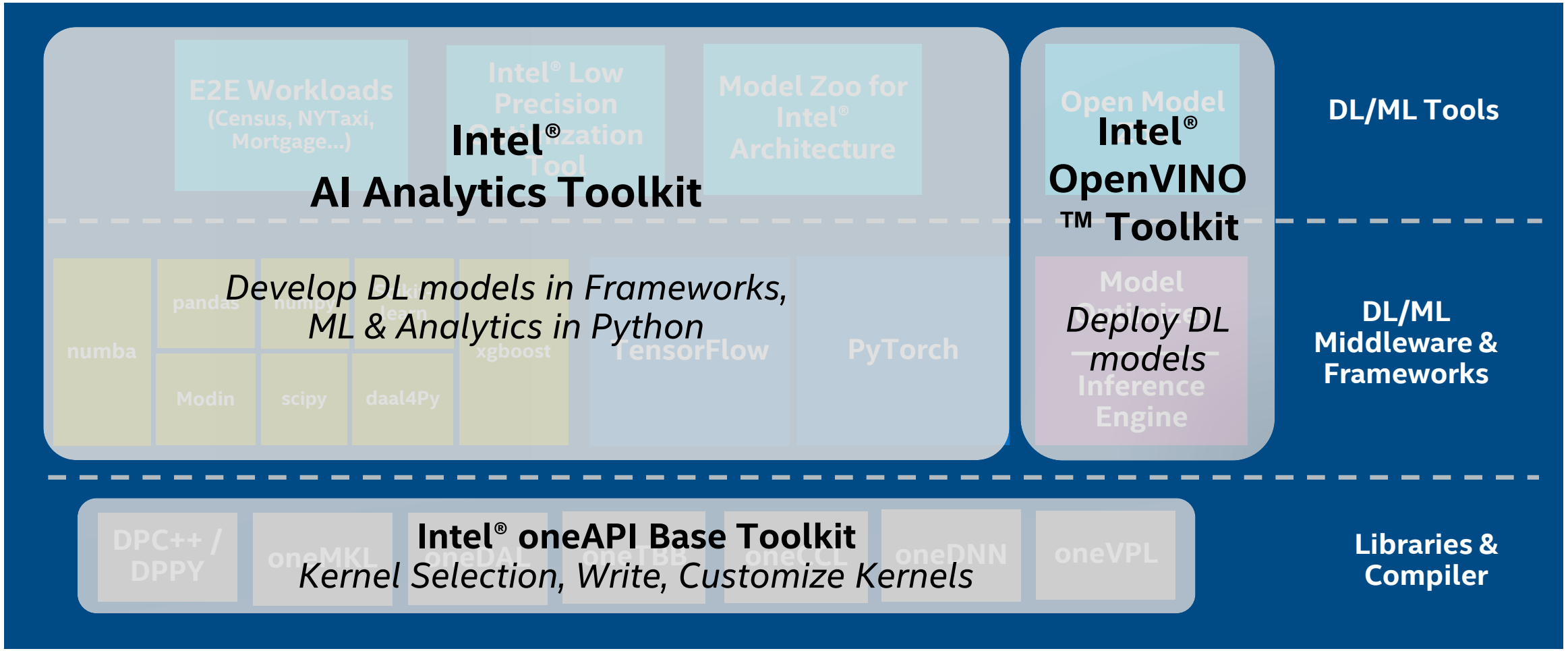
AI Software Stack for Intel® XPUs

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



AI Software Stack for Intel® XPUs

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

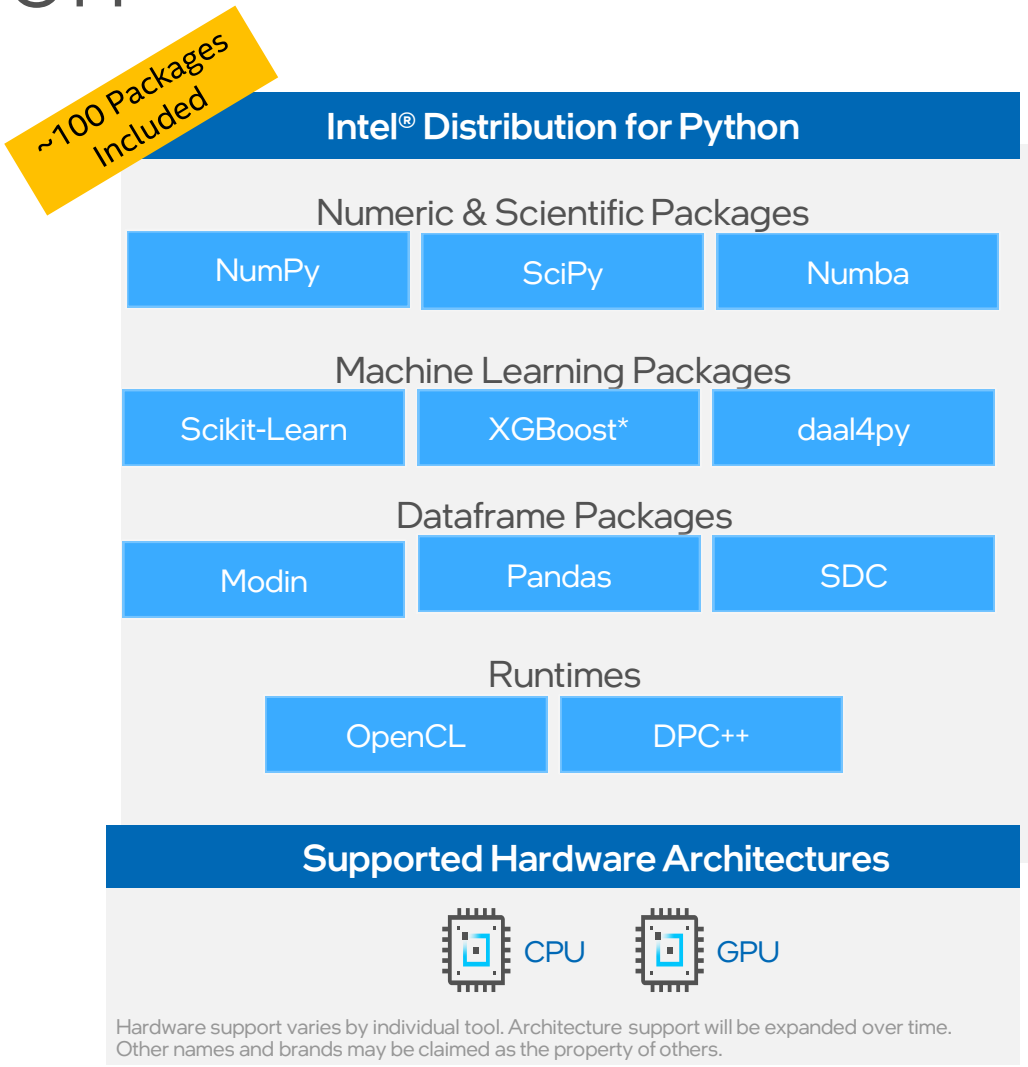


Full Set of AI ML and DL Software Solutions Delivered with Intel's oneAPI Ecosystem

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

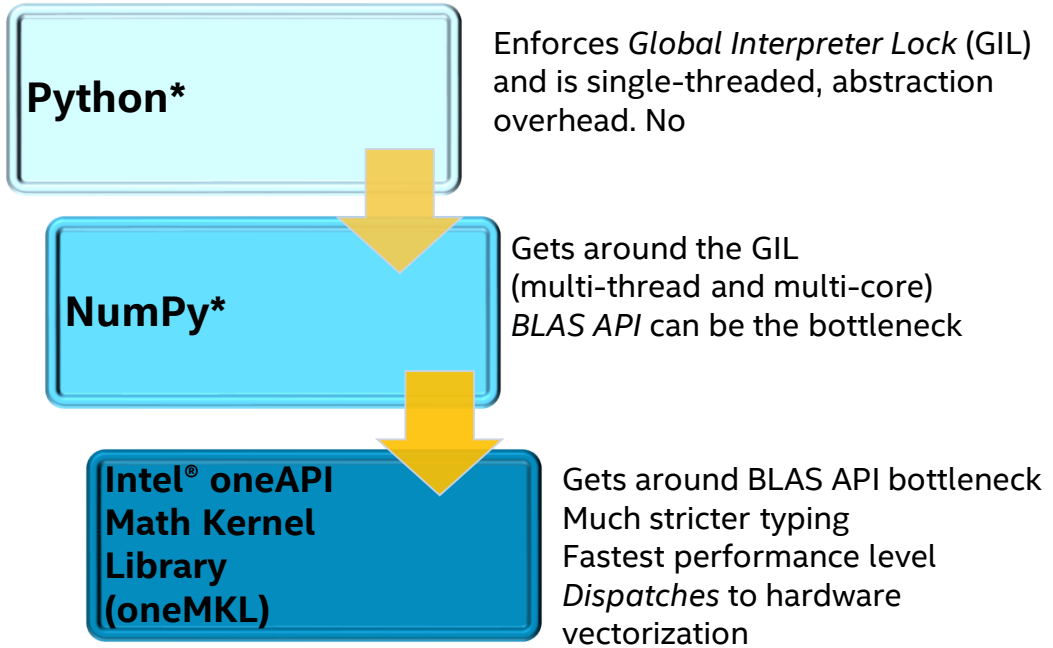
Initial GPU support enabled with Data Parallel Python

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

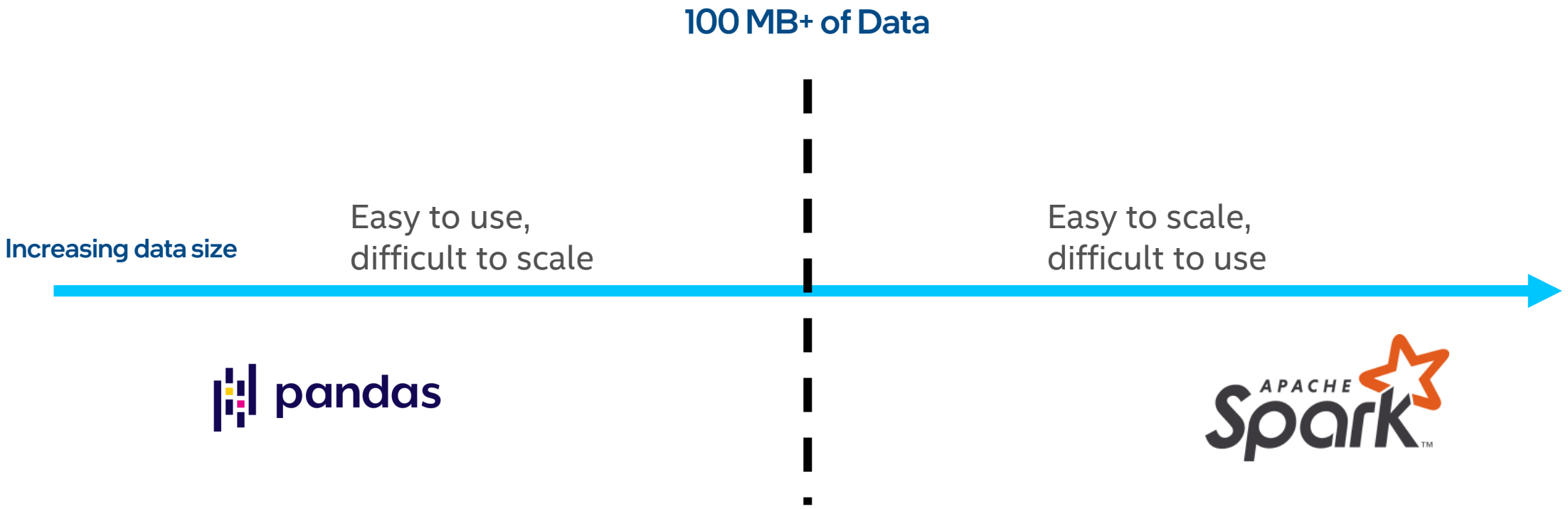


Intel® oneMKL included with Anaconda* standard bundle; is Free for Python*

Intel® Modin Library

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

Intel distribution of Modin

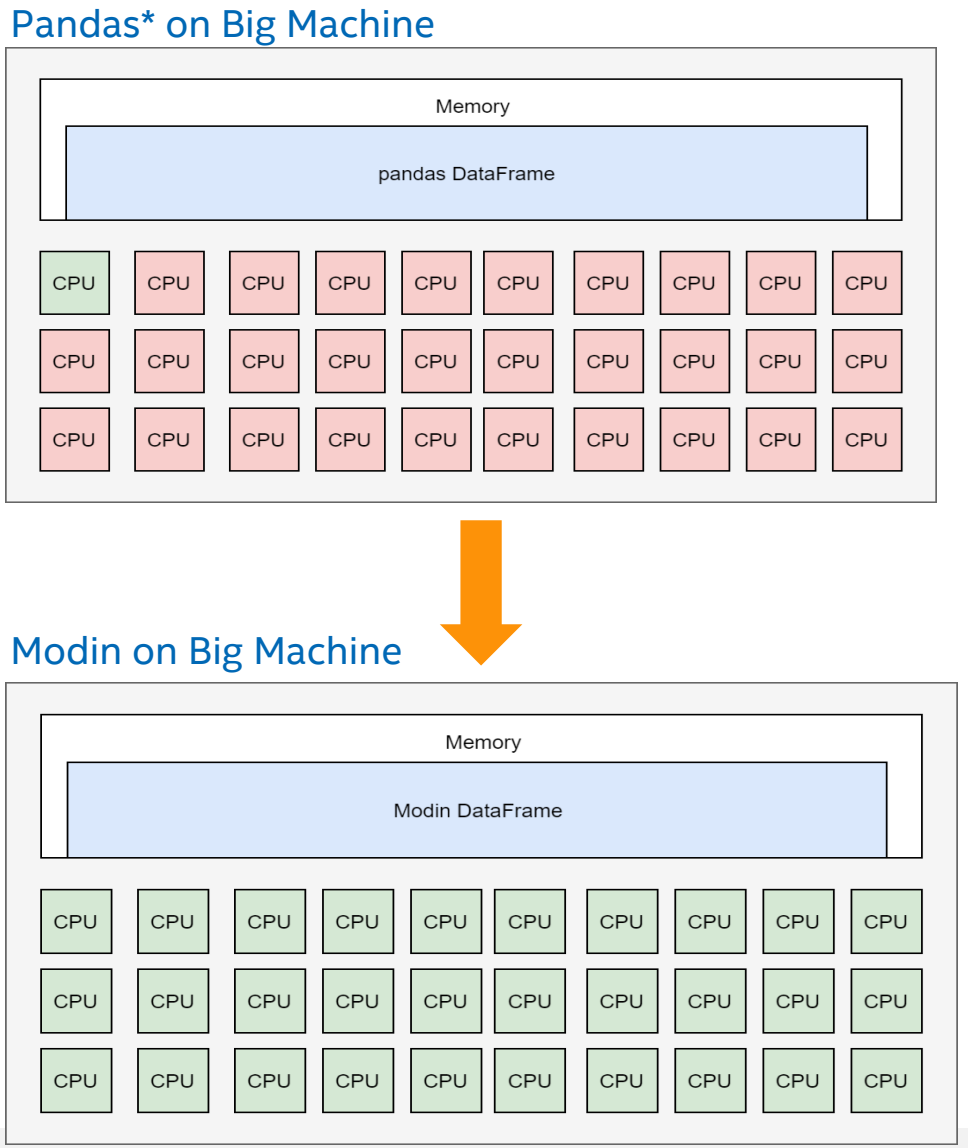
- Accelerate your Pandas* workloads across multiple cores and multiple nodes

```
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 laptop!)

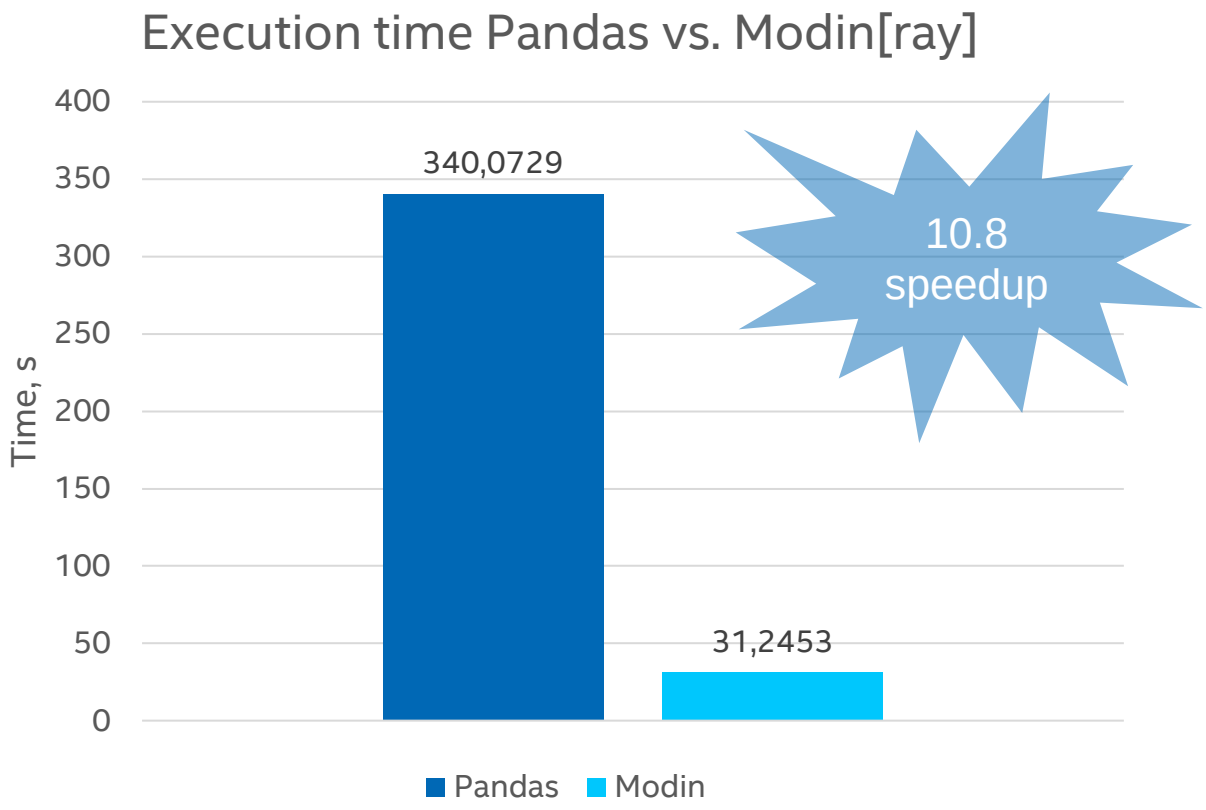
Intel distribution of Modin

- 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()
```



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

- Dataset size: 2.4GB

Demo

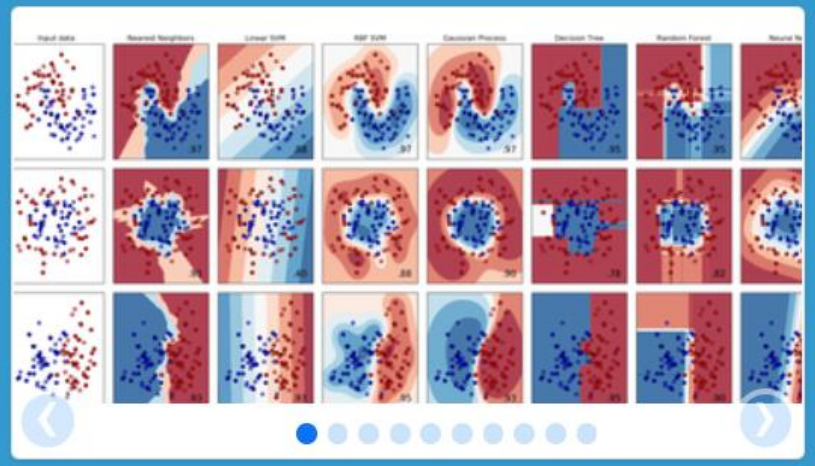
Intel® Extension for Scikit-Learn

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

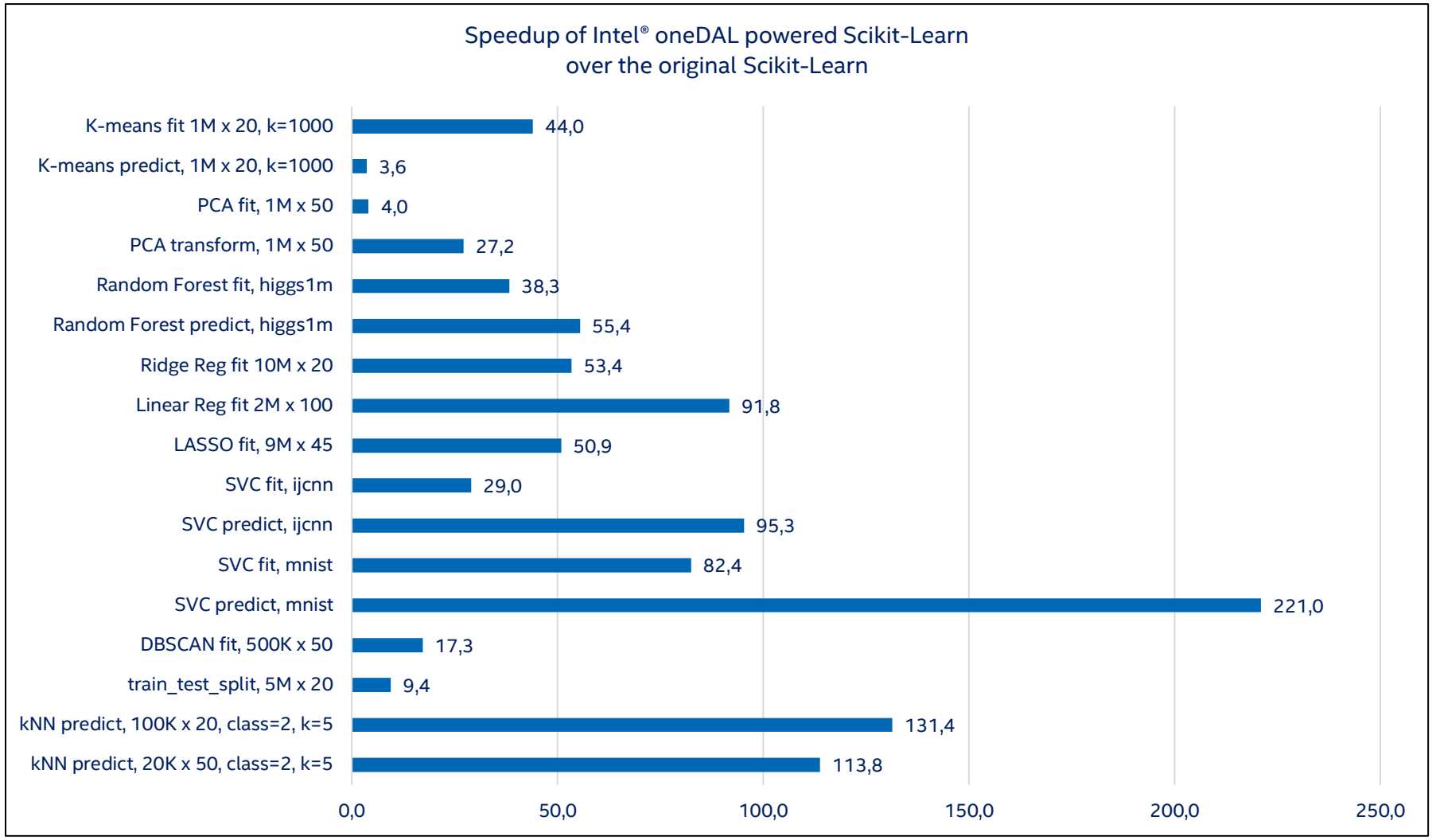
Available through:

- 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

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 - DAAL 2020.3
  - kNN (kd-tree and brute force) - DAAL 2020.3

Intel optimized Scikit-Learn



HW: Intel Xeon Platinum 8276L CPU @ 2.20GHz, 2 sockets, 28 cores per socket;
Details: https://medium.com/intel-analytics-software/accelerate-your-scikit-learn-applications-a06cacf44912

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

Demo

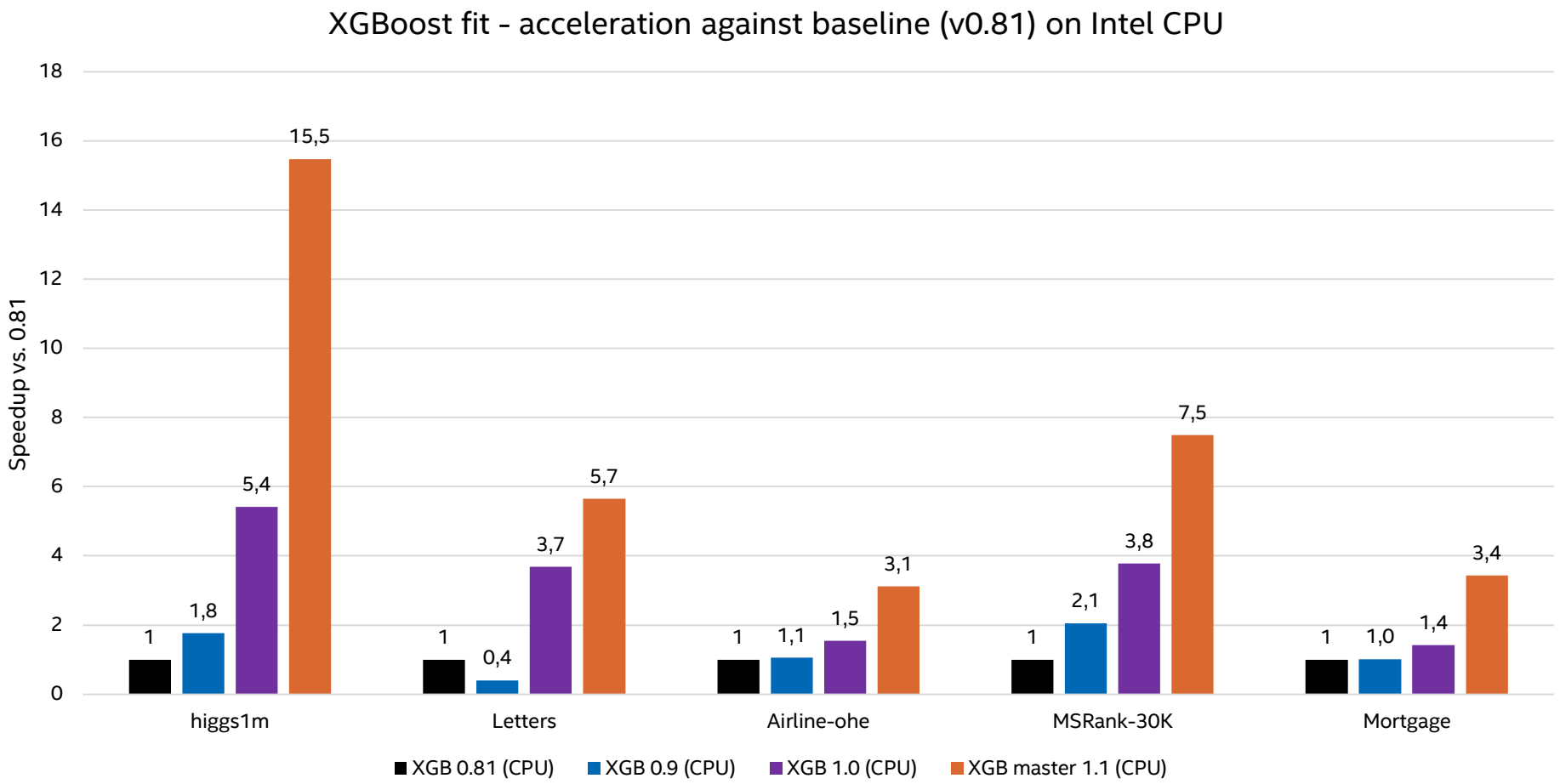
XGBoost Library

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, …

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)

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:

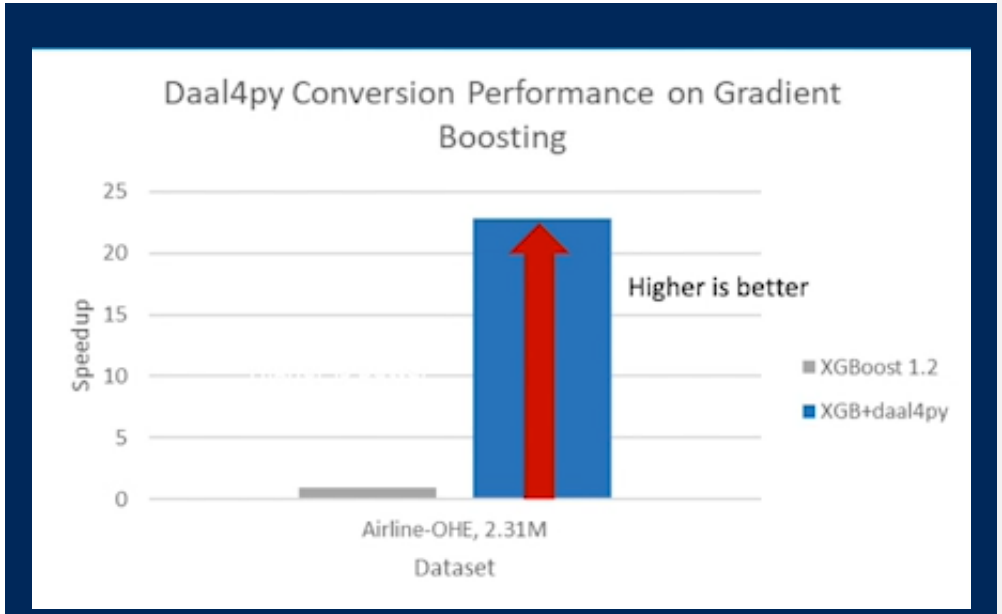
- Moved from Intel® oneDAL to XGBoost (v1.3)
- Already available in Intel® oneDAL, potential optimizations for XGBoost*

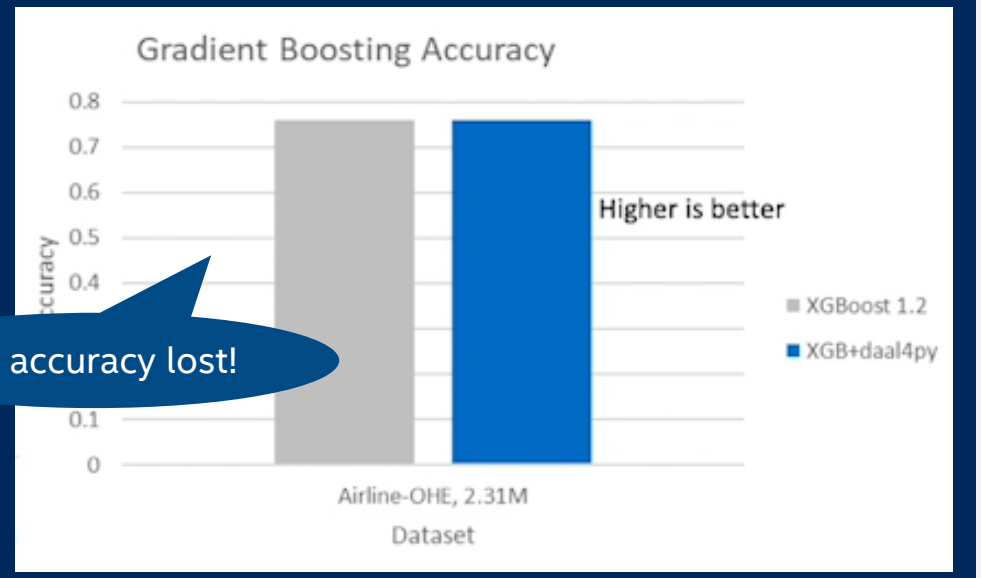
XGBoost* and LightGBM* Prediction Acceleration with Daal4Py

- 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.

Demo

QnA