IXPUG Webinar Series

Performance optimizations for AI Pipelines

Meena Arunachalam, Vrushabh Sanghavi

March 11, 2021

Notices & Disclaimers

Results have been estimated or simulated.

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.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

All product plans and roadmaps are subject to change without notice.

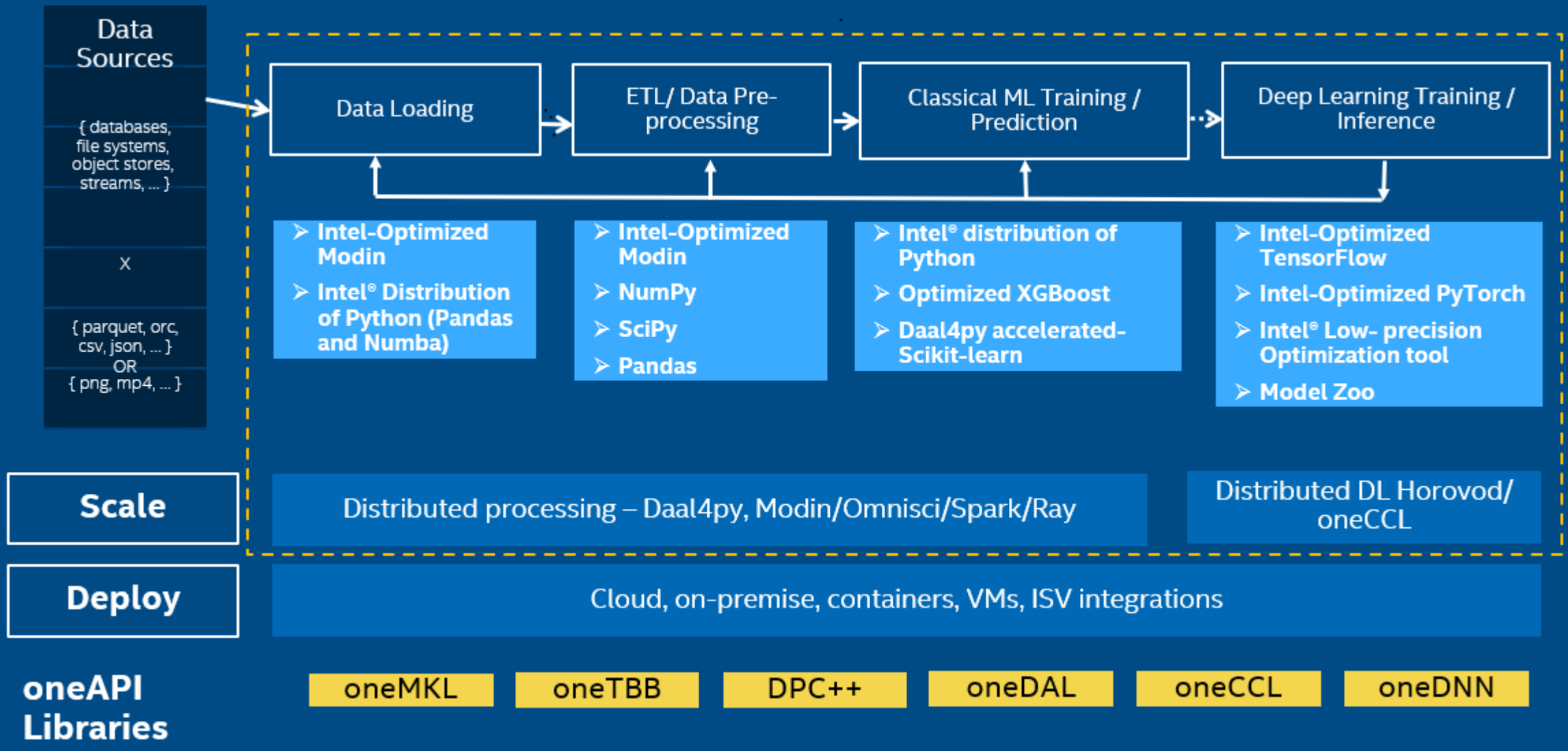
Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.

Intel contributes to the development of benchmarks by participating in, sponsoring, and/or contributing technical support to various benchmarking groups, including the BenchmarkXPRT Development Community administered by Principled Technologies.

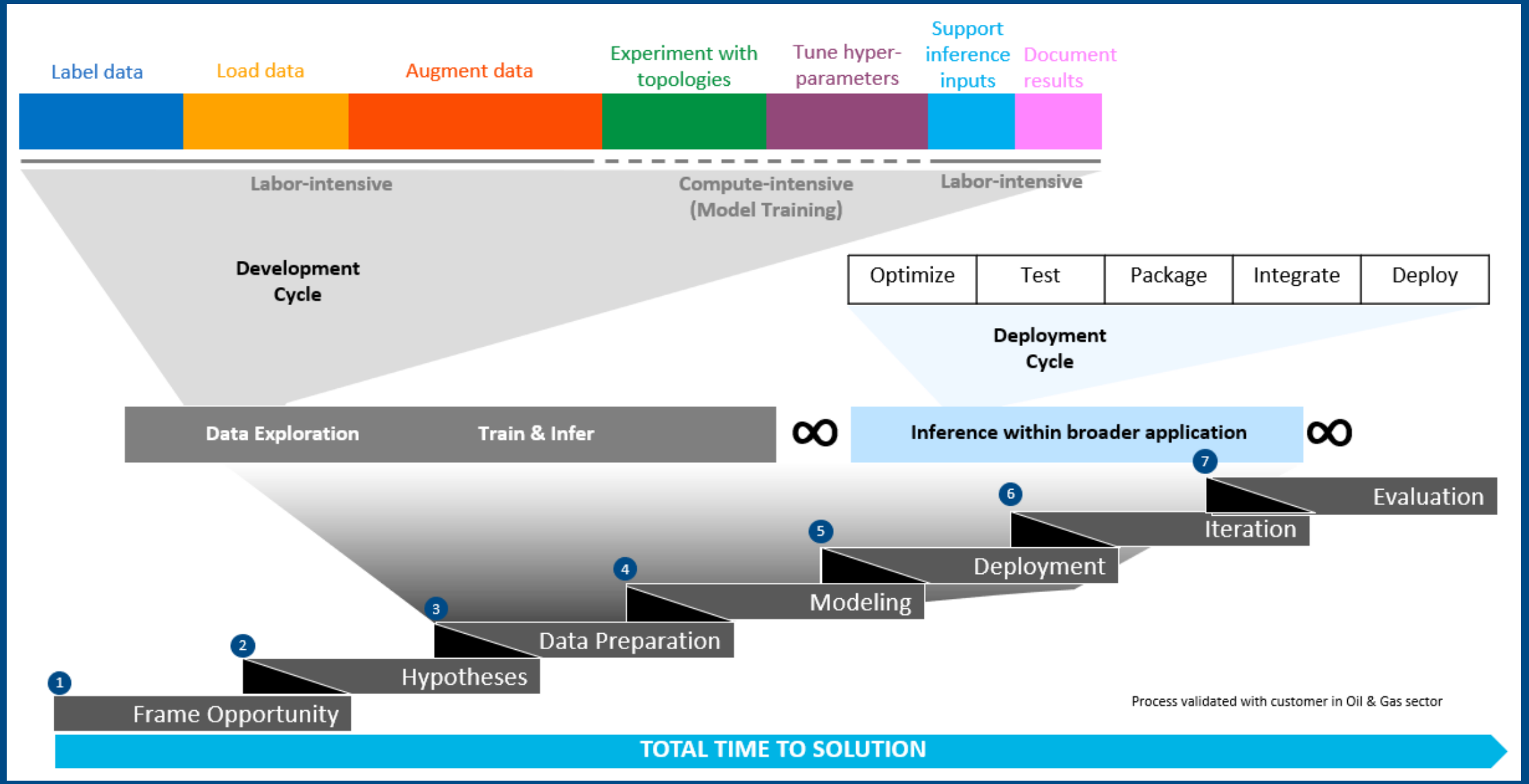
Outline

- End-to-end AI Pipelines
  - Intel-Optimized Solutions for End-to-End Analytics & AI pipeline
  - Journey to production AI
- Intel® oneAPI AI analytics toolkit
  - Intel's oneAPI Ecosystem
  - How to download
- Intel® Distribution of Modin
  - Modin layered API view
  - NYC Taxi Workload and performance
- Machine Learning – Intel® Extension for Scikit-learn and XGBoost
  - Intel® Extension for Scikit-learn
  - XGBoost CPU acceleration
- End-to-end workloads
  - Workload description and code flow
  - End-to-end workload performance
  - Hyper-parameter tuning with SigOpt
- Intel® oneAPI containers

Intel-Optimized Solutions for End-to-End Analytics & AI pipeline



Journey to Production AI



Intel's oneAPI Ecosystem

Built on Intel's Rich Heritage of CPU Tools Expanded to XPUs

oneAPI

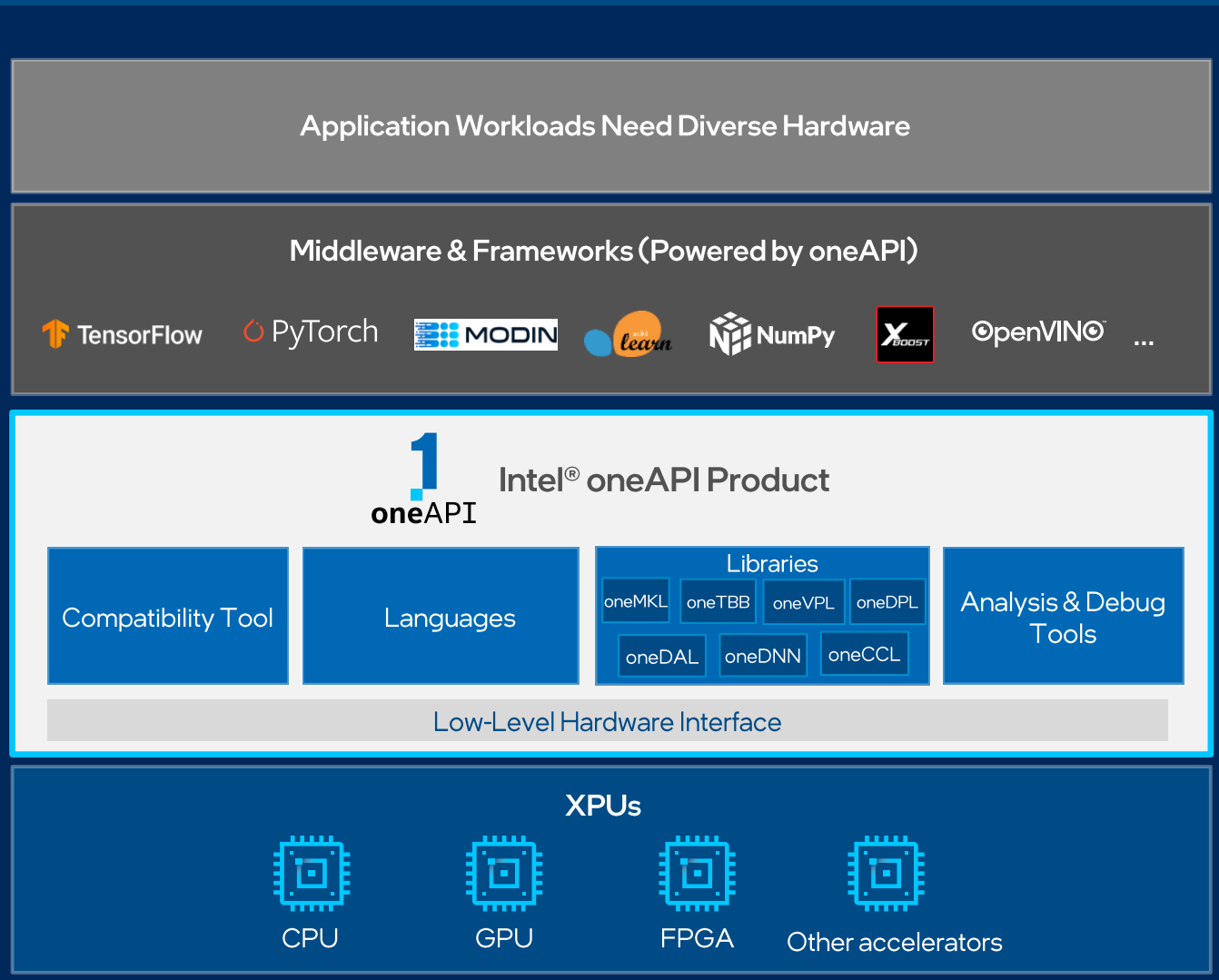
A cross-architecture language based on C++ and SYCL standards

Powerful libraries designed for acceleration of domain-specific functions

A complete set of advanced compilers, libraries, and porting, analysis and debugger tools

Powered by oneAPI

Frameworks and middleware that are built using one or more of the oneAPI industry specification elements, the DPC++ language, and libraries listed on oneapi.com.



Available Now

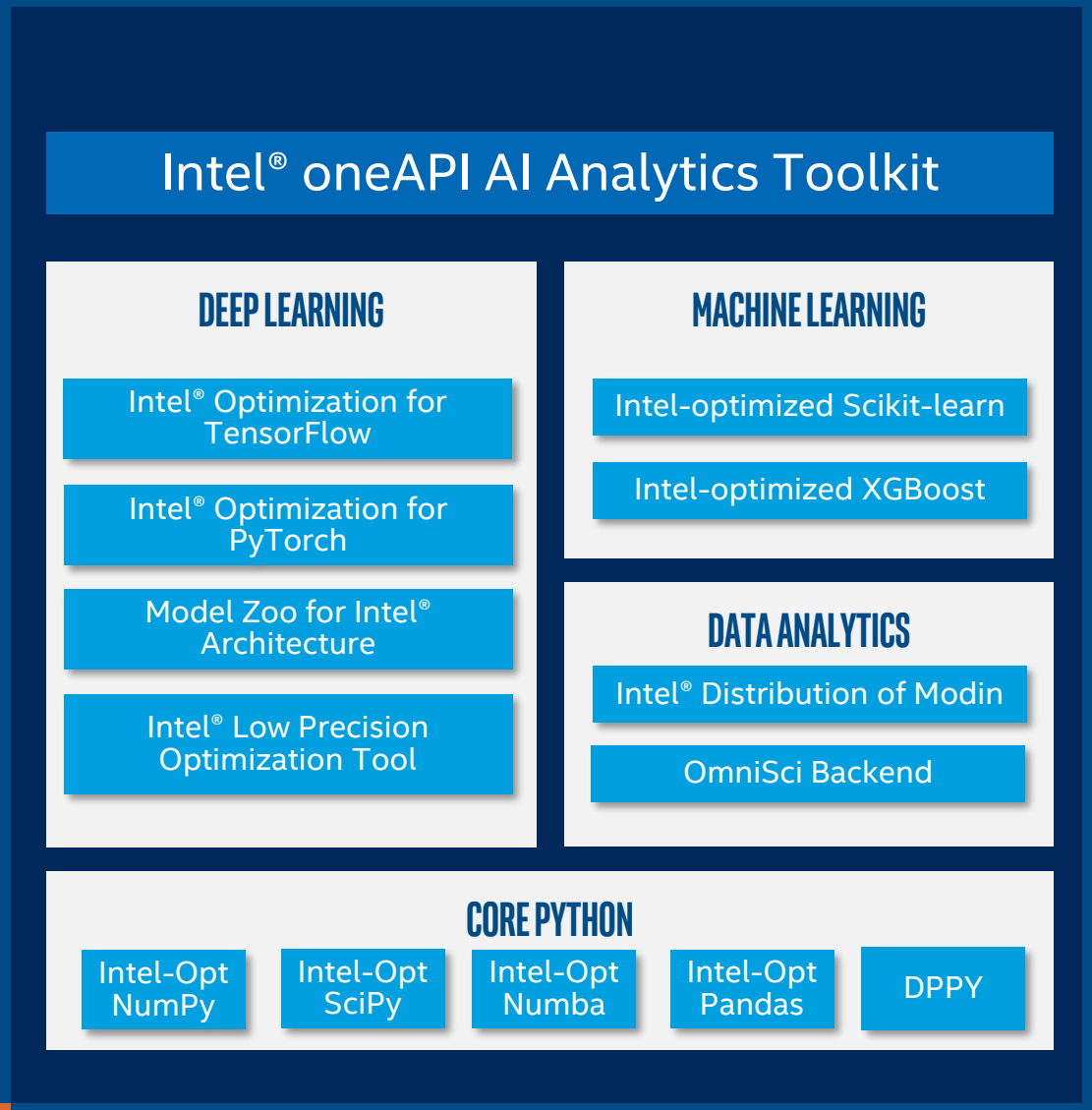
Visit software.intel.com/oneapi for more details
Some capabilities may differ per architecture and custom-tuning will still be required. Other accelerators to be supported in the future.

Intel® oneAPI AI Analytics Toolkit

Accelerates end-to-end Machine Learning and Data Analytics pipelines with frameworks and libraries optimized for Intel® architectures

Who Uses It?

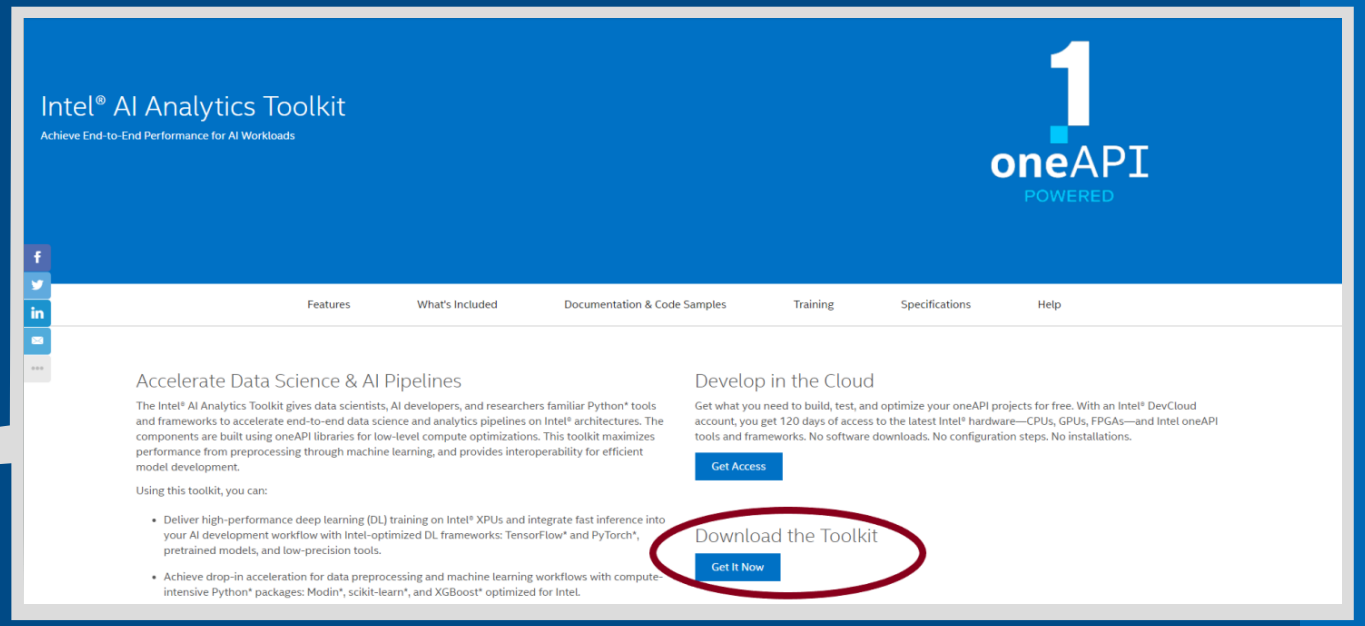
Data scientists, AI Researchers, Machine and Deep Learning developers, AI application developers

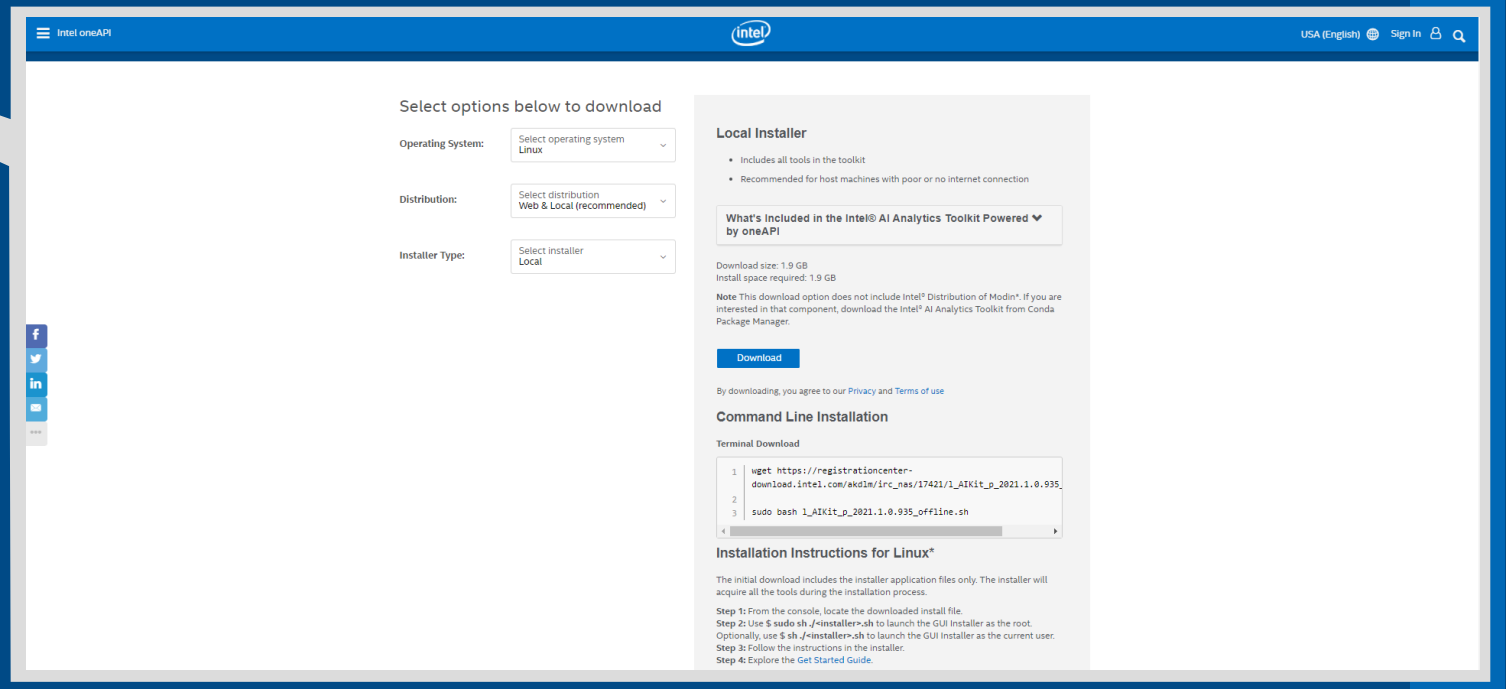


Learn More: intel.com/oneAPI-AIKit

How to download and install Intel® oneAPI Analytics Toolkit?

- Link to download page
  - Select 'Get It Now'
  - Select the OS and distro
  - Follow the download instructions and steps
- Other Links
  - Getting Started Guide
  - Build and Run a Sample
  - Intel® oneAPI AI Analytics Toolkit Code Samples.





Intel® Distribution for Modin

Data Ingestion, Data pre-processing and Data wrangling

https://github.com/modin-project/modin

Modin

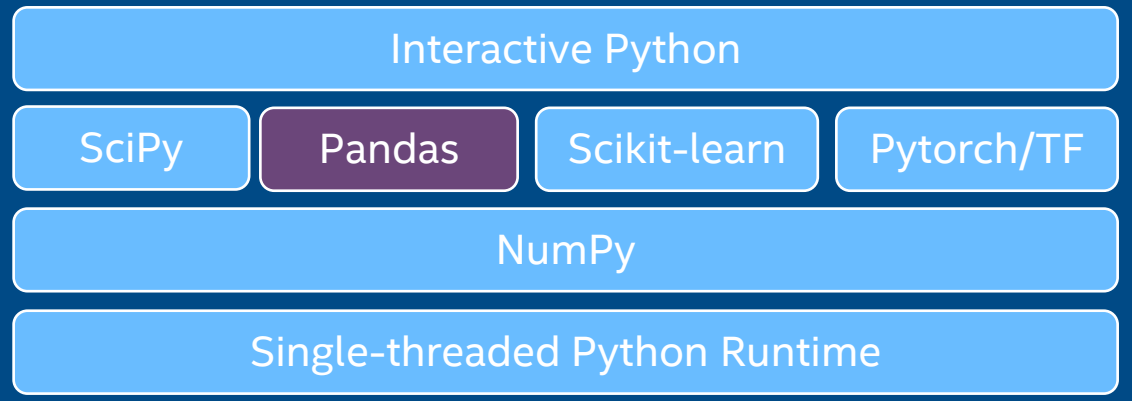
- **Modin helps scale Pandas workflow by changing a single line of code**
- To use Modin, users do not need to know how many cores the system has, and do not need to specify how to distribute the data. Users can continue using their previous Pandas notebooks while experiencing a considerable speedup from Modin, even on a single machine
- Modin transparently distributes the data and computation across available cores, unlike Pandas which only uses one core at a time
- With Modin, users get a fast data frame at 1MB and 1TB+ alike allowing users to use the same tool for doing the same computation on different sizes of data

Modin can be installed from PyPI:

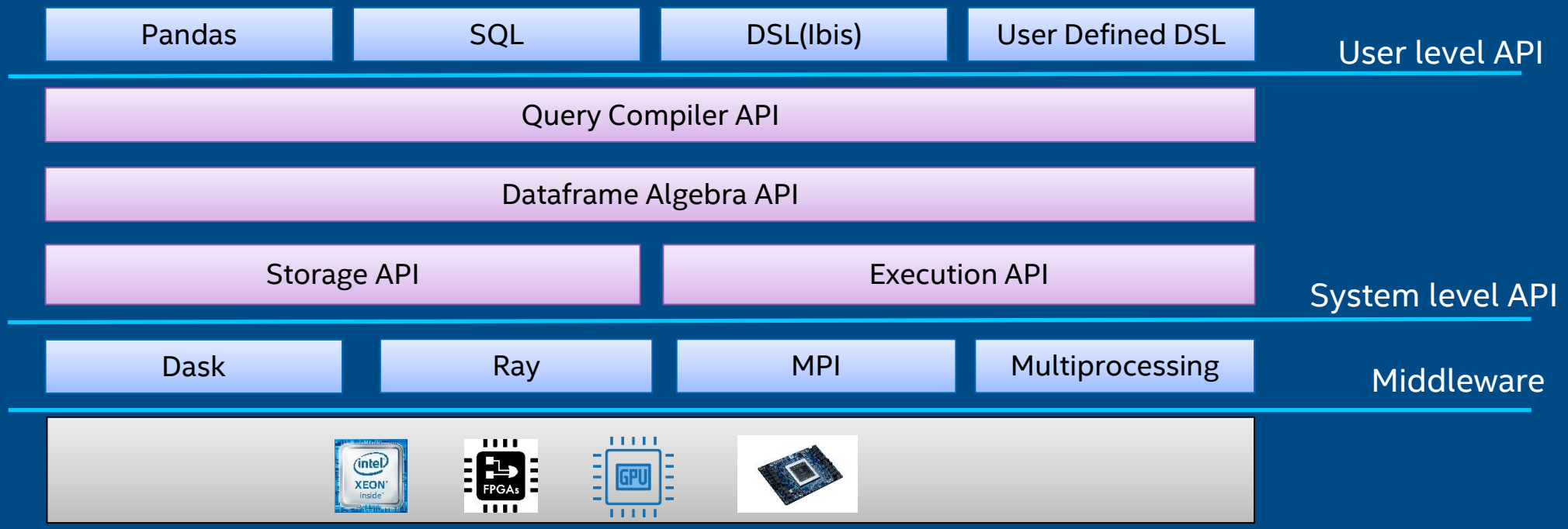
```
pip install modin
```

```
# import pandas as pd
import modin.pandas as pd
```

Only a single line import change needed to run modin instead of pandas



Modin – layered API view



NYC Taxi Workload

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

- The dataset consists of up to 1.1 billion individual taxi trips in the city of New York from January 2009 through June 2015, covering both yellow and green taxis
- The NYCtaxi workload ingests the large dataset into a data frame and performs queries over them

Data Ingestion into a data frame

```
df = pd.read_csv(
    os.path.expanduser('~/trips_xaa.csv'),
    names=columns_names,
    dtype=all_but_dates,
    parse_dates=dates_only,
)
```

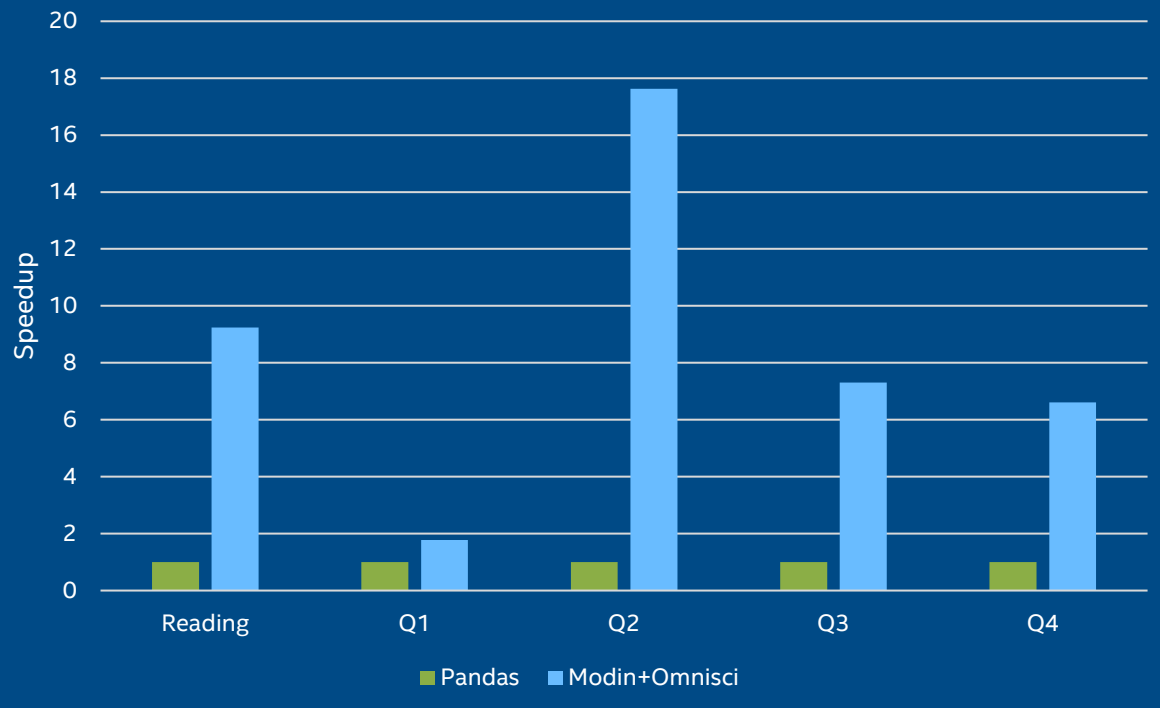
Data processing and wrangling queries over the data frame

```
def q1_omnisci(df):
    q1_pandas_output = df.groupby("cab_type").size()
    q1_pandas_output.shape  # to trigger real execution
    return q1_pandas_output

def q2_omnisci(df):
    q2_pandas_output = df.groupby("passenger_count").agg({"total_amount": "mean"})
    q2_pandas_output.shape  # to trigger real execution
    return q2_pandas_output

def q3_omnisci(df):
    df["pickup_datetime"] = df["pickup_datetime"].dt.year
    q3_pandas_output = df.groupby(["passenger_count", "pickup_datetime"]).size()
    q3_pandas_output.shape  # to trigger real execution
    return q3_pandas_output

def q4_omnisci(df):
    df["pickup_datetime"] = df["pickup_datetime"].dt.year
    df["trip_distance"] = df["trip_distance"].astype("int64")
    q4_pandas_output = (
        df.groupby(["passenger_count", "pickup_datetime", "trip_distance"], sort=False)
        .size()
        .reset_index()
        .sort_values(by=["pickup_datetime", 0], ignore_index=True, ascending=[True, False])
    )
    q4_pandas_output.shape  # to trigger real execution
    return q4_pandas_output
```

NYCTaxi workload performance

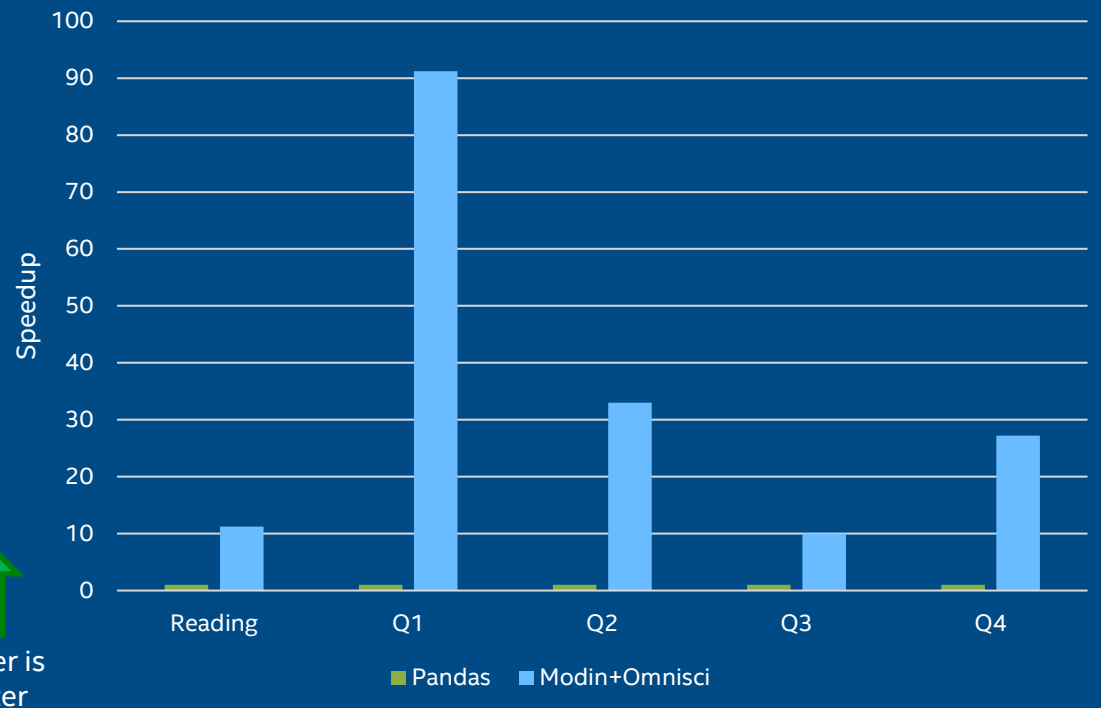
Pandas vs Modin

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



Higher is better

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



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

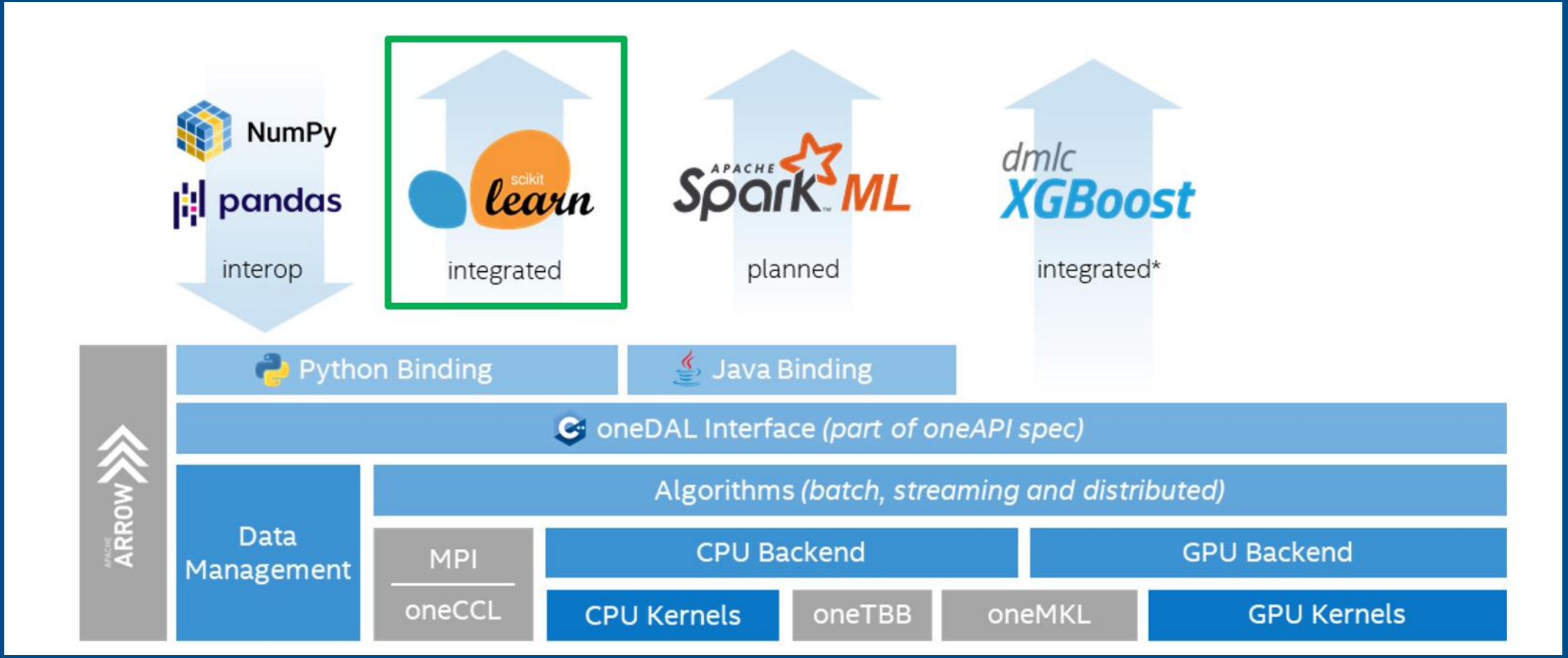
Intel® Extension for Scikit-learn and XGBoost

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

Article: https://medium.com/intel-analytics-software

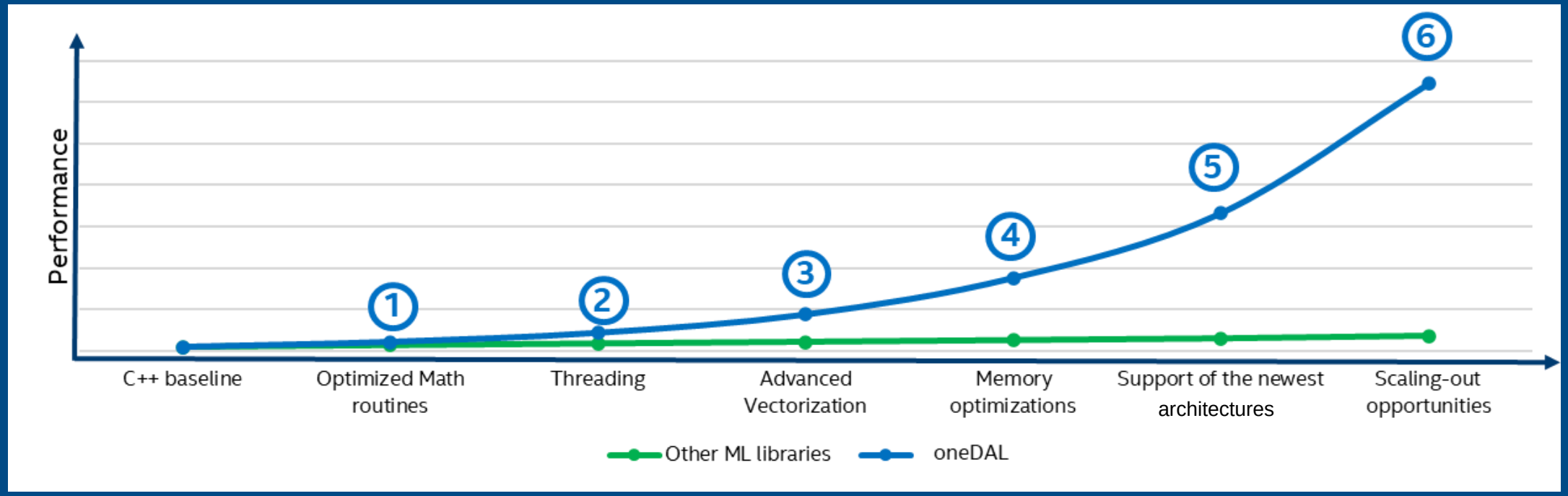
oneAPI Data Analytics Library (oneDAL)

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



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

What makes oneDAL faster?



1. The best performance on Intel Architectures with oneMKL (Intel® MKL) vs. lower performance on BLAS/LAPACK libs

2. oneDAL 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, intrinsic. Usually, other ML libs build applications without vector-instructions support or

4. oneDAL 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 launch. Usually, other ML libs do this with long delay.

6. oneDAL provides distributed algorithms which scale on many nodes

Intel Distribution for Python (IDP) 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

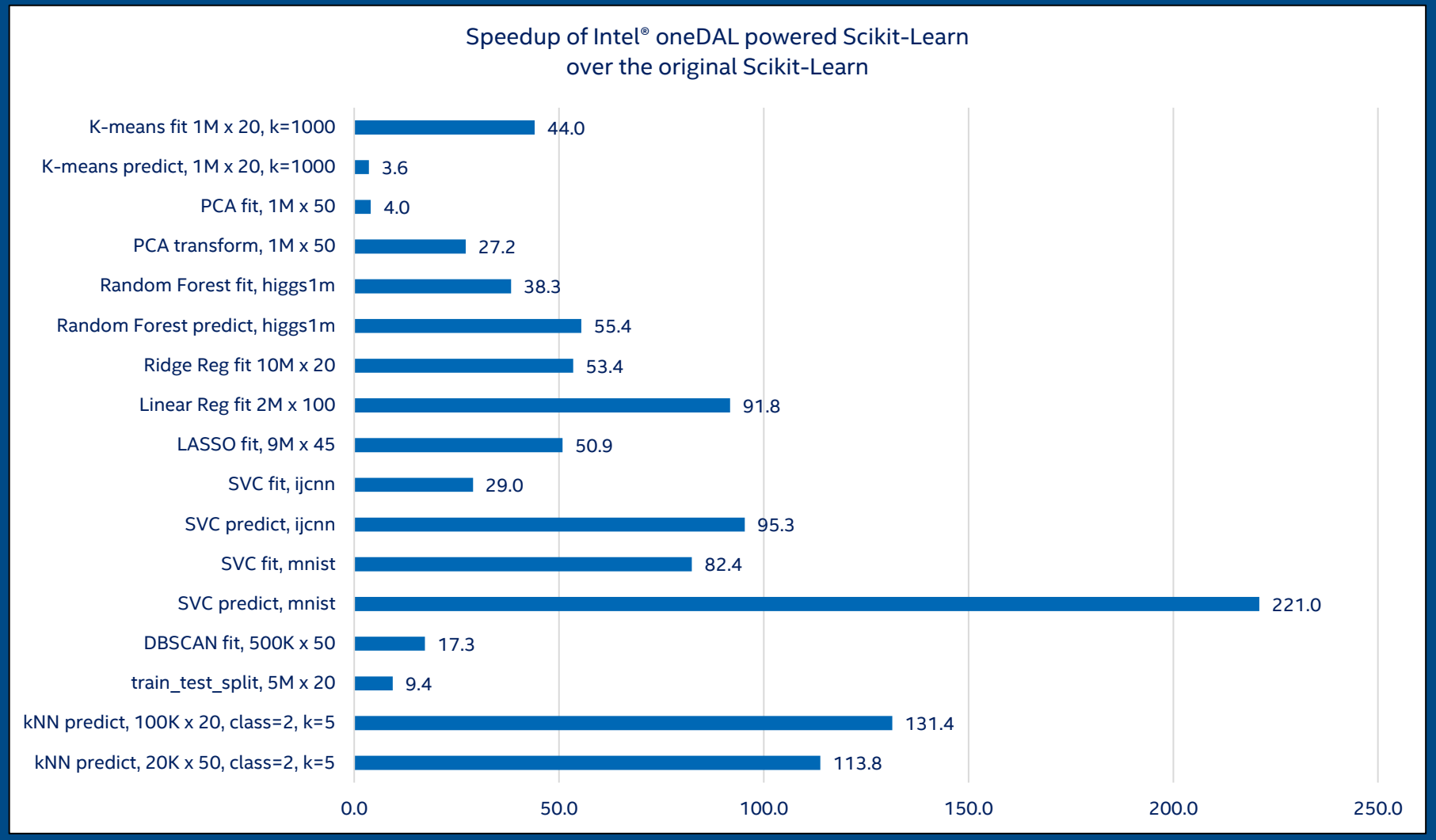
```
import daal4py as d4p
d4p.patch_sklearn()
from sklearn.svm import SVC

X, Y = get_dataset()

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

Available through PyPi
pip install daal4py

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

Intel® Extension for Scikit-learn (accelerated by oneDAL)



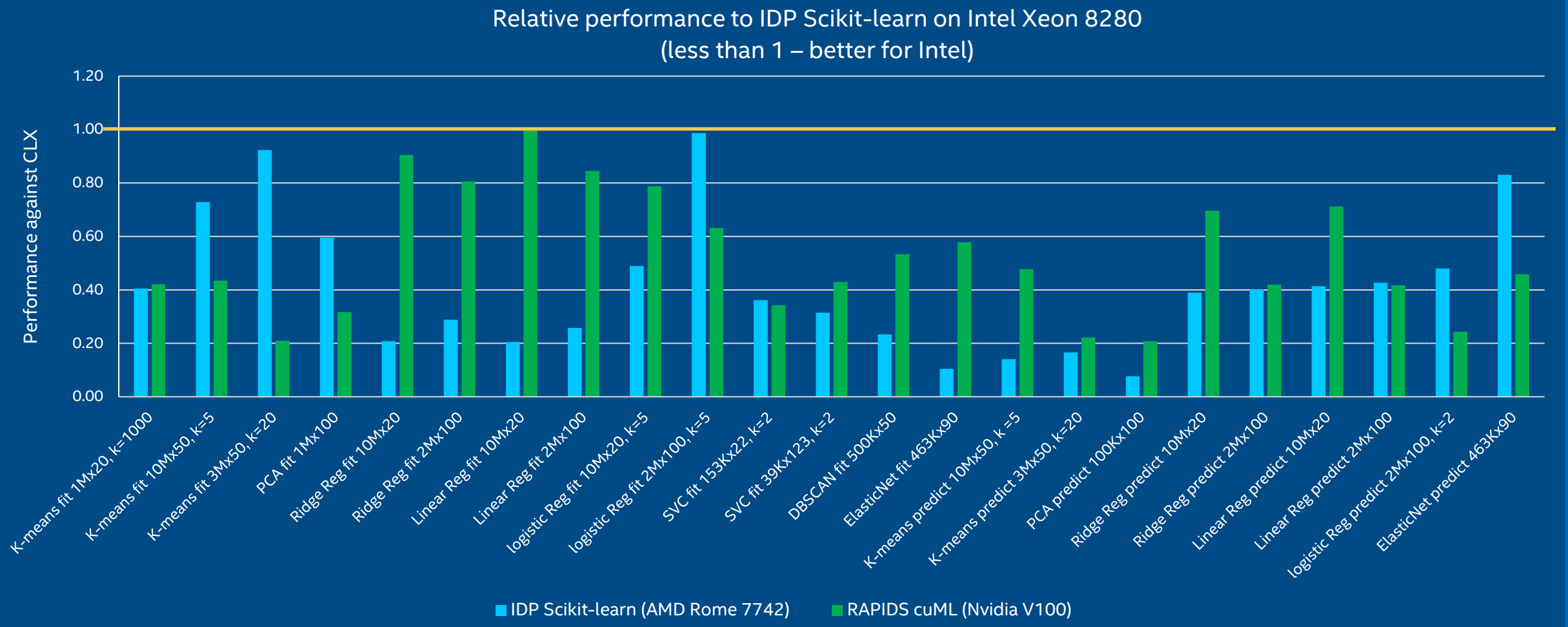
Same Code,
Same Behavior

PASSED

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

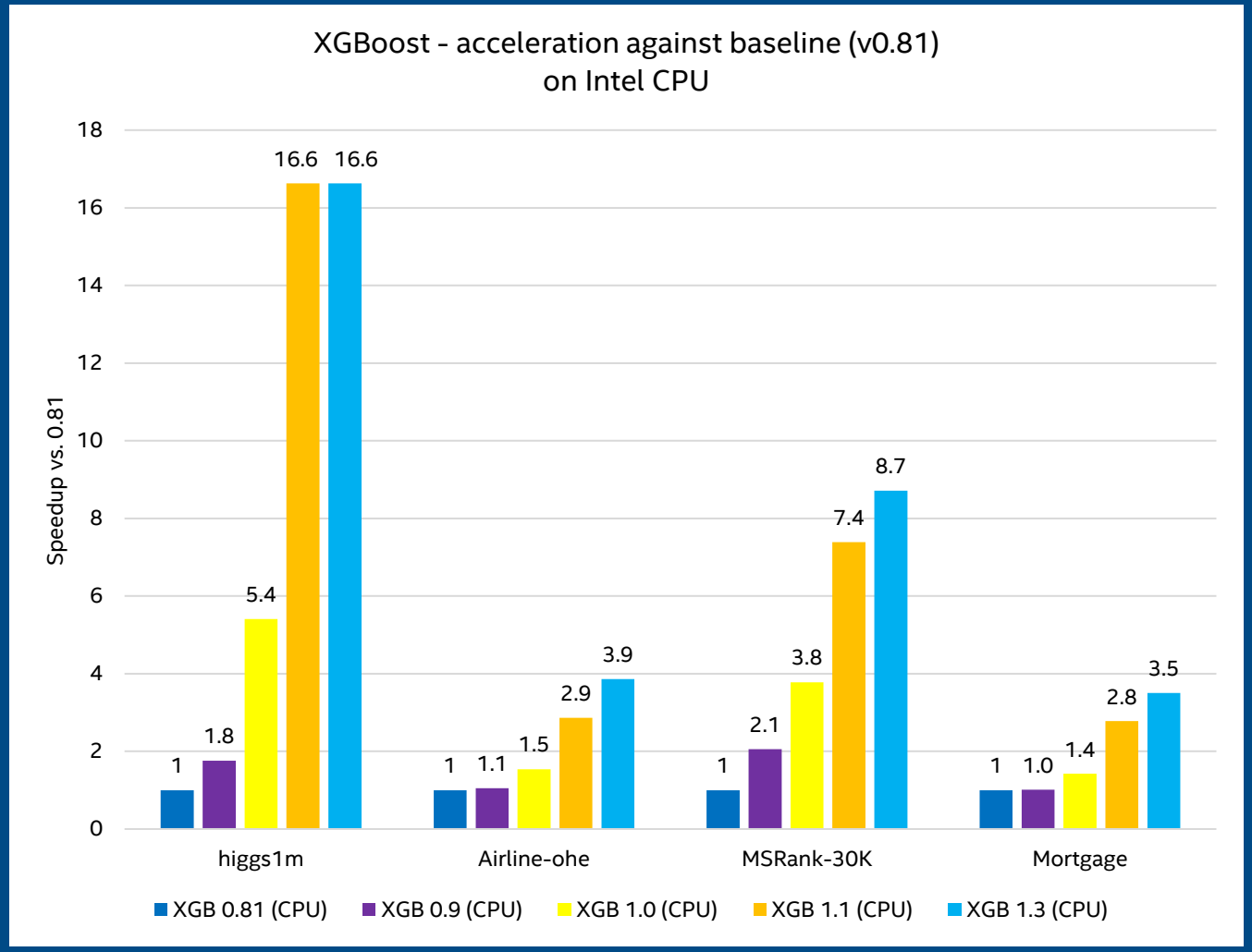
HW: Intel Xeon Platinum 8276L CPU @ 2.20GHz, 2 sockets, 28 cores per socket;
Details: https://medium.com/intel-analytics-software/accelerate-your-scikit-learn-applications-a06cacf44912

Intel® Extension for Scikit-learn (accelerated by oneDAL)



Intel CPU: Intel Xeon Platinum 8280 CPU @ 2.70GHz, 2 sockets, 28 cores per socket;
AMD CPU: Rome 7742 @2.25GHz, 2 sockets, 64 cores per socket
Nvidia GPU: Tesla V100-16GB
SW: daal4py 2020.3, Scikit-learn 0.23.2, RAPIDS cuML 0.15

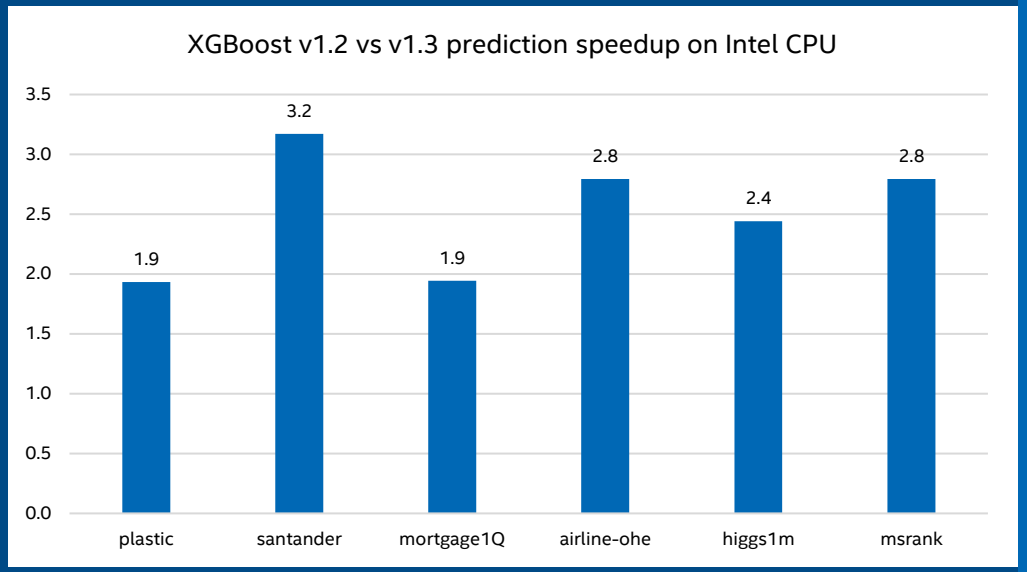
XGBoost CPU acceleration ("hist" method)



+ Reducing memory consumption of fitting

memory, Kb	Airline	Higgs1m
Before	28311860	1907812
#5334	16218404	1155156
reduced:	1.75	1.65

Higher is better



CPU configuration: CLX 8280 @ 2.7GHz, 2 sockets, 28 cores per socket

XGBoost CPU acceleration – gain sources

Training

Pseudocode for XGBoost v0.81 (baseline):

```
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

Nested parallelism by dtree nodes

Usage uint8 instead of uint32

Advanced parallelism for samples partitioning, reducing seq loops

SIMD instructions instead of scalar code

Pseudocode for optimized XGBoost:

```
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)
```

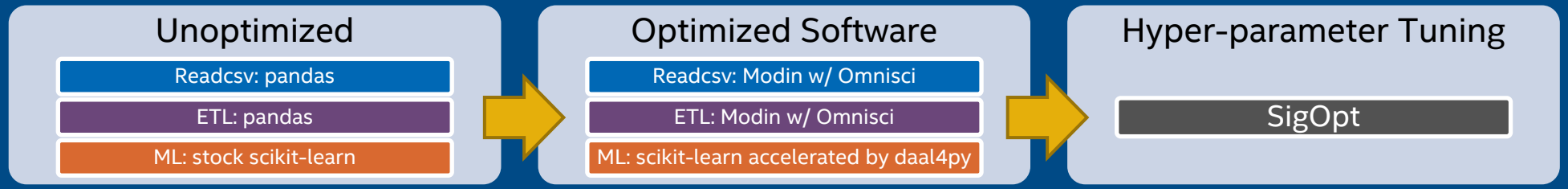
Legend: Released XGB 1.3 | Future scope

End-to-end Machine Learning workloads

E2E workload

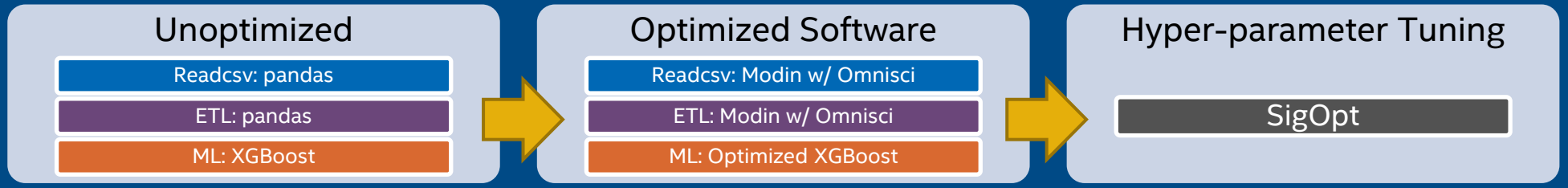
Census

- The Census workload uses the U.S. census data from 1970 to 2010 in order to build a ridge regression-based training model to predict the education attainment level based on income, age, gender



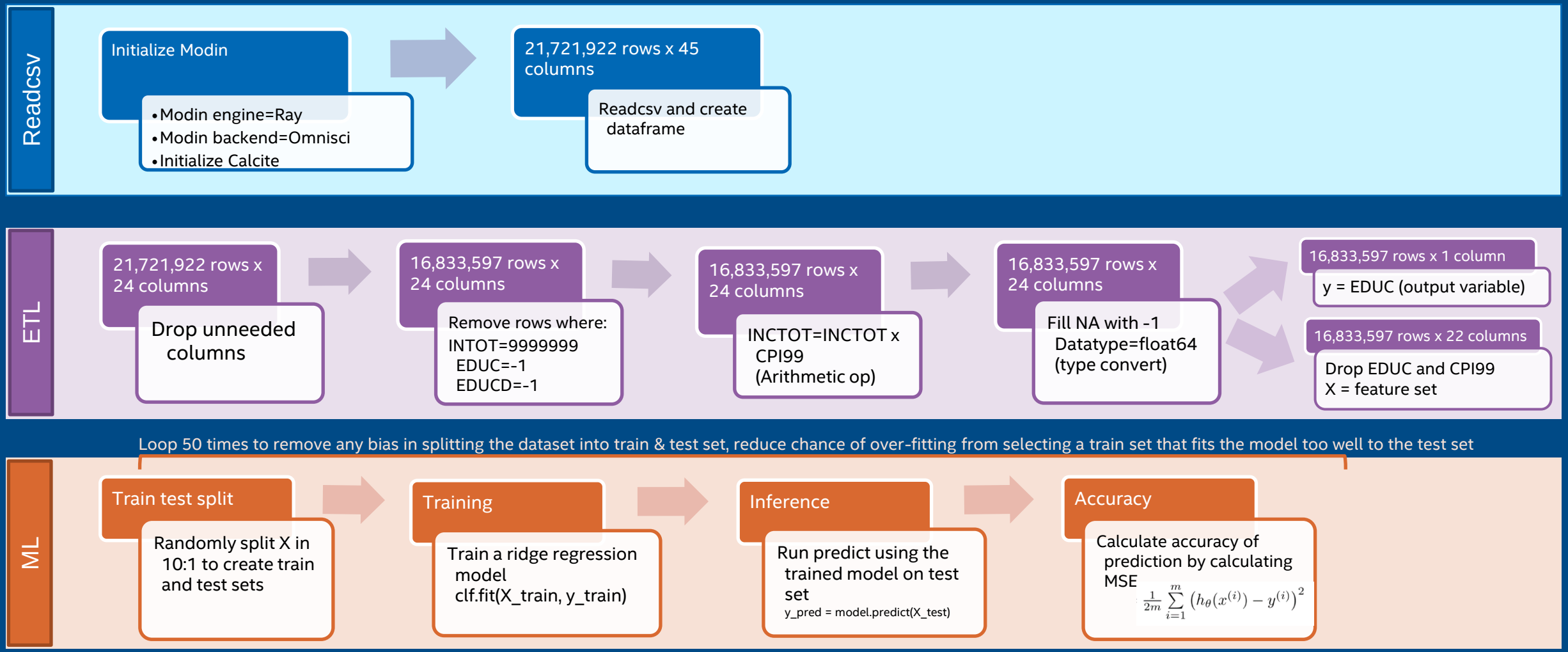
PLAsTiCC

- PLAsTiCC is an open data challenge to classify objects in the sky that vary in brightness using simulated astronomical time-series data. The challenge is to determine a probability that each object belongs to one of 14 classes of astronomical filters.



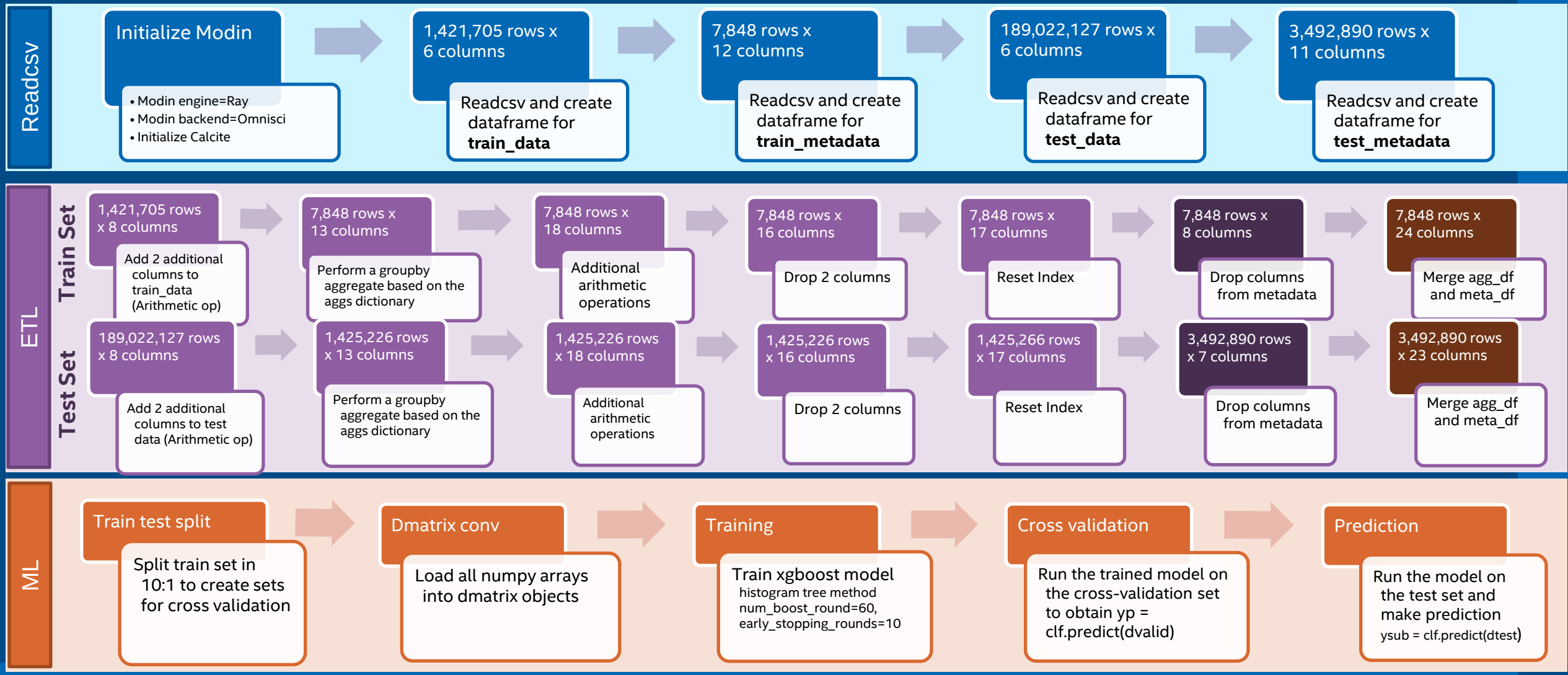
Census

Dataset includes census year, household serial number, inflation adjustment factor, age, sex, personal income and education

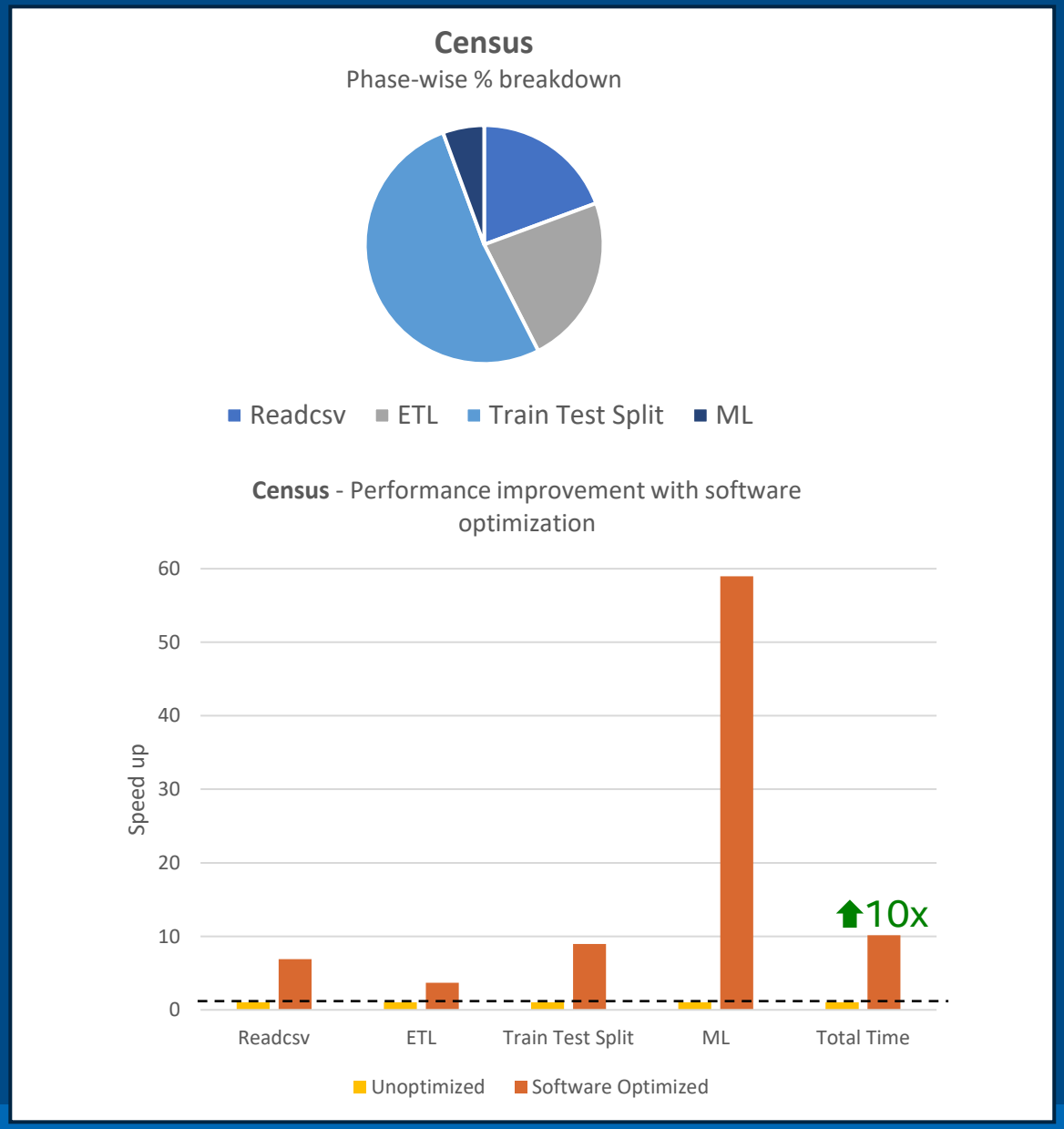


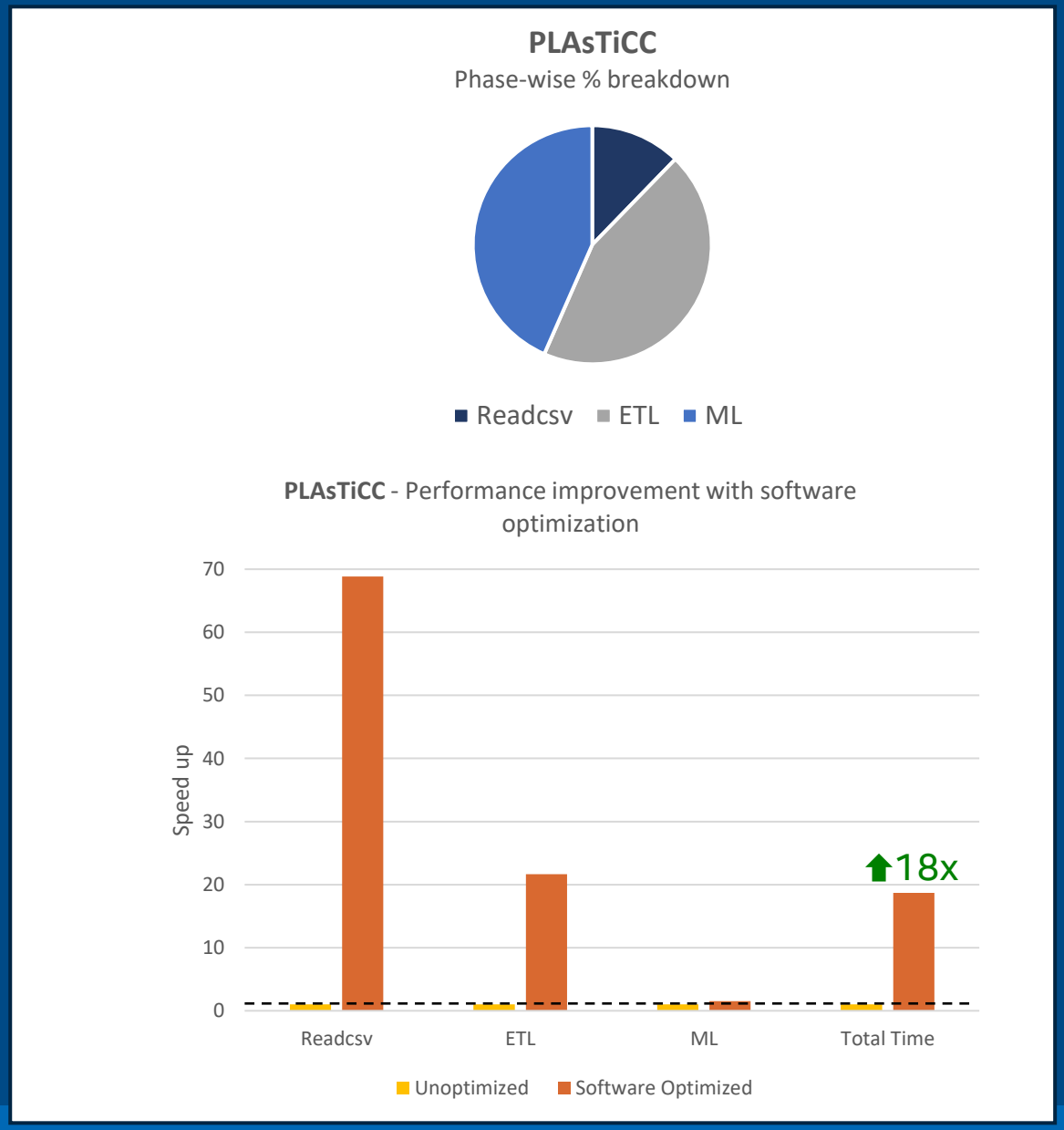
PLAsTiCC

The PLAsTiCC data are separated into a training data set and a test set; the latter is the data without classifications that needs to be classified
Train/test meta data lists each source in the data indexed by a unique identifier & each row of the table lists the properties of that source
Train/test time-series data contains information about the sources and their brightness in different passbands as a function of time



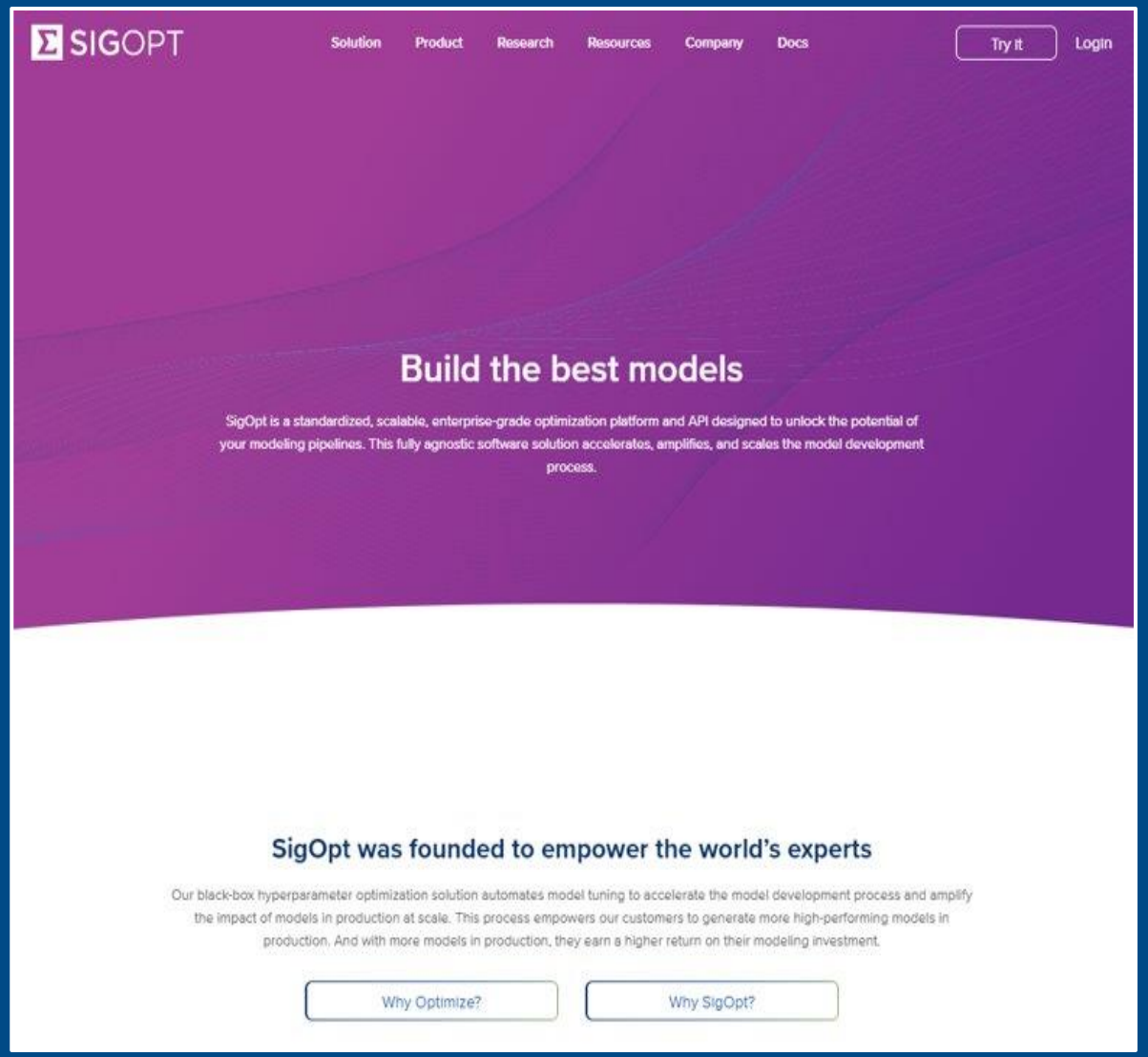
End-to-end workload performance





Higher is better

Hyper parameter tuning with SigOpt



- https://sigopt.com/
- Sigopt features:
  - Easy to track runs, visualize training, and scale hyperparameter optimization
  - advanced Optimization Engine that delivers better results, faster and cheaper
  - Easy to use and parallelize for any type of model built with any library on any infrastructure

Example Sigopt Use Case for XGBOOST

Write a new python script that runs the application

```python
import subprocess
import numpy as np
from sigopt import Connection

def main(assignments):

    # parameters
    dxgb_cpu_params = {
        'nthread': assignments['nthread'], 'nround': assignments['nround'],
        'subsample': assignments['subsample'],'colsample_bytree': assignments['colsample_bytree'],
        'max_bin': assignments['max_bin'], 'max_depth': assignments['max_depth'],
        'min_child_weight': assignments['min_child_weight'],'learning_rate': assignments['learning_rate'],
        'gamma': assignments['gamma'],'alpha': assignments['alpha'],'reg_lambda': assignments['reg_lambda'],
        'scale_pos_weight': assignments['scale_pos_weight'],'tree_method': 'hist','predictor': 'cpu_predictor',
    }

    nthread=dxgb_cpu_params['nthread']

    #set channel and core for numactl
    nchannel = str("0") if nthread<=48 else str("0-1")
    if nthread<=24:
        ncore = str("0-")+str(nthread-1)
    if nthread>24 and nthread<=48:
        ncore = str("0-23,48-")+str(48+nthread-1-24)
    if nthread>48 and nthread<=72:
        ncore = str("0-23,48-71,72-")+str(72+nthread-1-48)
    if nthread>72:
        ncore = str("0-")+str(nthread-1)
    print("numactl -m ",nchannel," -C ",ncore," python optuna_numa.py xgb")

    # transfer dict to json str
    import json
    input_data=json.dumps(str(dxgb_cpu_params))
```

Read result metric that needs to be optimized, set as return value

```python
    # commands
    subprocess.call('numactl -m '+nchannel+' -C '+ncore+' python sigopt_numa.py xgb '+str(input_data)+' > 1.txt', shell=True)
    subprocess.call('grep 'timing *' 1.txt > 2.txt')
    subprocess.call('cut -d" " -f 2,4 2.txt > 3.txt', shell=True)
    result=np.loadtxt('3.txt')

    return [{'name': 'timing', 'value': result[0]},  {'name': 'accuracy', 'value': result[1]}]
```

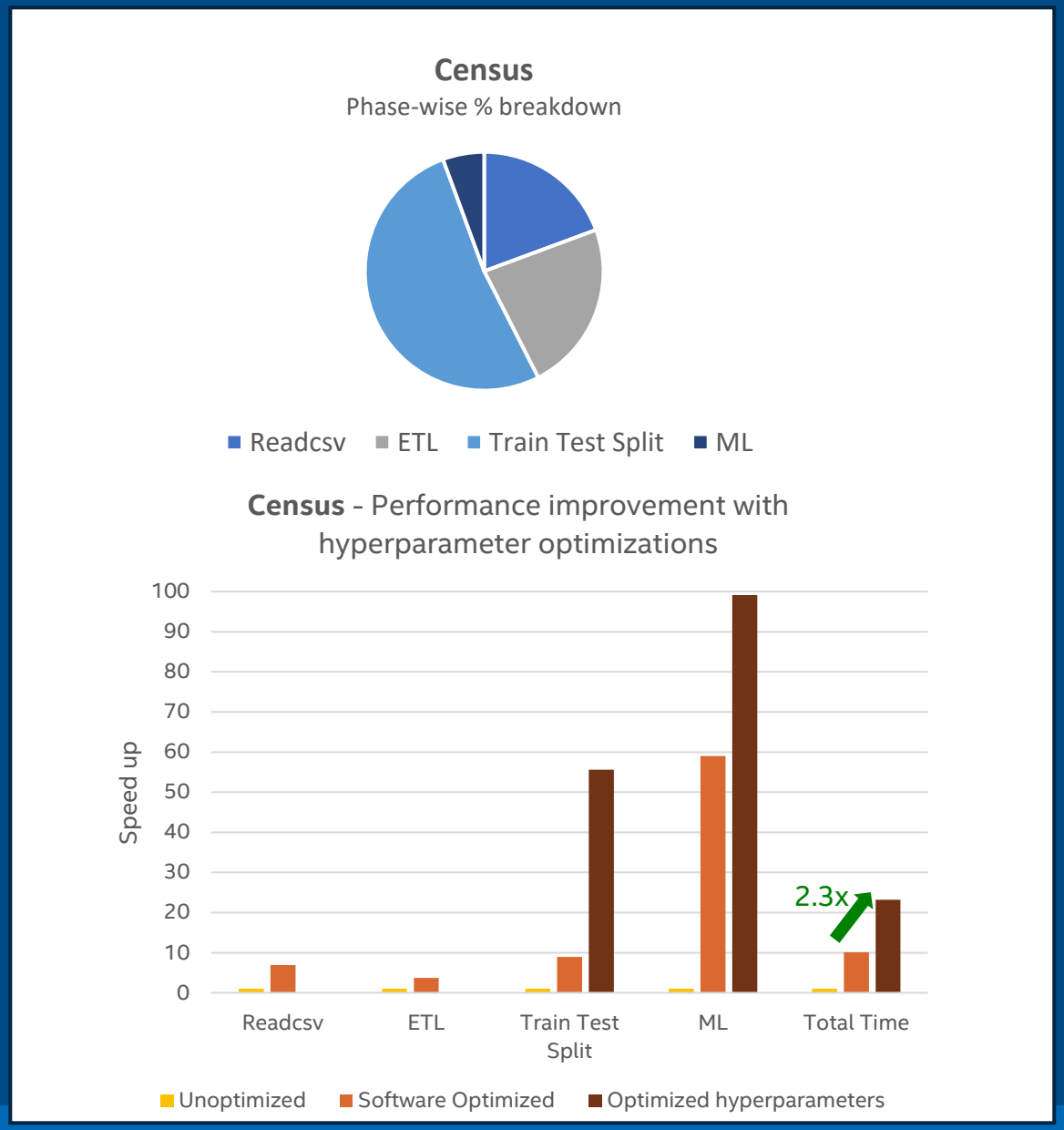
```python
# ////////////////////////////////  sigopt ///////////////////////////////////
conn = Connection(client_token="                                        ")

experiment = conn.experiments().create(
    name="xgboost",
    project="sigopt-examples",
    metrics=[
        dict(name='accuracy', objective='minimize'),
        dict(name='timing', objective='minimize')
    ],
    parameters=[
        dict(name="nthread", type="int", bounds=dict(min=4, max=92)),
        dict(name="nround", type="int", bounds=dict(min=1, max=20)),
        dict(name="subsample", type="double", bounds=dict(min=0, max=1)),
        dict(name="colsample_bytree", type="double", bounds=dict(min=0, max=1)),
        dict(name="max_bin", type="int", bounds=dict(min=4, max=1024)),
        dict(name="max_depth", type="int", bounds=dict(min=1, max=20)),
        dict(name="min_child_weight", type="double", bounds=dict(min=0, max=10)),
        dict(name="learning_rate", type="double", bounds=dict(min=0, max=1)),
        dict(name="gamma", type="double", bounds=dict(min=0, max=1)),
        dict(name="alpha", type="double", bounds=dict(min=0, max=1)),
        dict(name="reg_lambda", type="double", bounds=dict(min=0, max=1)),
        dict(name="scale_pos_weight", type="int", bounds=dict(min=0, max=10)),
    ],
    type="offline",
    observation_budget=100,
)
print("Created experiment: https://sigopt.com/experiment/" + experiment.id)

while experiment.progress.observation_count < experiment.observation_budget:
    suggestion = conn.experiments(experiment.id).suggestions().create()
    observation = main(suggestion.assignments)
    conn.experiments(experiment.id).observations().create(suggestion=suggestion.id, values=observation)
    experiment = conn.experiments(experiment.id).fetch()

print("done")
best_assignments = conn.experiments(experiment.id).best_assignments().fetch()
print(best_assignments)
```

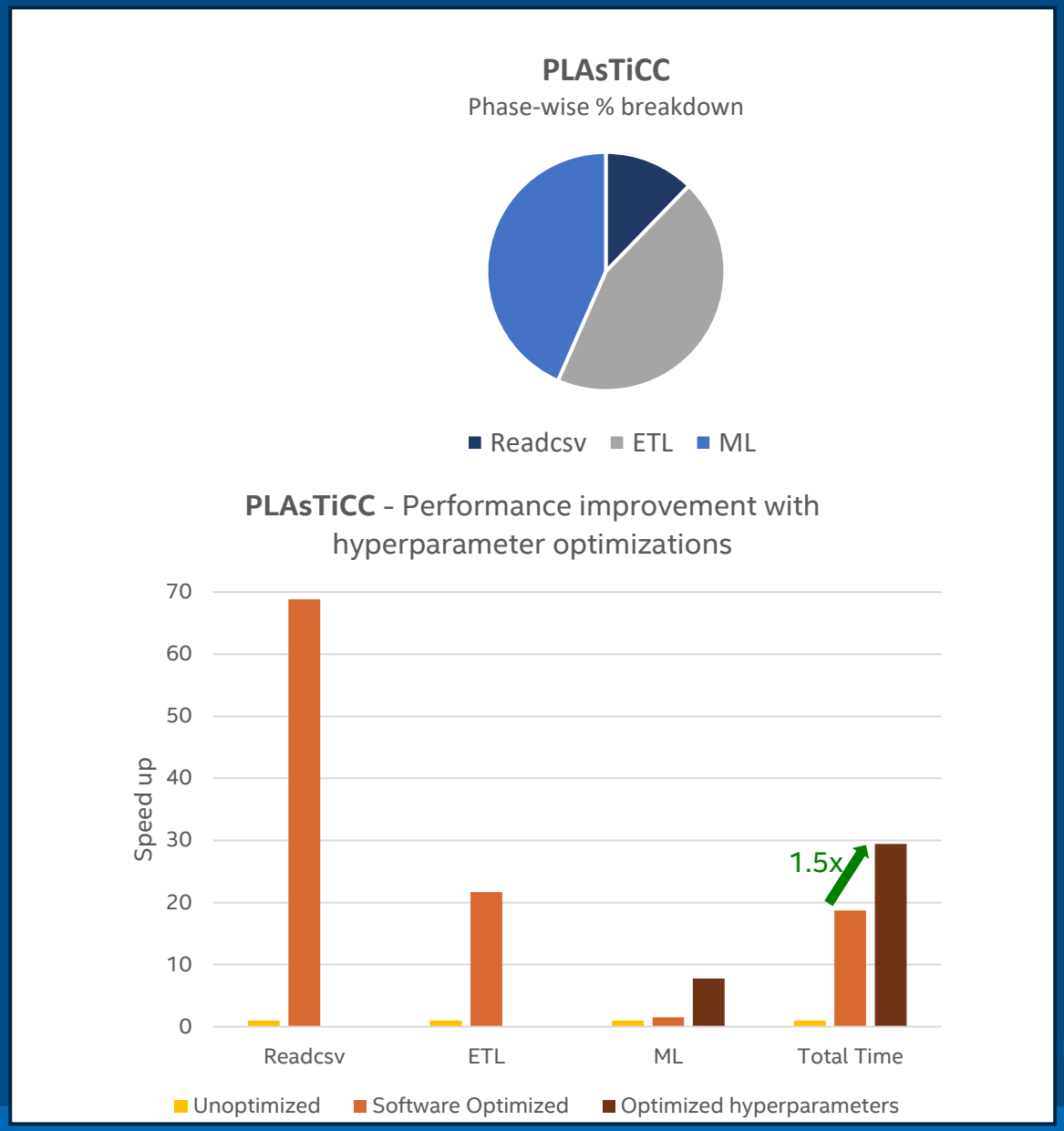
Tunable parameters

	Ridge Regression Parameters	meaning
1	alpha	L2 normalization, larger, weight decreasing faster, avoid overfitting
2	n_round	cross validation, avoid overfitting

	XGBoost Parameters	Meaning
1	tree_method	tree construction algorithm: exact, approx, hist
2	Nthread	number of threads
3	nround	number of trees
4	subsample	randomly sample training data before build tree to avoid overfitting
5	colsample_bytree	randomly sample features before building tree to avoid overfitting
6	max_bin	used to bucket continuous features, trade off between accuracy and timing
7	max_depth	larger, learned more local , easier to overfitting
8	min_child_weight	split tree if weight larger than this value, larger, less leaf, avoid overfitting
9	learning_rate (eta)	shrinks the feature weights
10	gamma	larger, more conservative when split, could avoid overfitting
11	(reg_) alpha	L1 normalization, larger, weight decreasing faster, avoid overfitting
12	(reg_) lambda	L2 normalization, larger, weight decreasing faster, avoid overfitting
13	scale_pos_weight	control the balance of positive and negative weights for unbalanced class

End-to-end workload performance with optimized hyperparameters

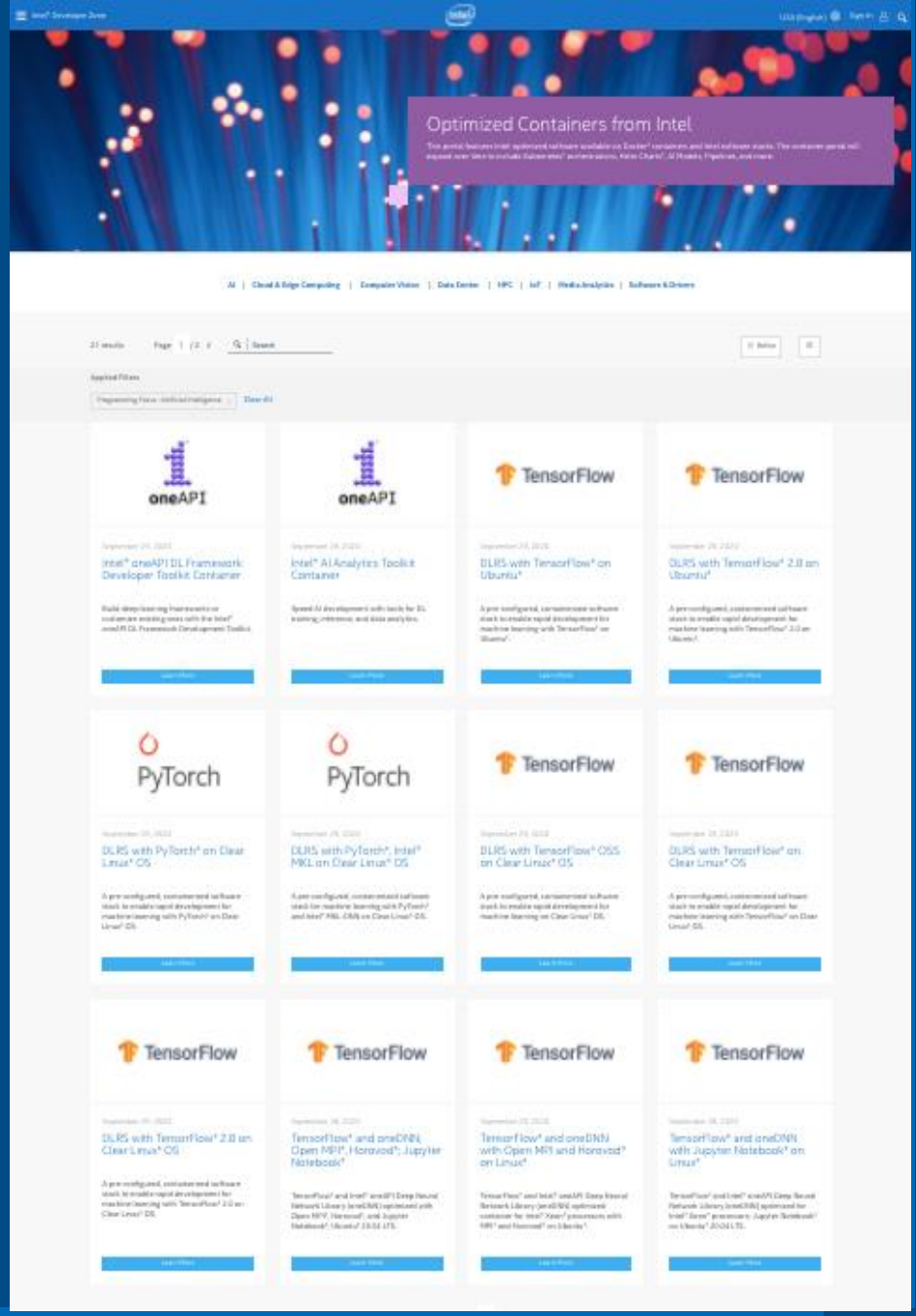




Higher is better

Intel oneAPI Containers

- Optimized, validated, deployable AI containers and artifacts for Intel platforms
- Download Docker* Image: from the Containers Repository
- Available via Docker containers and Intel software stacks. Easy access to Kubernetes orchestrations, Helm charts, AI Models, Pipelines, Pre-trained weights ready for training and inference, and more
- Link to the portal (*https://software.intel.com/containers*)



Key Models (More being added)

Image Classification
ResNet-50
ResNeXt-101

Image Segmentation
Mask R-CNN
3D-UNet

Object Detection
SSD-MobileNet-v1
SSD-ResNet34
Mobilenet-v1
Yolo-v4

Text to Speech
WaveNet

Recommendation
DLRM
DIEN
NCF
Wide & Deep

Language Modeling & Translation
BERT-Large
Transformer-LT

Speech Recognition
RNN-T

- oneAPI Containers Get Started Guide
- Intel® oneAPI Base Toolkit
- Intel® oneAPI HPC Toolkit
- Intel® oneAPI IoT Toolkit
- Intel® oneAPI DL Framework Developer Toolkit
- Intel® AI Analytics Toolkit
- Intel® oneAPI Runtime Libraries
- https://github.com/intel/oneapi-containers
- Singularity containers

Configuration Details

NYCTaxi Workload performance:

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

Scikit-learn Bench and XGBoost Workload performance :

Intel CPU: Intel Xeon Platinum 8280 CPU @ 2.70GHz, 2 sockets, 28 cores per socket; AMD CPU: Rome 7742 @2.25GHz, 2 sockets, 64 cores per socket
Nvidia GPU: Tesla V100-16GB SW: daal4py 2020.3, Scikit-learn 0.23.2, RAPIDS cuML 0.15

End-to-End Census Workload performance (Stock):

Tested by Intel as of 2/19/2021. 2 x Intel® Xeon Platinum 8280L @ 28 cores, OS: Ubuntu 20.04.1 LTS Mitigated, 384GB RAM (384GB RAM: 12x 32GB 2933MHz), kernel: 5.4.0-65-generic, microcode: 0x4003003, CPU governor: performance. SW: Scikit-learn 0.24.1, Pandas 1.2.2, Python 3.9.7, 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://doi.org/10.18128/D010.V10.0]*

End-to-End Census Workload performance (Optimized):

Tested by Intel as of 2/19/2021. 2 x Intel® Xeon Platinum 8280L @ 28 cores, OS: Ubuntu 20.04.1 LTS Mitigated, 384GB RAM (384GB RAM: 12x 32GB 2933MHz), kernel: 5.4.0-65-generic, microcode: 0x4003003, CPU governor: performance. SW: Scikit-learn 0.24.1 accelerated by daal4py 2021.2, modin 0.8.3, omniscidbe v5.4.1, Python 3.9.7, 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://doi.org/10.18128/D010.V10.0]*

End-to-End PLAsTiCC Workload performance (Stock):

Tested by Intel as of 2/19/2021. 2 x Intel® Xeon Platinum 8280L @ 28 cores, OS: Ubuntu 20.04.1 LTS Mitigated, 384GB RAM (384GB RAM: 12x 32GB 2933MHz), kernel: 5.4.0-65-generic, microcode: 0x4003003, CPU governor: performance. SW: Scikit-learn 0.24.1, Pandas 1.2.2, XGBoost 1.3.3, Python 3.9.7, PLAsTiCC Data Training set: (1421705, 6); Test set: (189022127, 6). Dataset is from Kaggle challenge *"PLAsTiCC Astronomical Classification"* https://www.kaggle.com/c/PLAsTiCC-2018/data

End-to-End PLAsTiCC Workload performance (Optimized):

Tested by Intel as of 2/19/2021. 2 x Intel® Xeon Platinum 8280L @ 28 cores, OS: Ubuntu 20.04.1 LTS Mitigated, 384GB RAM (384GB RAM: 12x 32GB 2933MHz), kernel: 5.4.0-65-generic, microcode: 0x4003003, CPU governor: performance. SW: Scikit-learn 0.24.1 accelerated by daal4py 2021.2, modin 0.8.3, omniscidbe v5.4.1, XGBoost 1.3.3, Python 3.9.7, PLAsTiCC Data Training set: (1421705, 6); Test set: (189022127, 6). Dataset is from Kaggle challenge *"PLAsTiCC Astronomical Classification"* https://www.kaggle.com/c/PLAsTiCC-2018/data


intel

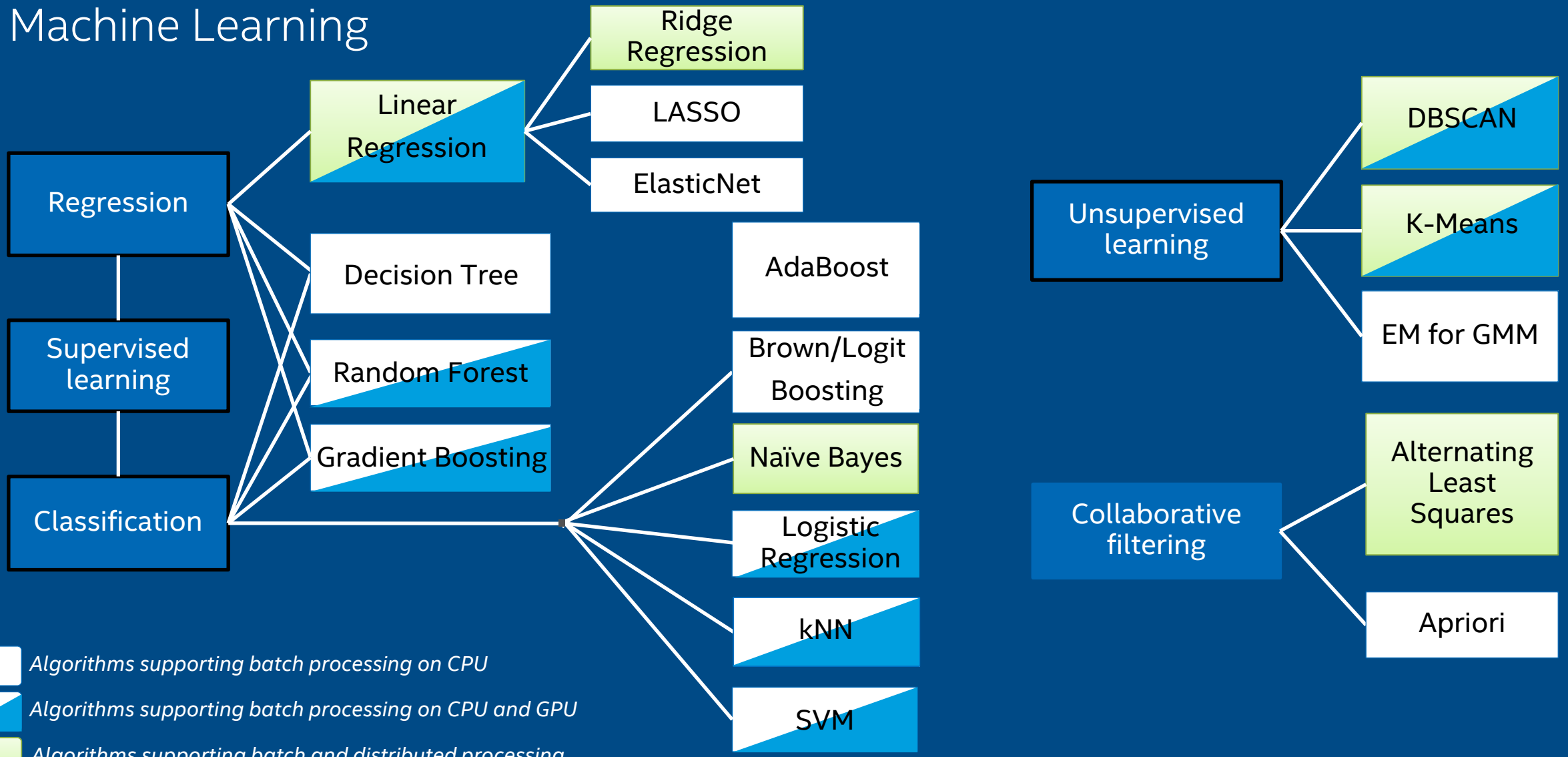
Getting Started with Intel® oneAPI AI Analytics Toolkit

Overview	Installation	Hands on	Learning	Support
▪ Visit Intel® oneAPI AI Analytics Toolkit (AI Kit) for more details and up-to-date product information ▪ Release Notes	▪ Download the AI Kit from Intel, Anaconda or any of your favorite package managers ▪ Get started quickly with the AI Kit Docker Container ▪ Installation Guide ▪ Utilize the Getting Started Guide	▪ Code Samples ▪ Build, test and remotely run workloads on the Intel® DevCloud for free. No software downloads. No configuration steps. No installations.	▪ Machine Learning & Analytics Blogs at Intel Medium ▪ Intel AI Blog site ▪ Webinars and Articles at Intel® Tech Decoded	▪ Ask questions and share information with others through the Community Forum ▪ Discuss with experts at AI Frameworks Forum

Download Now

oneDAL Algorithms

Machine Learning

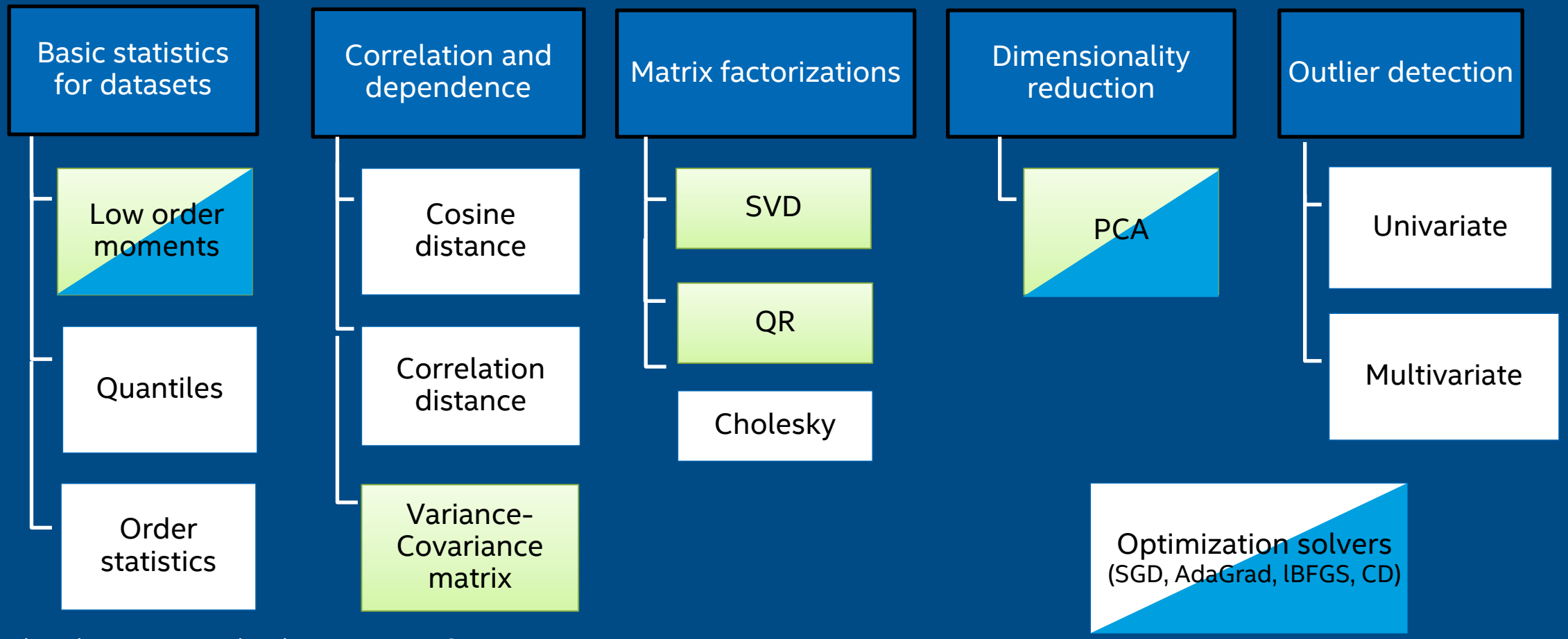


Algorithms supporting batch processing on CPU

Algorithms supporting batch processing on CPU and GPU

Algorithms supporting batch and distributed processing

oneDAL Algorithms
Data Transformation and Analysis



Algorithms supporting batch processing on CPU

Algorithms supporting batch processing on CPU and GPU

Algorithms supporting batch and distributed processing

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