Choose the Best Accelerated Technology

Intel Software Optimizations for Classical Machine Learning

Roy Allela– Deep Learning SW Engineer
roy.m.allela@intel.com

intel®

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

Deep Learning

- Intel® Optimization for TensorFlow
- Intel® Optimization for PyTorch
- Intel® Low Precision Optimization Tool
- Model Zoo for Intel® Architecture

Data Analytics & Machine Learning

Accelerated Data Frames

- Intel® Distribution of Modin
- OmniSci Backend

Intel® Distribution for Python

- XGBoost
- Scikit-learn
- Daal-4Py
- NumPy
- SciPy
- Pandas

Samples and End2End Workloads

CPU GPU

Supported Hardware Architectures[1]

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.

Get the Toolkit HERE or via these locations

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

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

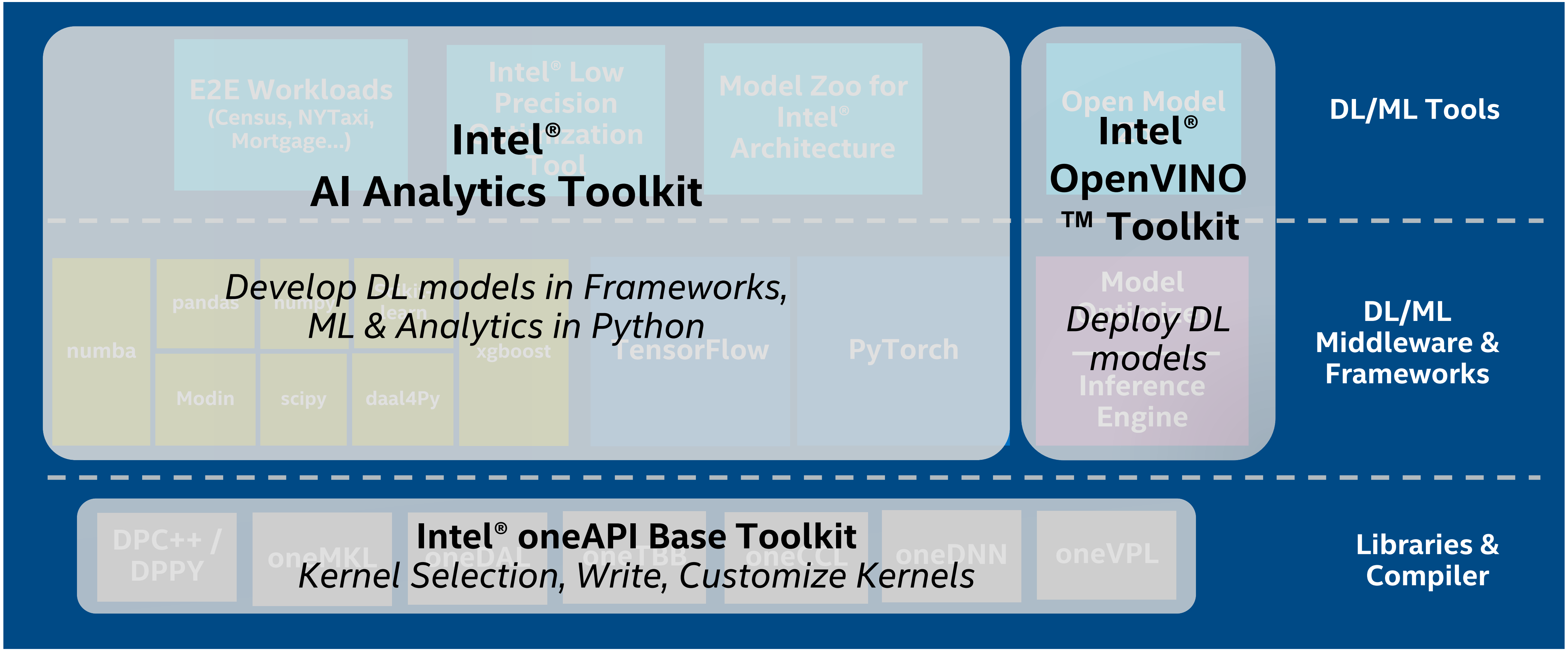
DL/ML Tools
- E2E Workloads (Census, NYTaxi, Mortgage…)
- Intel® Low Precision Optimization Tool
- Model Zoo for Intel® Architecture
- Open Model Zoo

DL/ML Middleware & Frameworks
- numba
- pandas
- numpy
- Scikit-learn
- Modin
- scipy
- daal4py
- xgboost
- TensorFlow
- PyTorch
- Model Optimizer / Inference Engine

Libraries & Compiler
- DPC++ / DPPY
- oneMKL
- oneDAL
- oneTBB
- oneCCL
- oneDNN
- oneVPL

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

Executive Summary

- 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)
- Access the latest vectorization and multithreading instructions, Numba* and Cython*, composable parallelism with Threading Building Blocks, and more

- Analysts
- Data Scientists
- Machine Learning Developers

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

~100 Packages Included

Intel® Distribution for Python

Numeric & Scientific Packages

NumPy | SciPy | Numba

Machine Learning Packages

Scikit-Learn | XGBoost* | daal4py

Dataframe Packages

Modin | Pandas | SDC

Runtimes

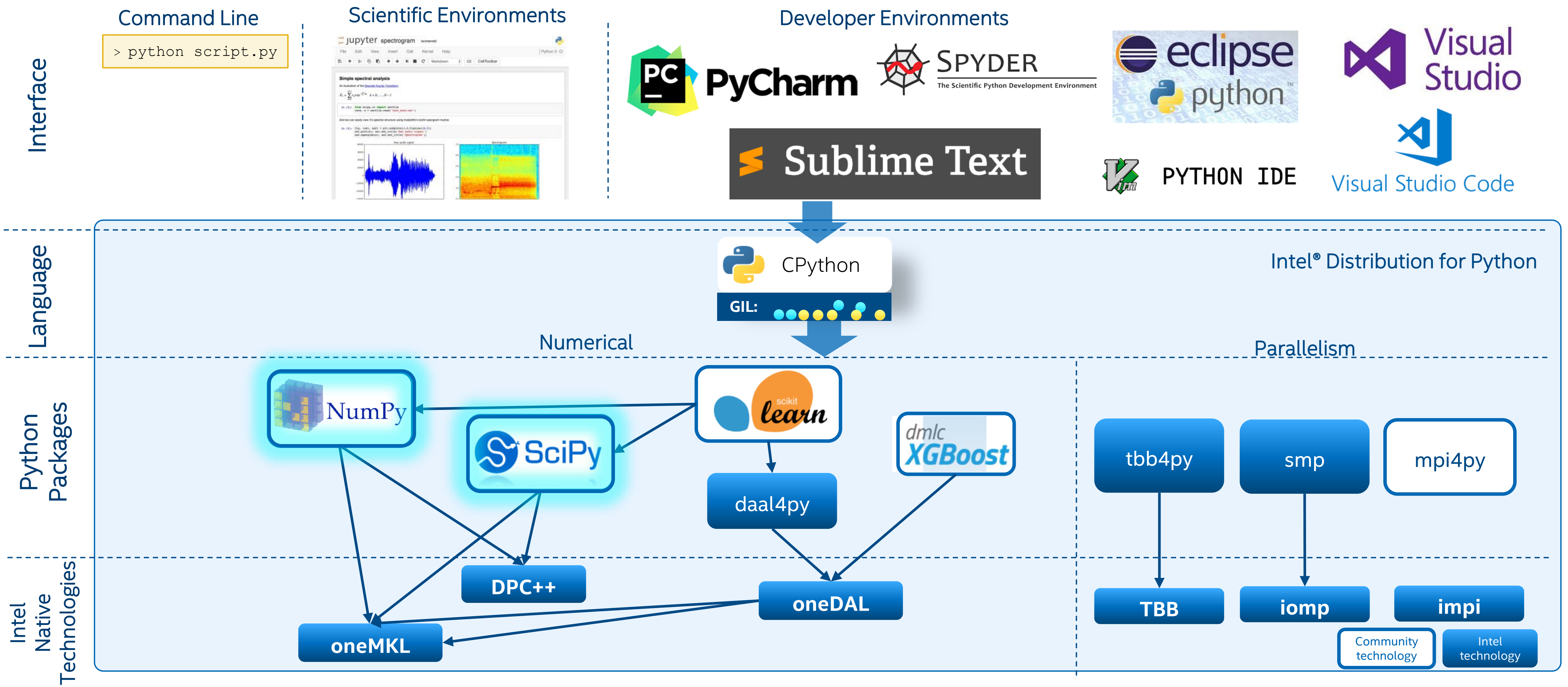
OpenCL | DPC++

Supported Hardware Architectures

CPU | GPU

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® Distribution for Python Architecture



Intel® Distribution for Python

Developer Benefits

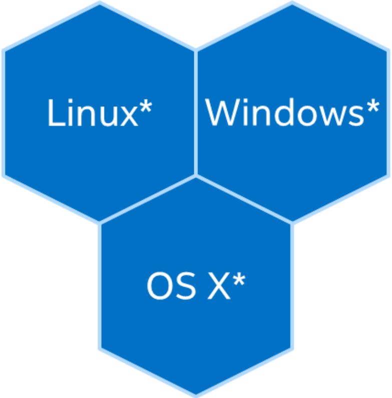
Maximize Performance	Minimize Development Cost	Vast Ecosystem
Performance Libraries, Parallelism, Multithreading, Language Extensions	**Drop-in Python Replacement**	**Familiar usage and compatibility**
Near-native performance comes through acceleration of core Python numerical packages Accelerated NumPy/SciPy/scikit-learn 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.

Choose Your Download Option

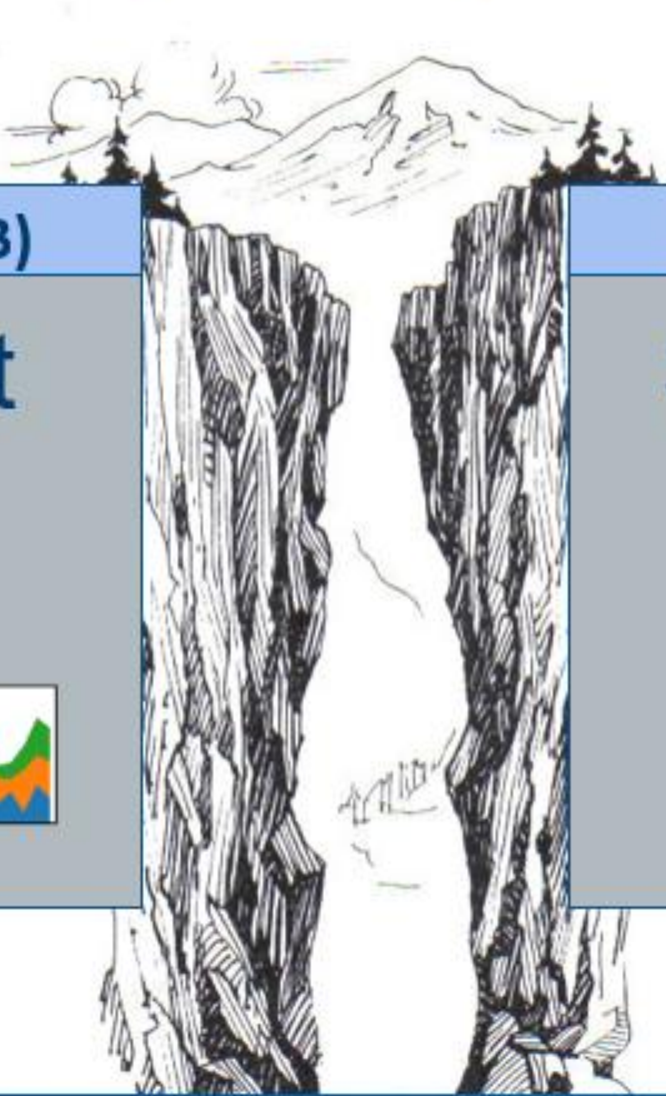
Python Solutions	Download Options
Tools and frameworks to accelerate end-to-end data science and analytics pipelines	Intel® AI Analytics Toolkit 
Develop fast, performant Python code with essential computational packages	Intel® Distribution for Python 
Optimized Python packages from package managers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



* Also available in the Intel® oneAPI Base Toolkit

Modin

Data Science Landscape: Today



Tools efficient for O(1MB)

Usable but not scalable

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

Tools efficient for O(100s GB+)

Scalable but not usable

APACHE Spark™

Data Science Landscape: Today

Tools efficient for O(1MB)

Usable and scalable

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

MODIN

Tools efficient for O(100s GB+)

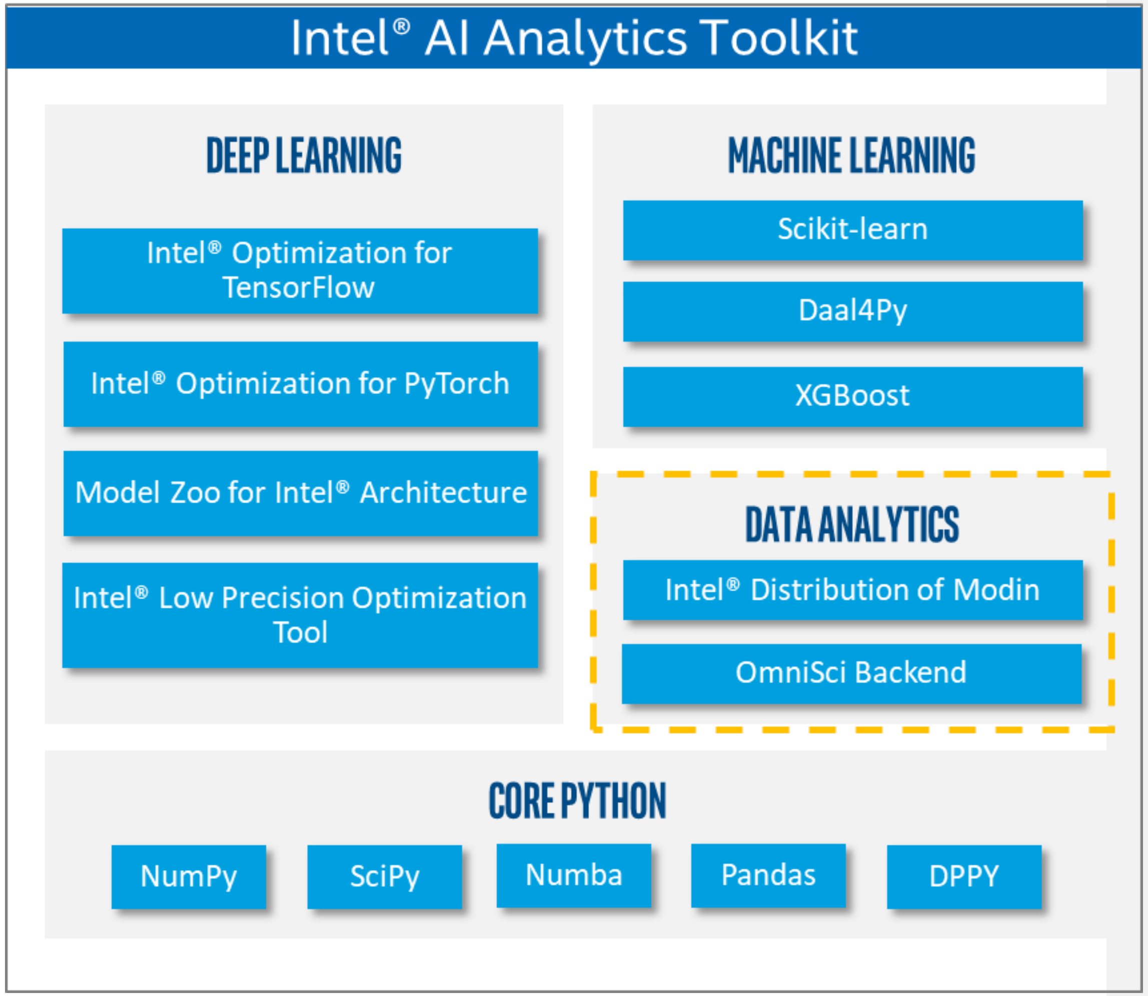
Scalable and usable

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

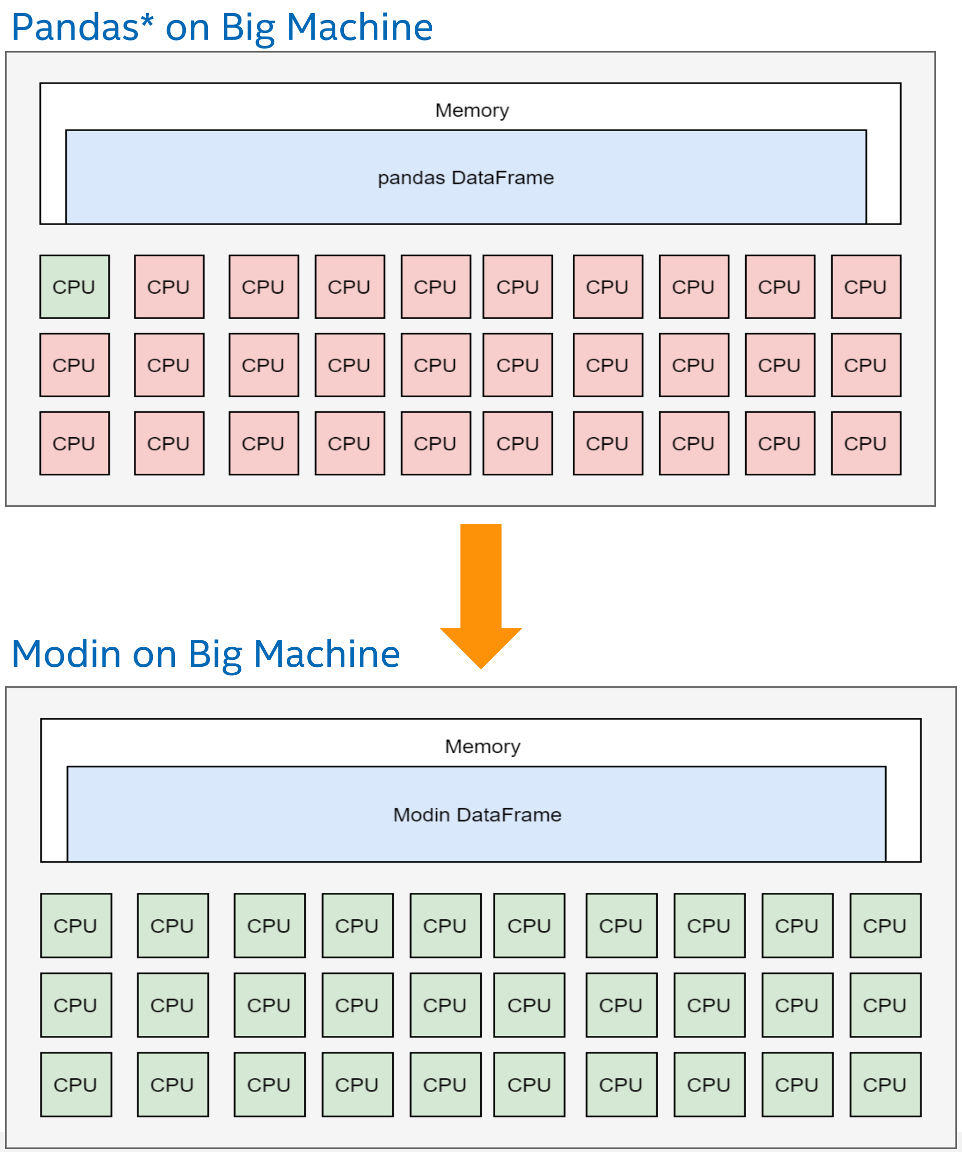
Intel distribution of Modin

- Accelerate your Pandas* workloads across multiple cores and multiple nodes
- No upfront cost to learning a new API
    - import modin.pandas as pd
- In the backend, Intel Distribution of Modin is supported by Omnisci*, a performant framework for end-to-end analytics that has been optimized to harness the computing power of existing and emerging Intel® hardware

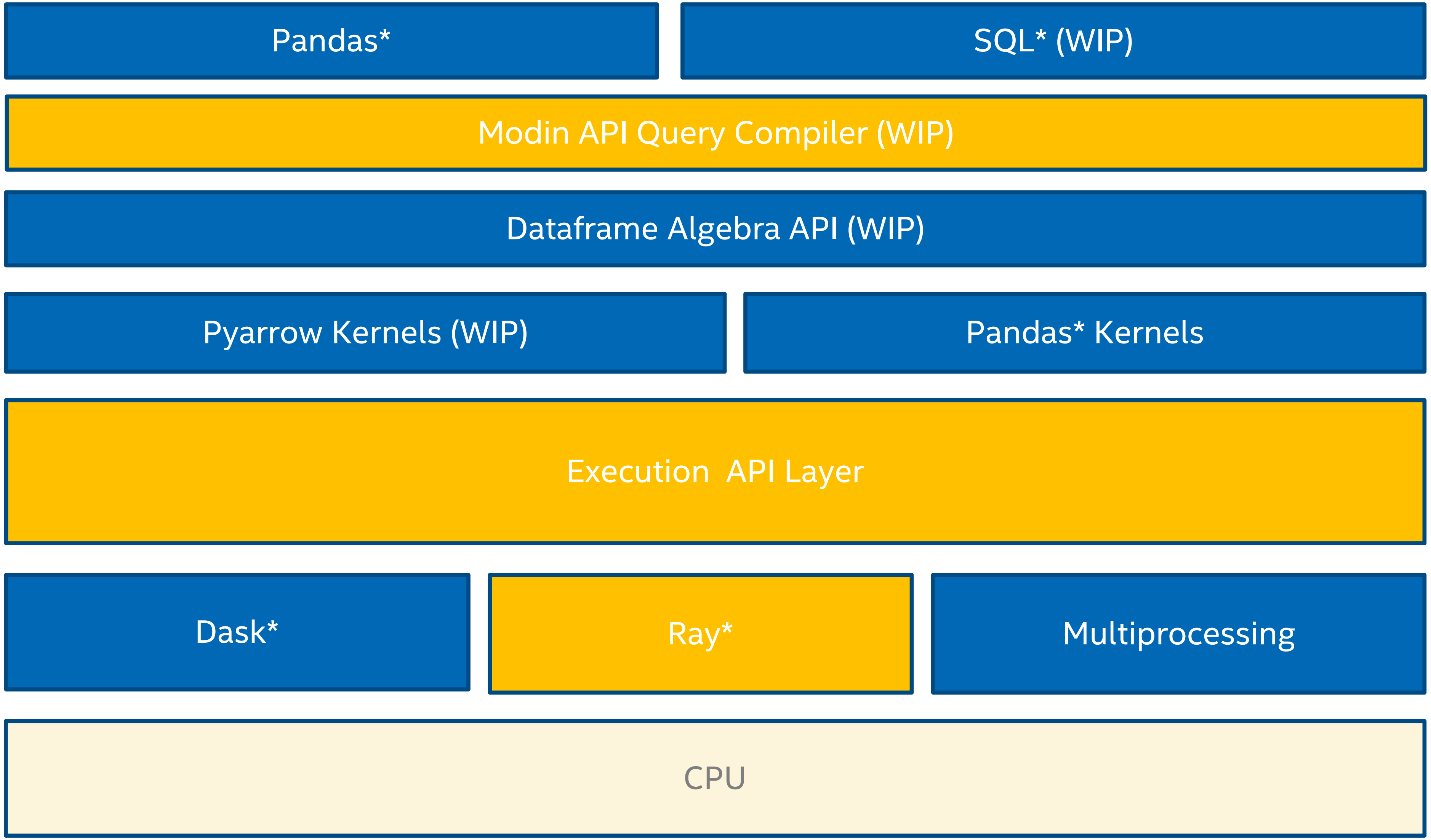


Intel distribution of Modin

- Recall: 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!)
- 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

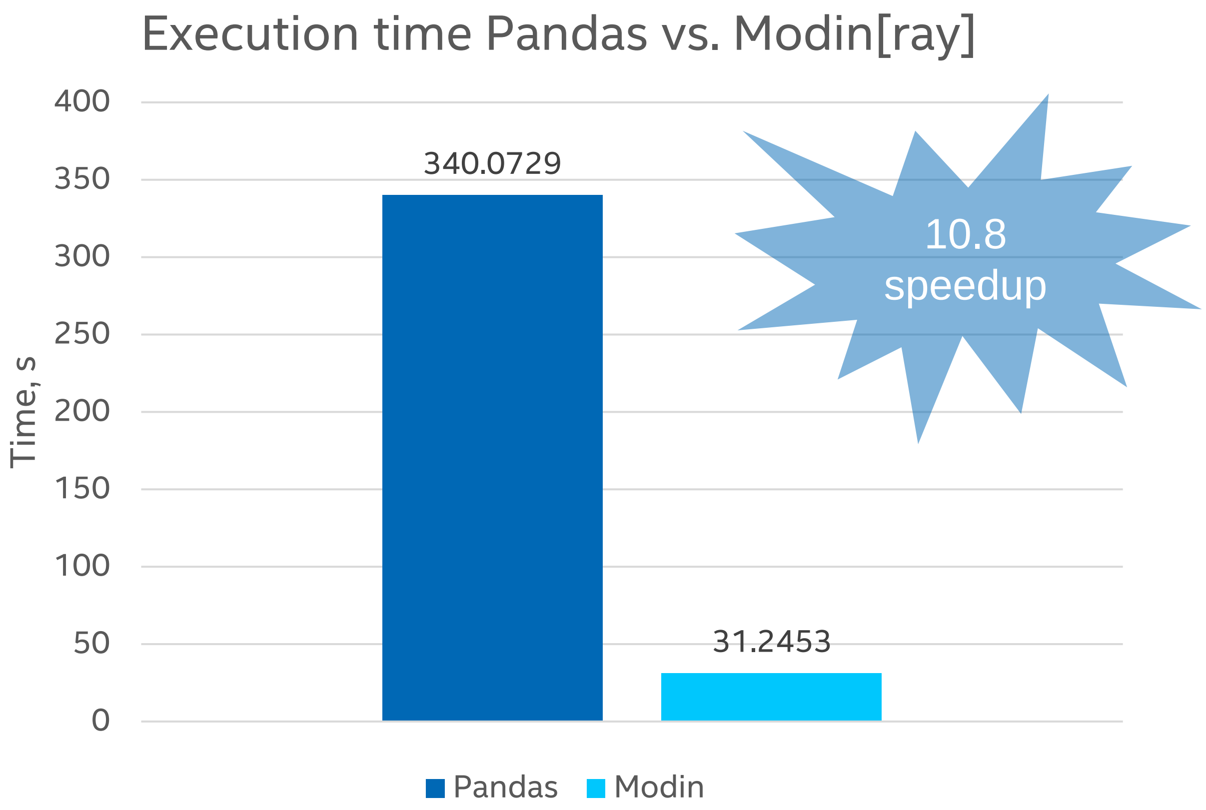


MODIN LAYERED ARCHITECTURAL DIAGRAM - NOW



Modin

```python
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()
```

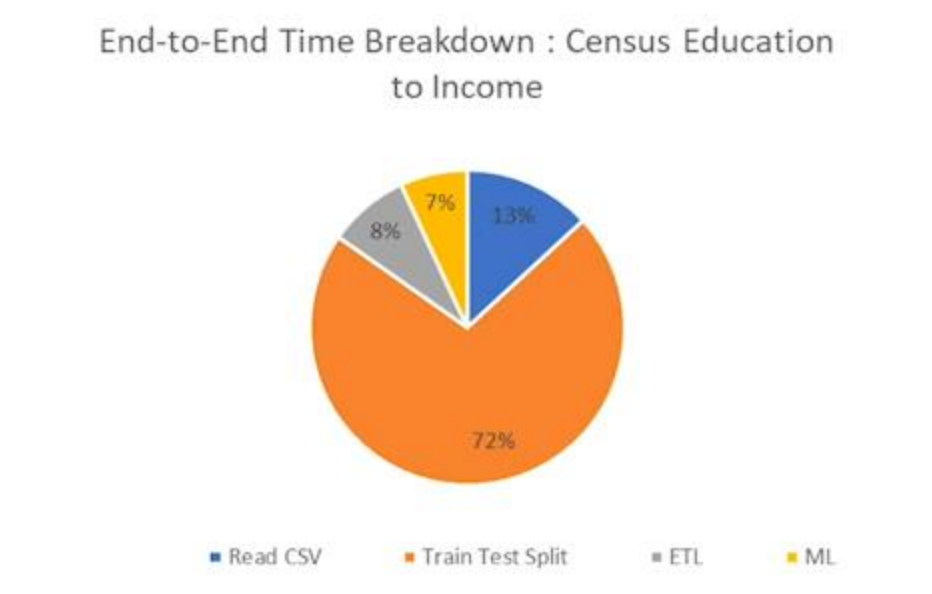


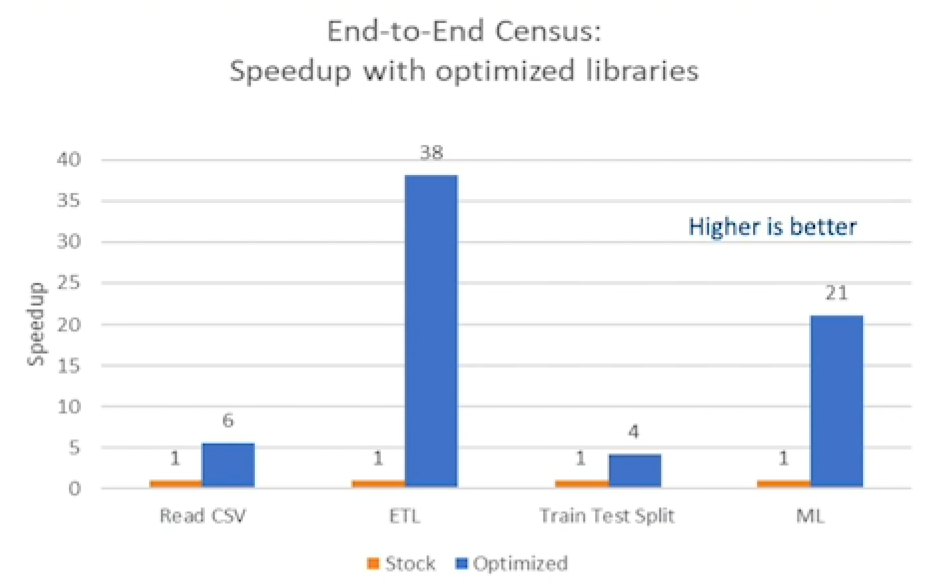
Intel® Xeon™ Gold 6248 CPU @ 2.50GHz, 2x20 cores

- Dataset size: 2.4GB

End-to-End Data Pipeline Acceleration

- **Workload:** Train a model using 50yrs of Census dataset from IPUMS.org to predict income based on education
- **Solution:** Intel Modin for data ingestion and ETL, Daal4Py and Intel scikit-learn for model training and prediction
- **Perf Gains:**
  - Read_CSV (Read from disk and store as a dataframe) : **6x**
  - ETL operations : **38x**
  - Train Test Split : **4x**
  - ML training (fit & predict) with Ridge Regression : **21x**





Demo

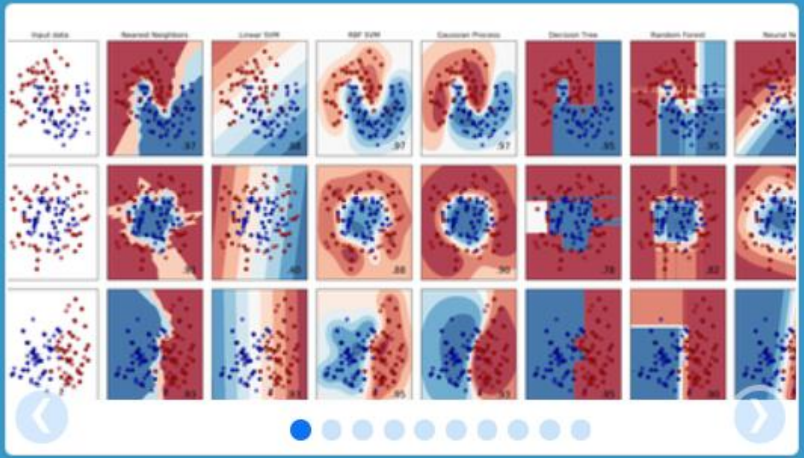
Intel(R) Extension for Scikit-learn

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

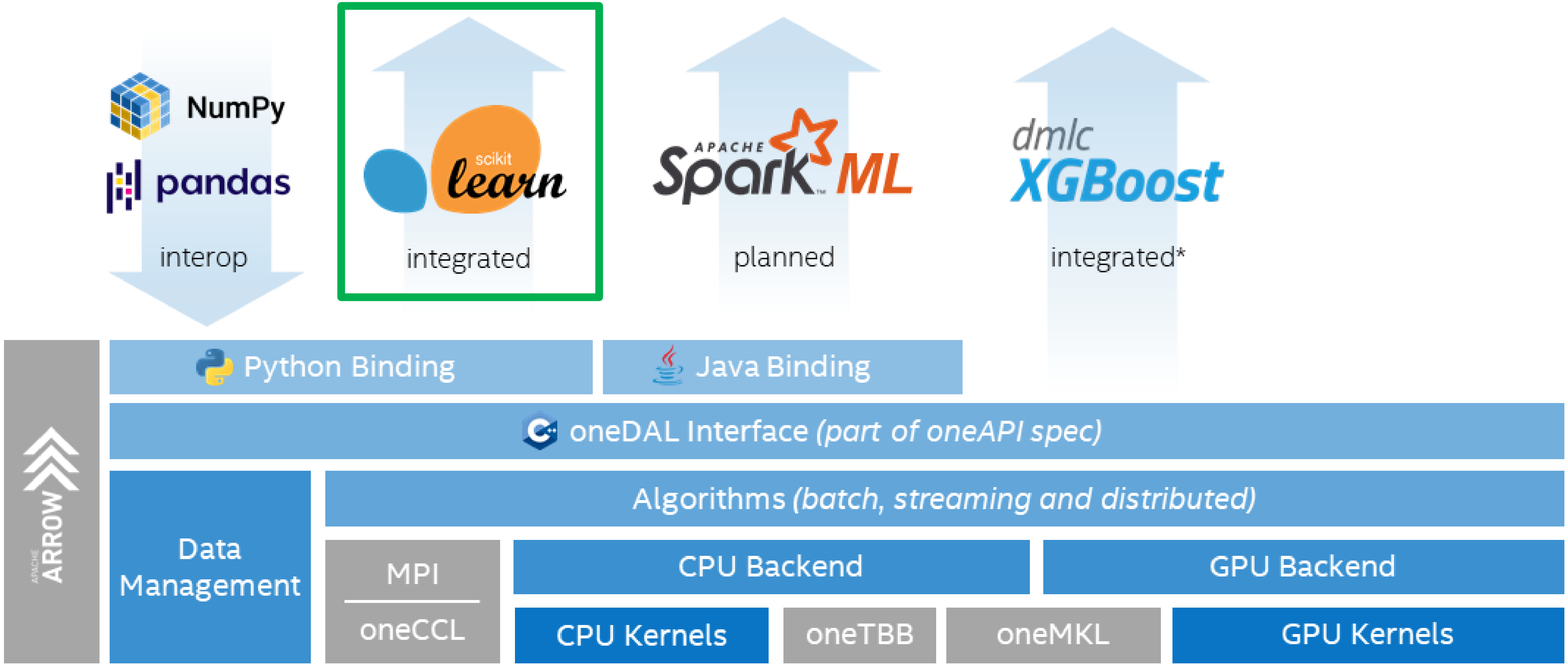
Automatic grouping of similar objects into sets.

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

— Examples

oneAPI Data Analytics Library (oneDAL)

Optimized building blocks for all stages of data analytics on Intel Architecture



GitHub: https://github.com/oneapi-src/oneDAL

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

```
import daal4py as d4p
d4p.patch_sklearn()
from sklearn.svm import SVC

X, Y = get_dataset()


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

Available through Intel conda
(conda install daal4py –c intel)

```
> python -m daal4py <your-scikit-learn-script>
```

Monkey-patch any scikit-learn*
on the command-line

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

Common Scikit-learn

```
from sklearn.svm import SVC

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

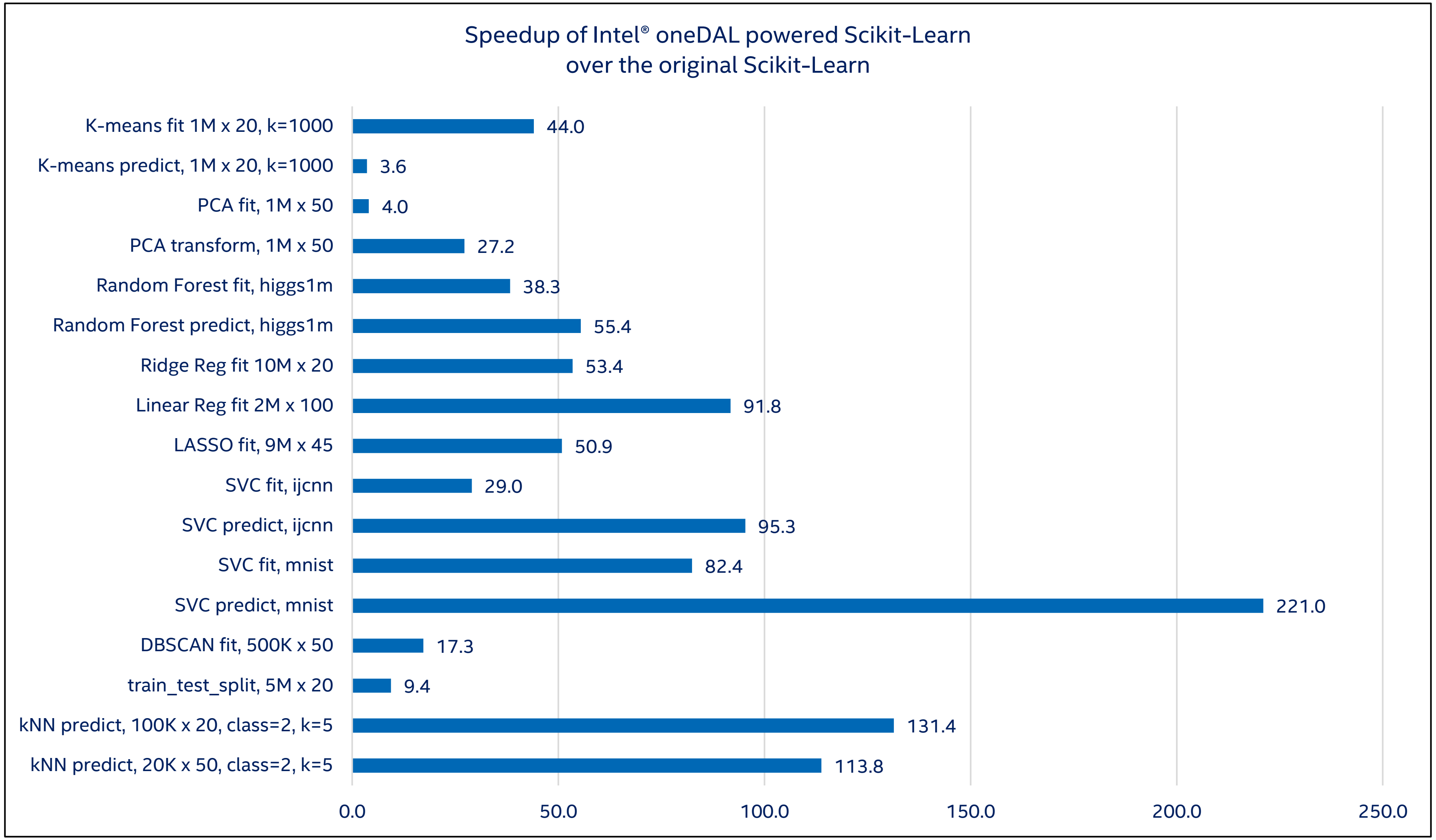
Scikit-learn with daal4py optimizations

```
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 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

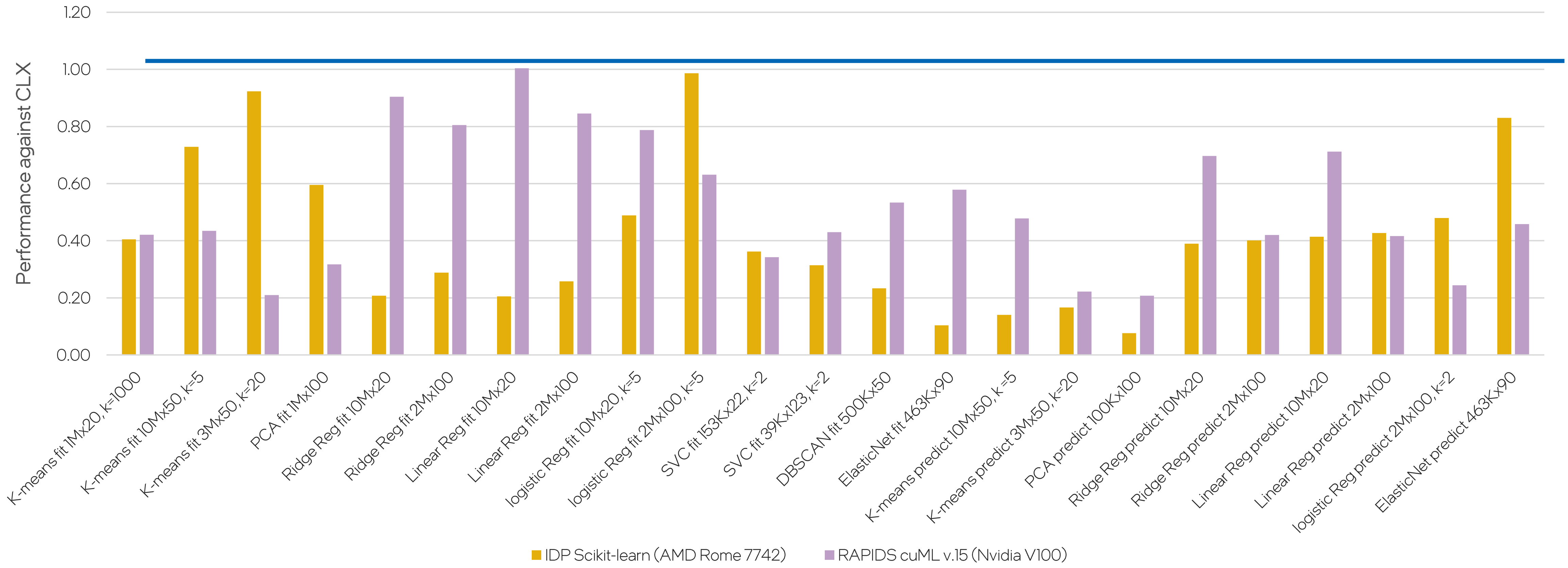
Same Code, Same Behavior

PASSED

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

Competitor's Relative Performance vs. Intel® Distribution for Python* (IDP) with Scikit-learn* from the Intel® AI Analytics Toolkit

(Intel = 1)



Testing Date: Performance results are **based on testing by Intel as of October 23, 2020** and may not reflect all publicly available security updates.

Configuration Details and Workload Setup: Intel® oneDAL beta10, Scikit-learn 0.23.1, Intel® Distribution for Python 3.7, Intel® AI Analytics Toolkit 2021.1, Intel(R) Xeon(R) 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, Intel® oneDAL beta10, Scikit-learn 0.23.1, Intel® Distribution for Python 3.7. NVIDIA Configuration: Nvidia Tesla V100-16Gb, total available memory 376 GB, 12X32GB modules, DDR4, Intel(R) Xeon(R) 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.

Demo

XGBoost

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)

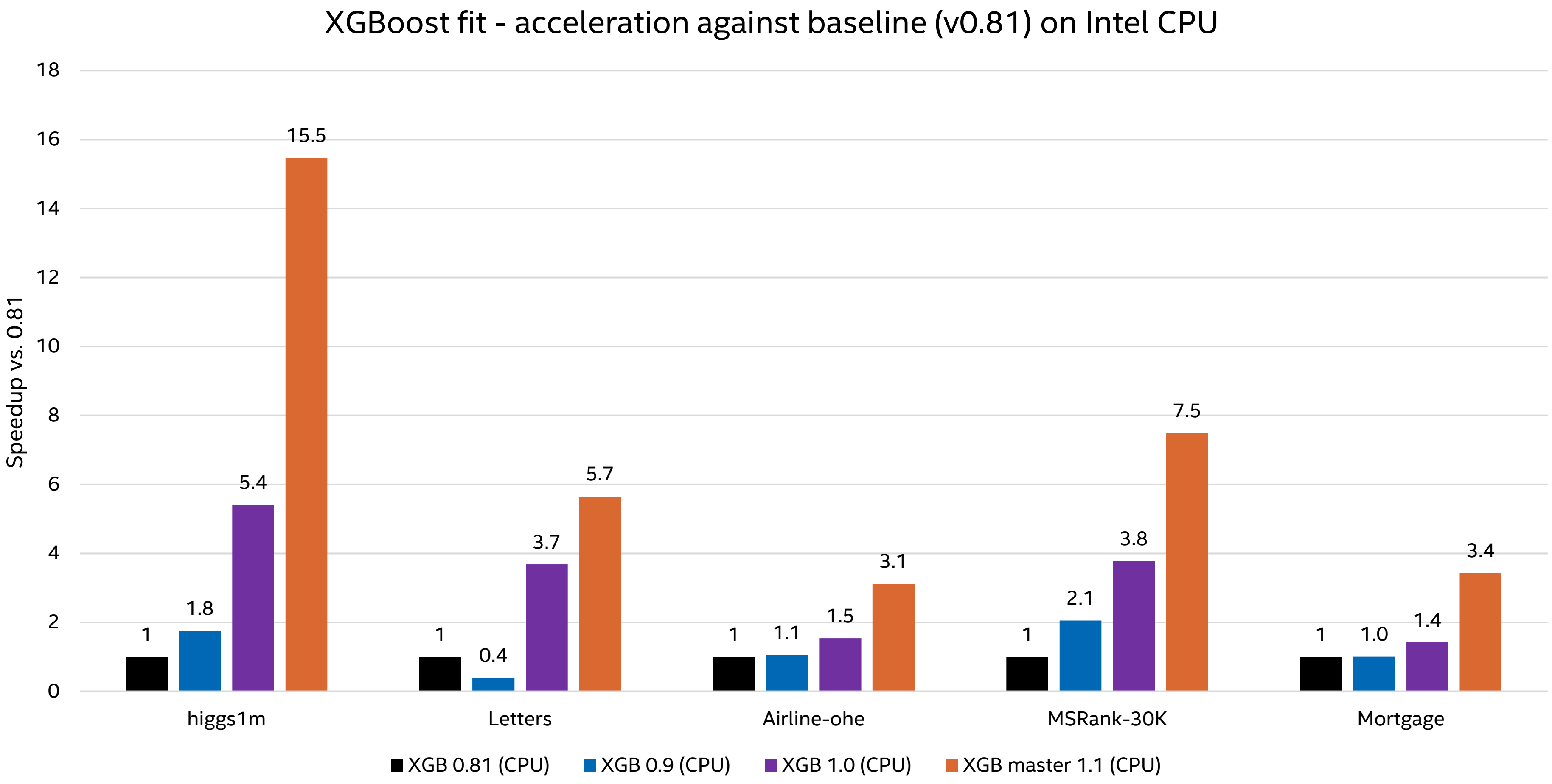
Pseudocode for Intel® oneDAL implementation

```
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® DAAL, potential optimizations for XGBoost*

XGBoost* fit CPU acceleration ("hist" method)

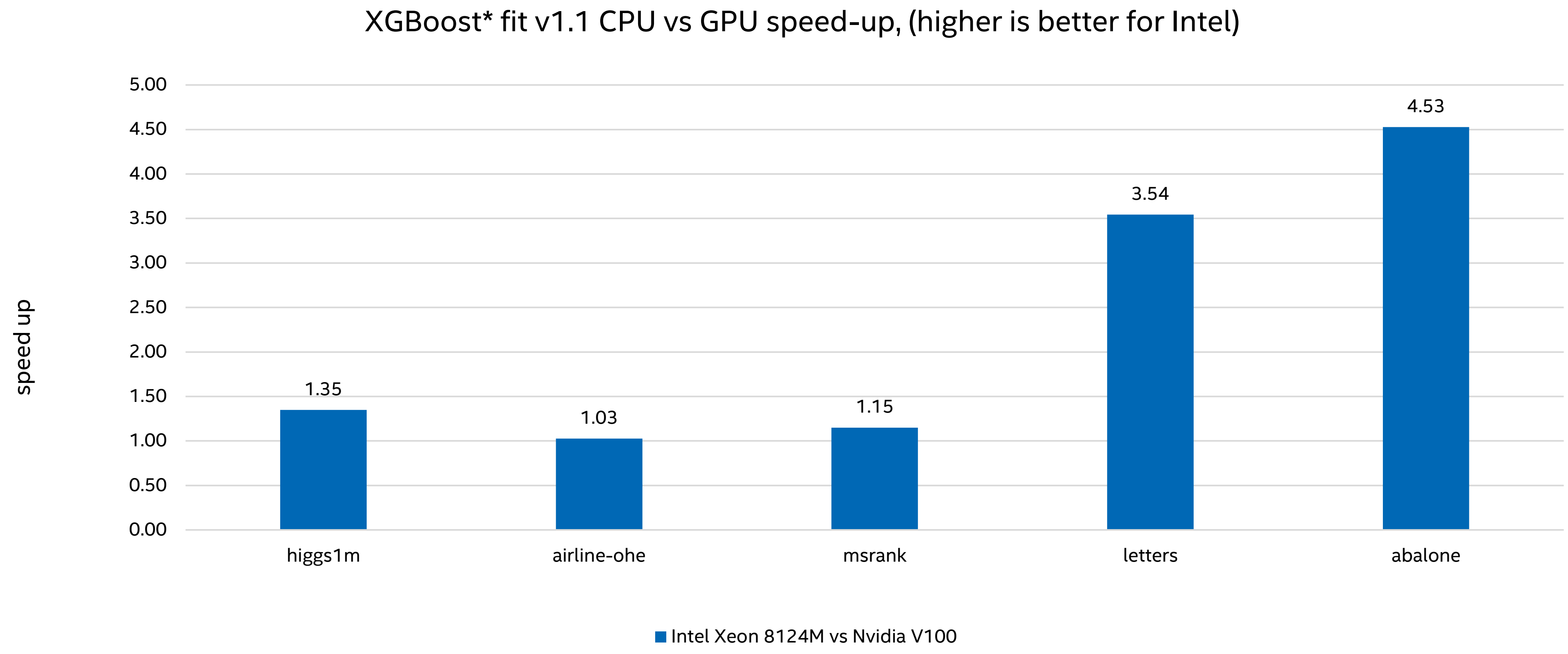


+ Reducing memory consumption

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* CPU vs. GPU



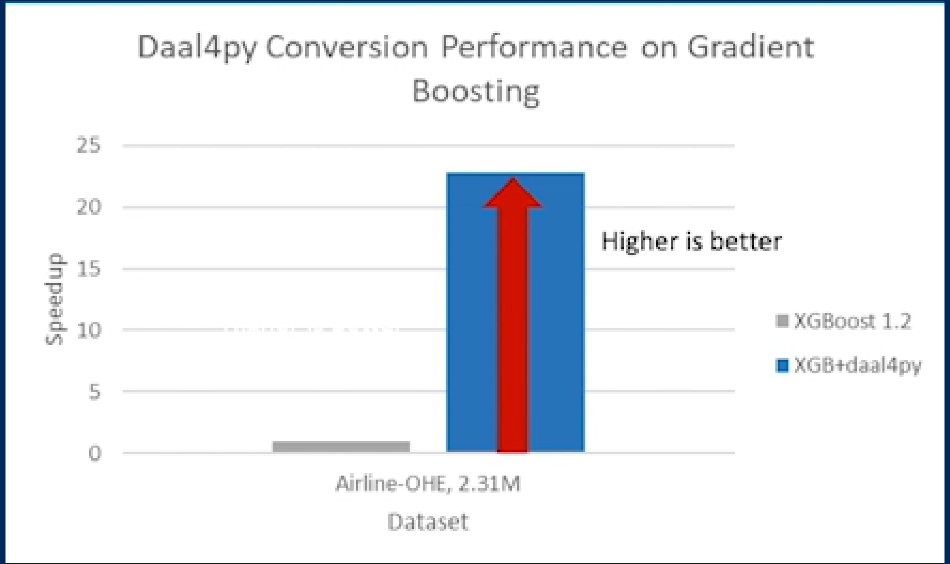
Details: https://medium.com/intel-analytics-software/new-optimizations-for-cpu-in-xgboost-1-1-81144ea21115
CPU: c5.18xlarge AWS Instance (2 x Intel® Xeon Platinum 8124M @ 18 cores, OS: Ubuntu 20.04.2 LTS, 193 GB RAM.
GPU: p3.2xlarge AWS Instance (GPU: NVIDIA Tesla V100 16GB, 8 vCPUs), OS: Ubuntu 18.04.2 LTS, 61 GB RAM.
SW: XGBoost 1.1:build from sources. compiler – G++ 7.4, nvcc 9.1. Intel DAAL: 2019.4 version, downloaded from conda. Python env: Python 3.6, Numpy 1.16.4, Pandas 0.25, Scikit-lean 0.21.2.
Testing Date: 5/18/2020

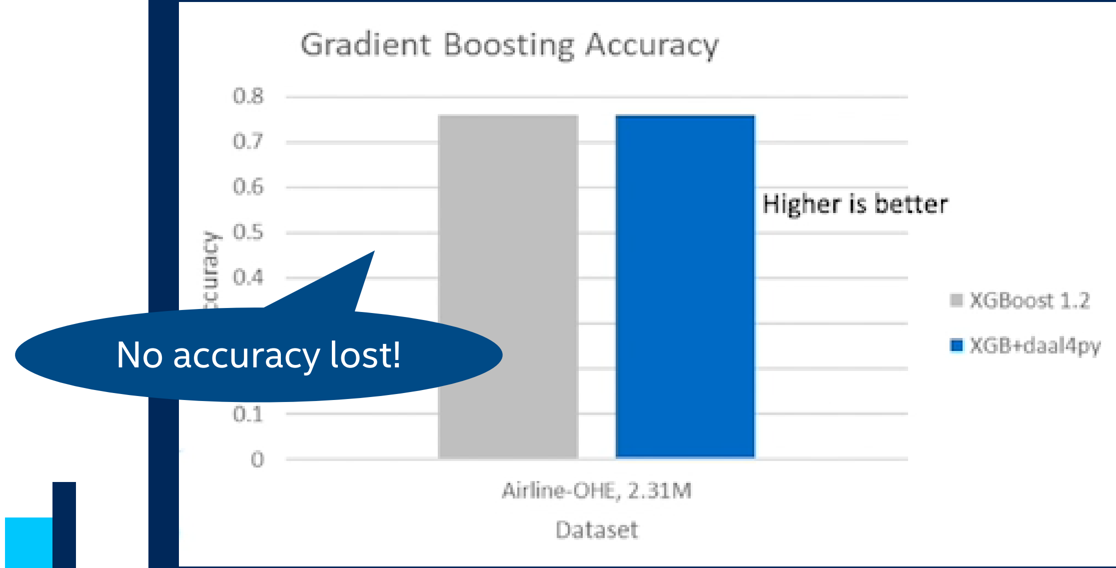
XGBoost* and LightGBM* Prediction Acceleration with Daal4Py

- 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