

Intel® AI Workshop
16.02.2022

Intel® oneAPI AI tools

Shailen Sobhee - Senior AI Solution Engineer
- HPC Software Architect for Exascale
- (shailen.sobhee@intel.com)



Notices and Disclaimers

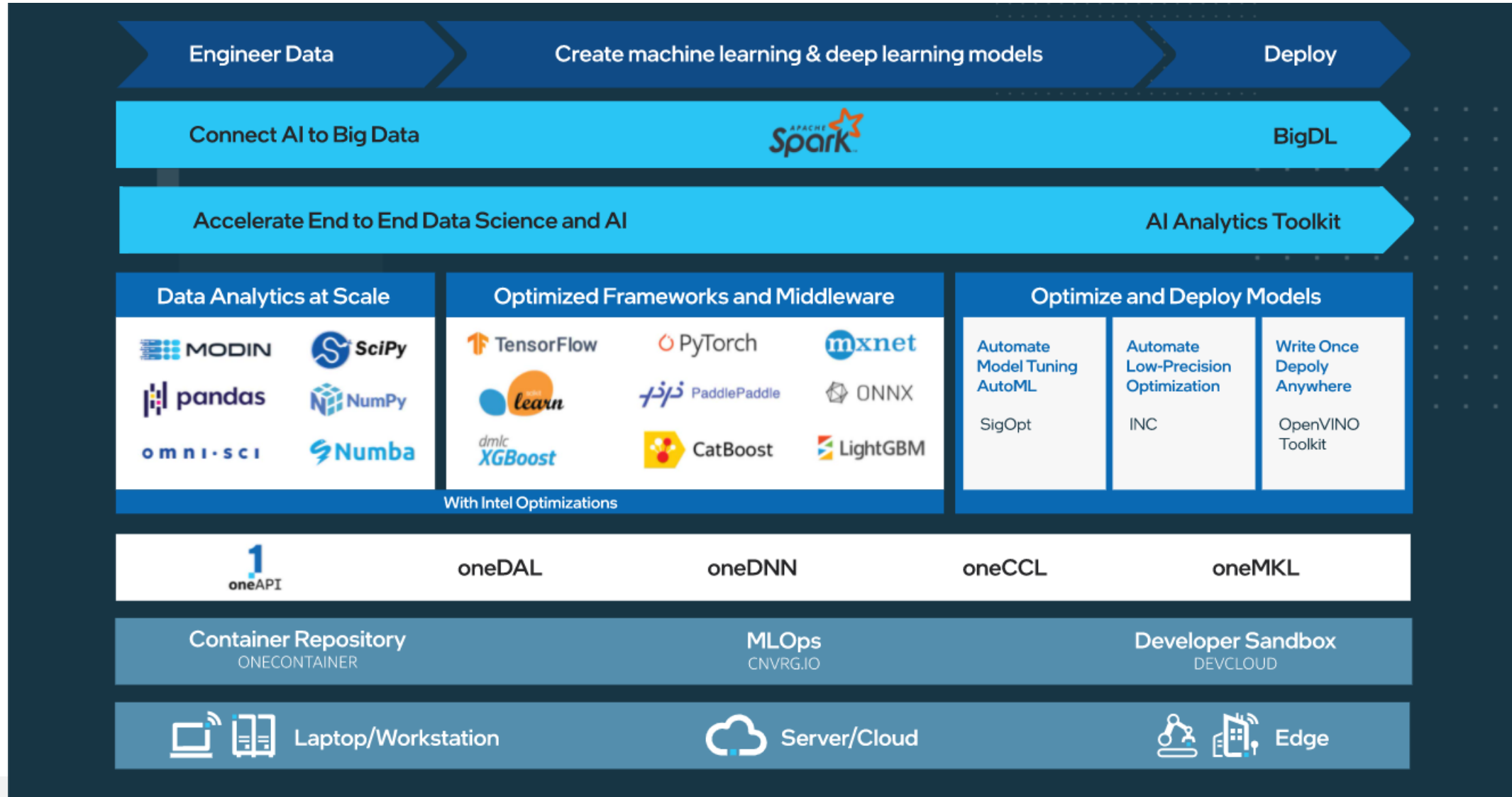
- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration.
- No product or component can be absolutely secure.
- Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. For more complete information about performance and benchmark results, visit <http://www.intel.com/benchmarks>.
- Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit <http://www.intel.com/benchmarks>.
- Intel® Advanced Vector Extensions (Intel® AVX) provides higher throughput to certain processor operations. Due to varying processor power characteristics, utilizing AVX instructions may cause a) some parts to operate at less than the rated frequency and b) some parts with Intel® Turbo Boost Technology 2.0 to not achieve any or maximum turbo frequencies. Performance varies depending on hardware, software, and system configuration and you can learn more at <http://www.intel.com/go/turbo>.
- Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.
- Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.
- Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.
- © Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Agenda

- AI Software Stack for XPU
- Intel® oneAPI AI Analytics toolkit
- oneAPI Tools for Machine Learning
- oneAPI Tools for Deep Learning

AI Software Stack for Intel® XPUs

Intel offers a Robust Software Stack to Maximize Performance of Diverse Workloads

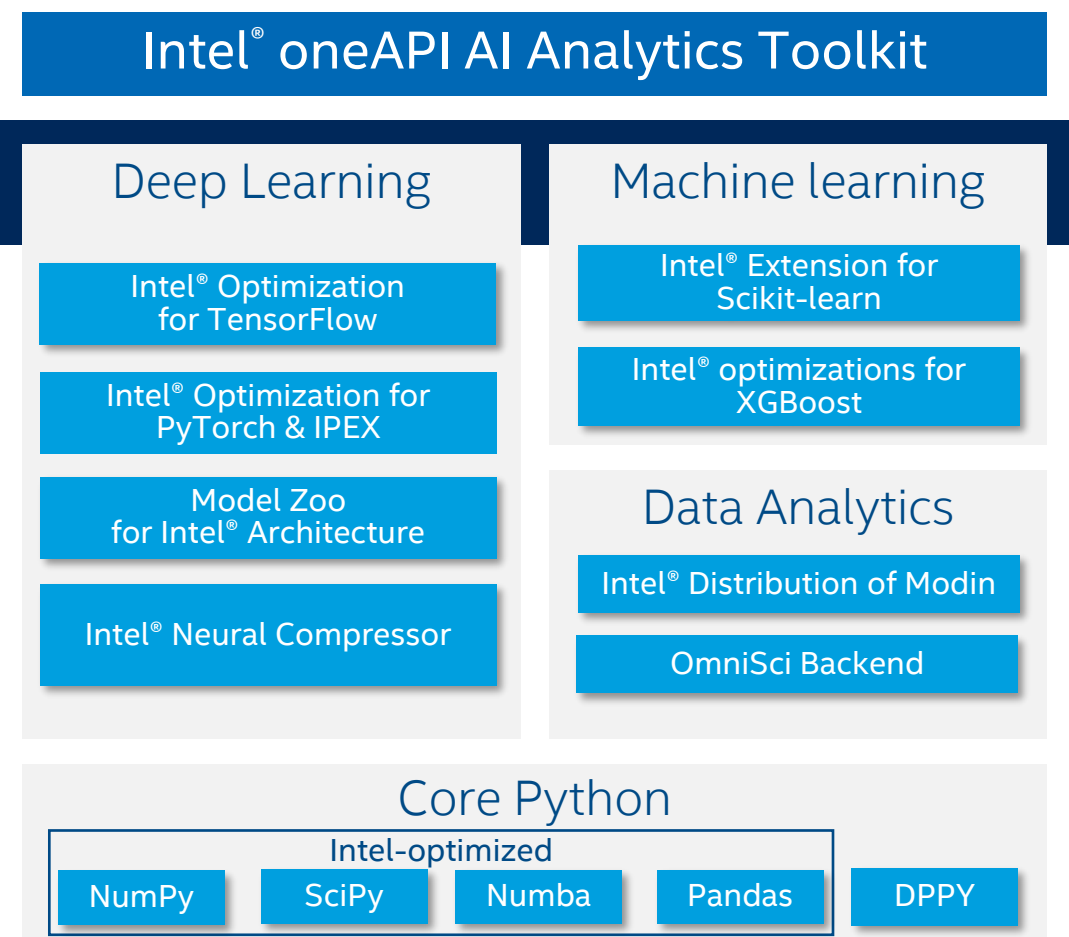


Intel® oneAPI AI Analytics Toolkit

- 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

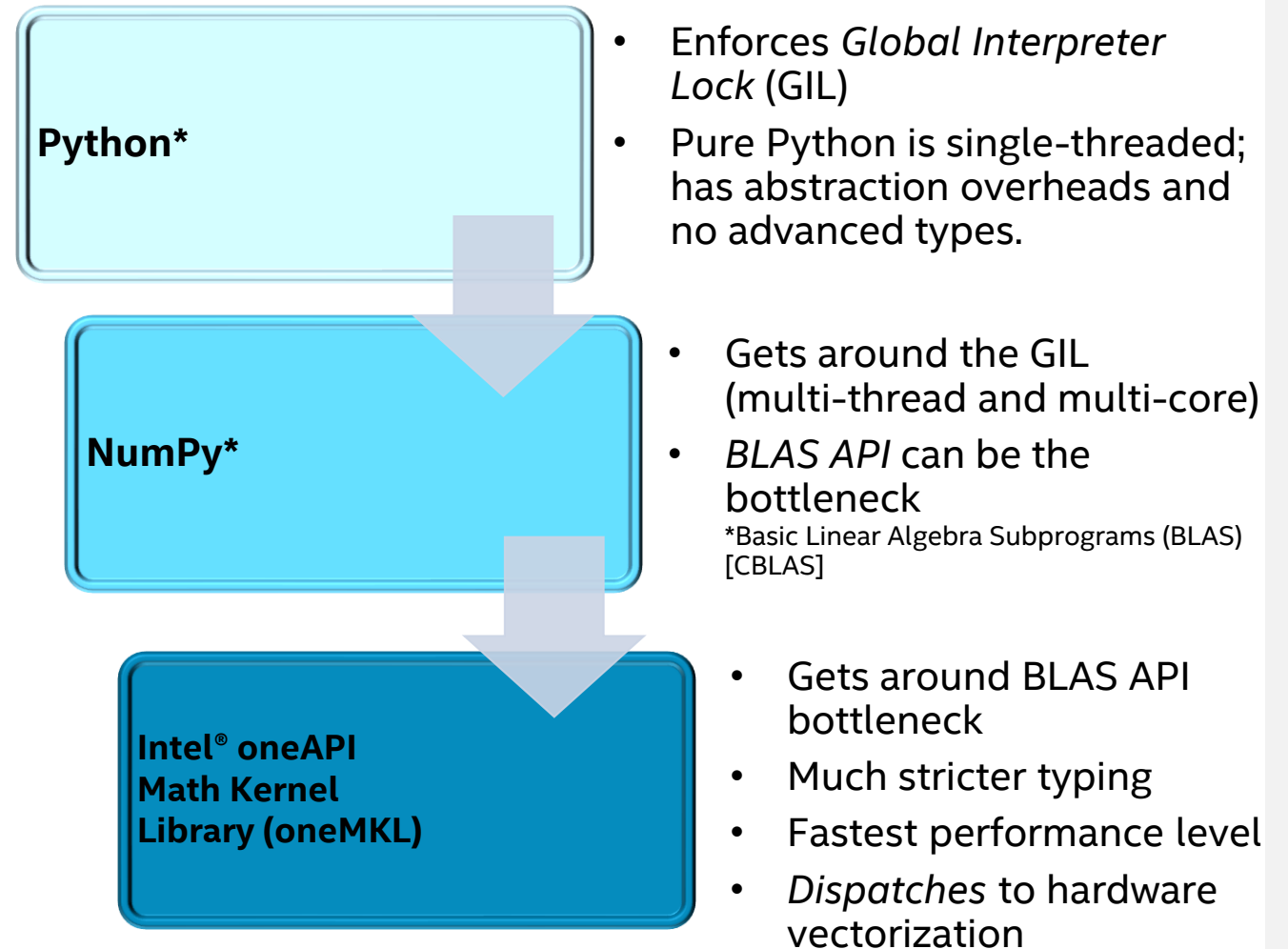


Learn More: intel.com/oneAPI-AIKit

Introduction to Python* Performance

■ 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



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

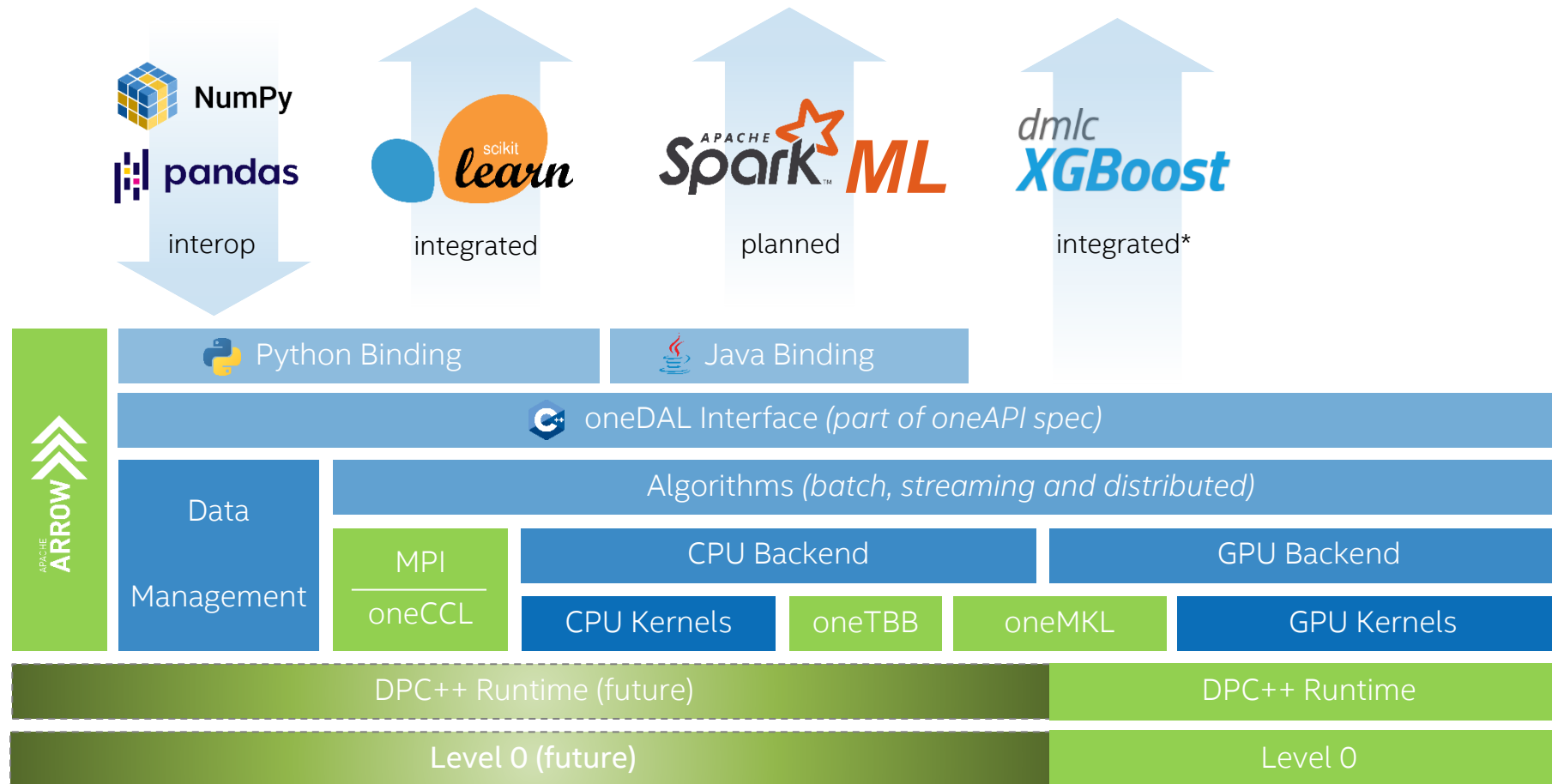
Accelerating Machine Learning with Intel® oneAPI Analytics Toolkit

Intel® oneAPI Data Acceleration Library (Intel® oneDAL)



Intel®oneAPI Data Analytics Library (oneDAL)

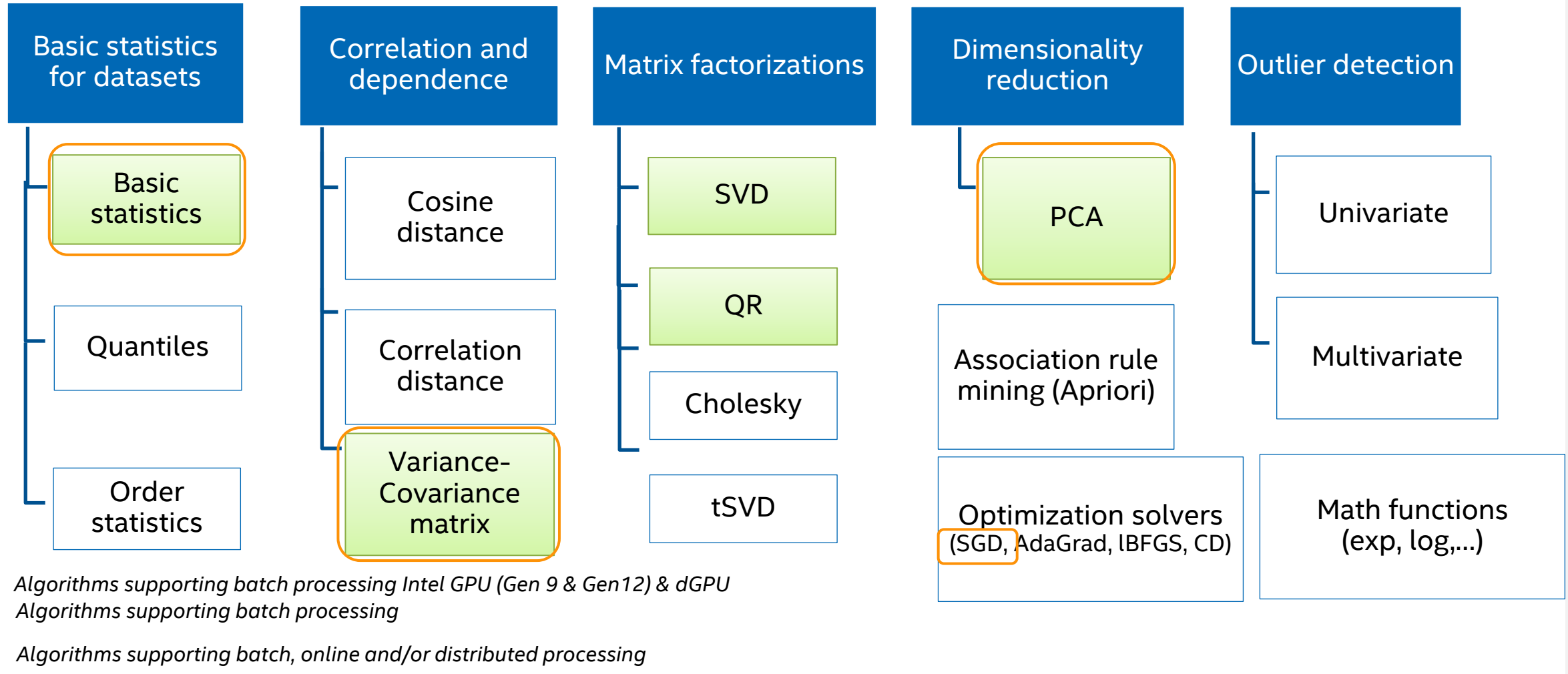
Framework Interfaces & Software Stack



API designed to be hardware and vendor independent
Relies on Data Parallel C++ (DPC++) and C++17

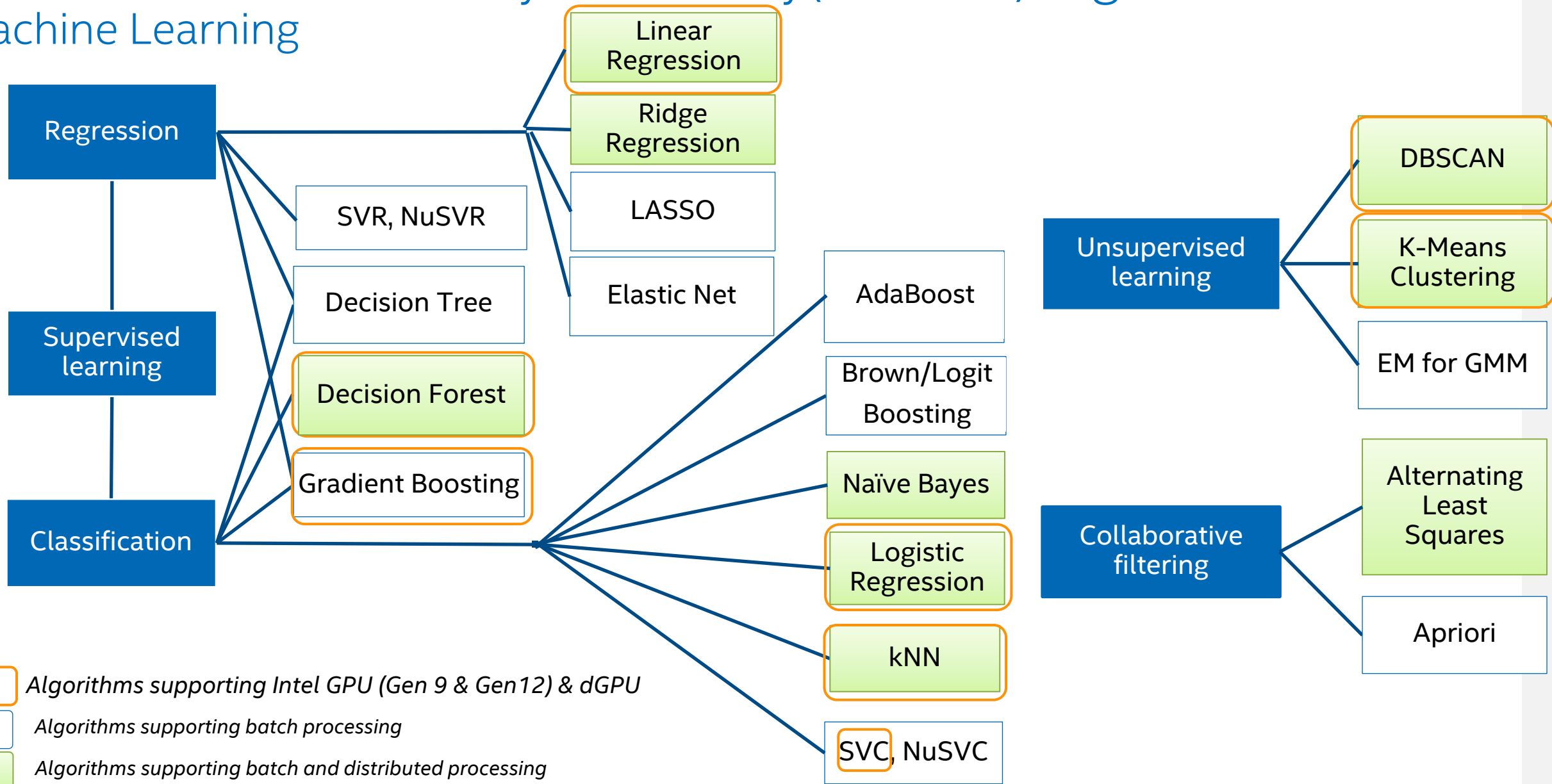
Intel® oneAPI Data Analytics Library (oneDAL) algorithms

Data Transformation and Analysis

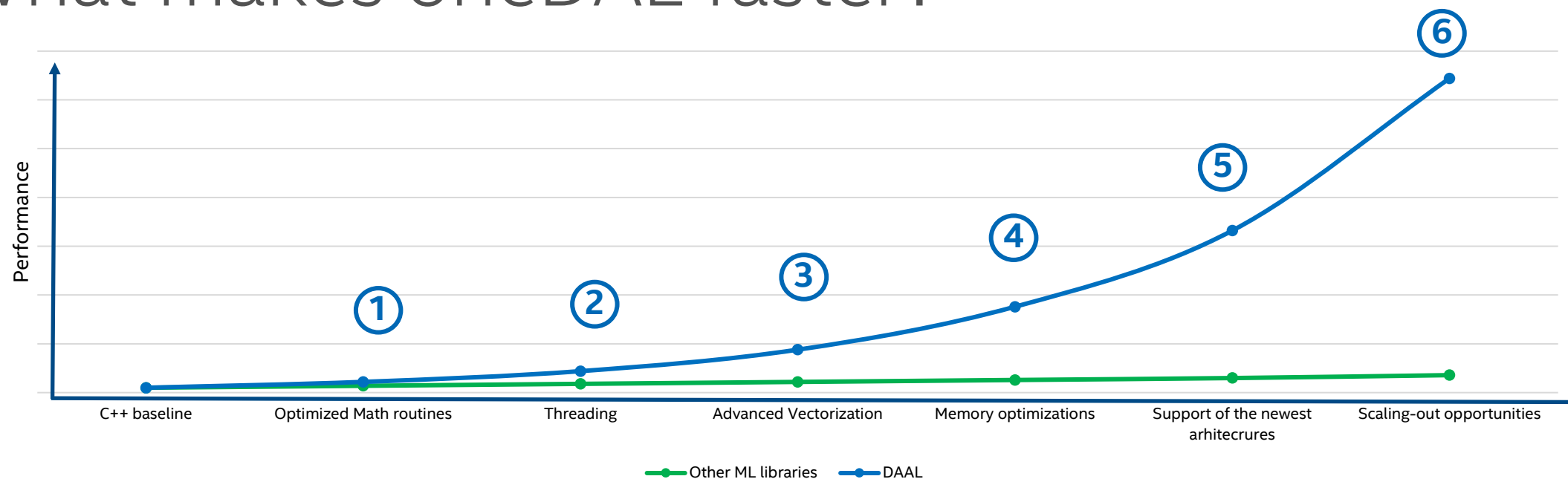


Intel® oneAPI Data Analytics Library(oneDAL) Algorithms

Machine Learning



What makes oneDAL faster?



1 The best performance on Intel Architectures with oneMKL vs. less performance OS 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, intrinsics. Usually other ML libs build application without vector-instructions support or sse4.2 only.

4 oneDAL's 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® 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

- Directly from the script:

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

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)
```

Intel® extension for sklearn

- Through [global patching](#) to enable patching for your scikit-learn installation for all further runs:

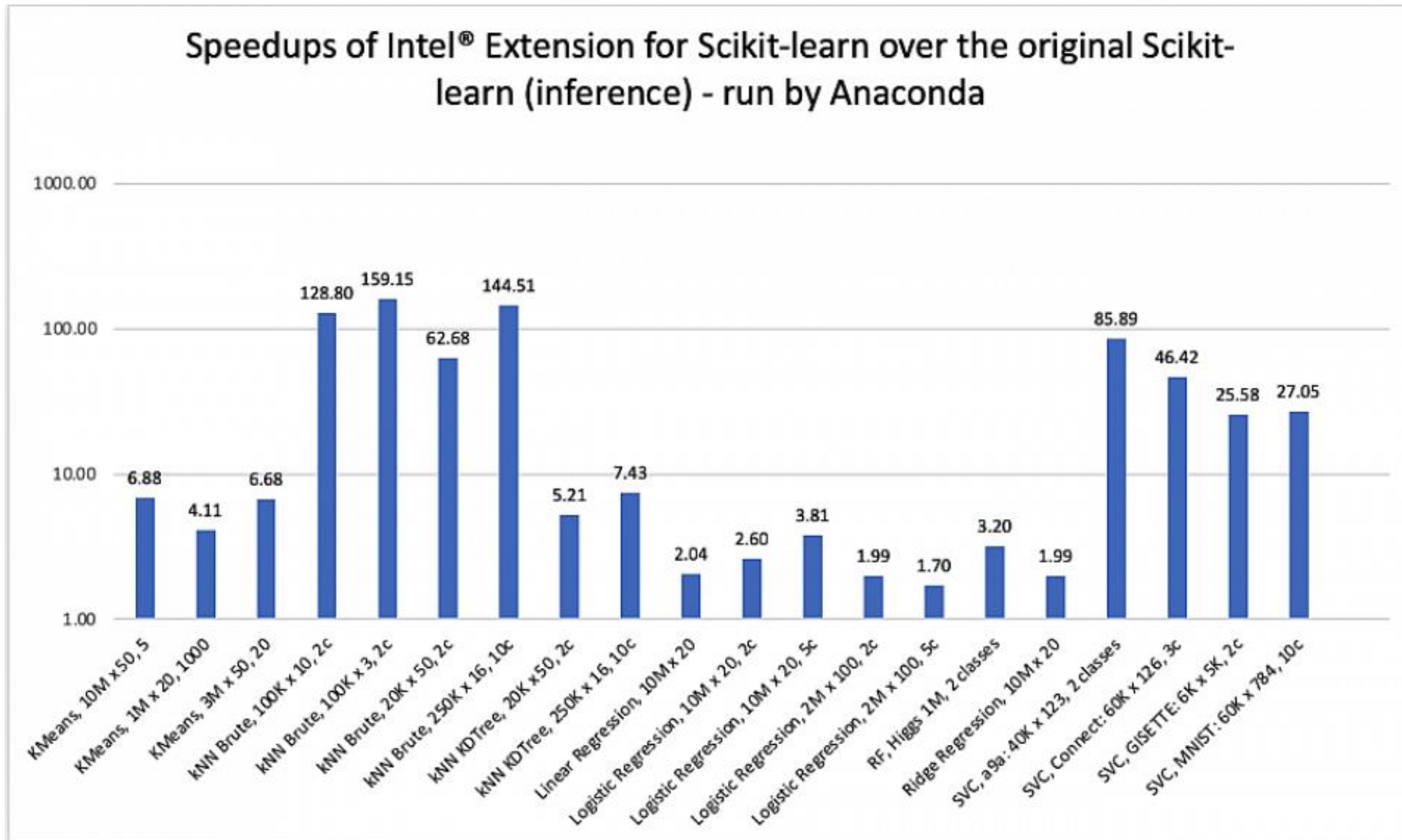
```
python sklearnex.glob patch_sklearn
```

Same Code,
Same Behavior

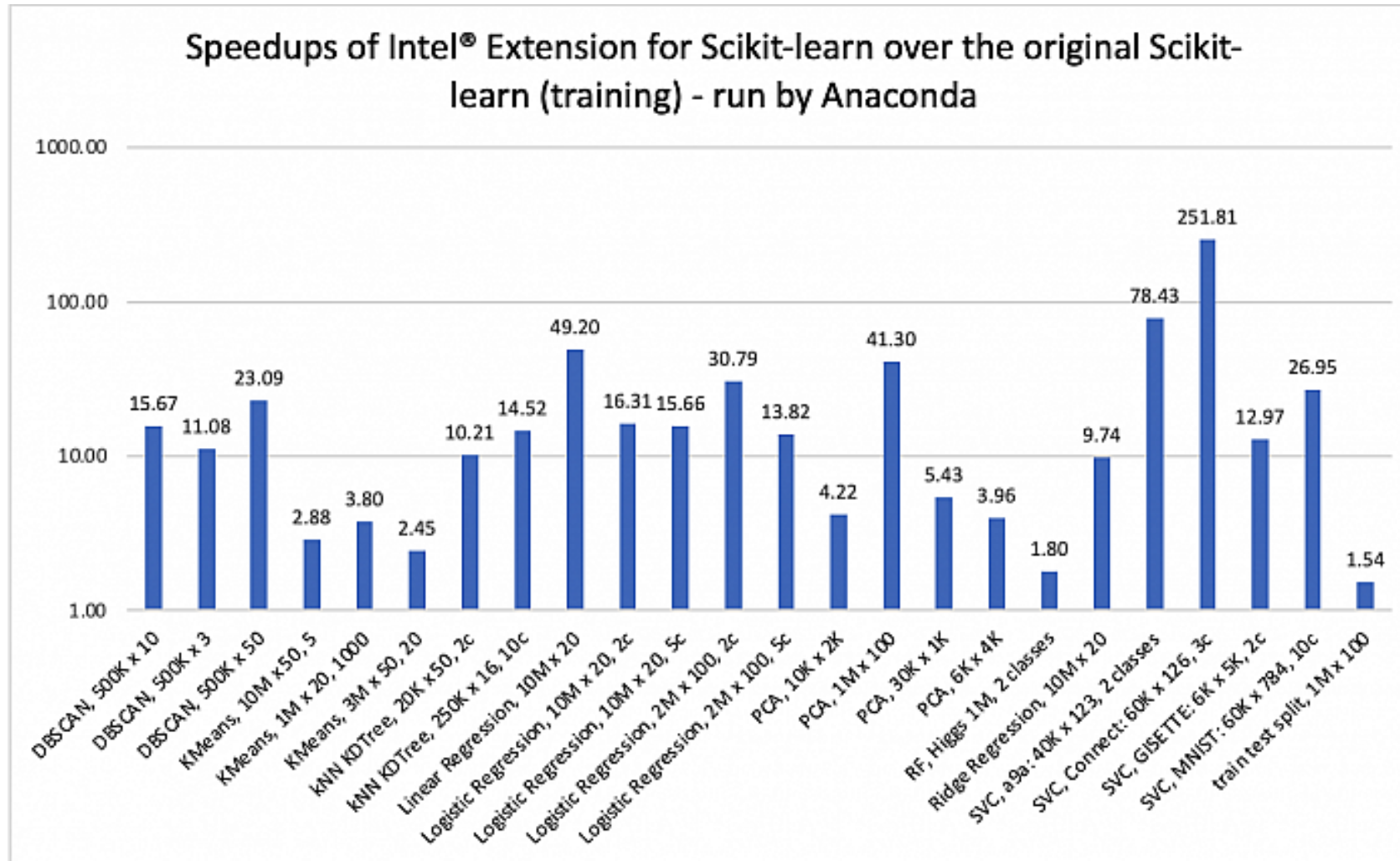
 PASSED

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

Intel® Extension for scikit-learn (Inference)

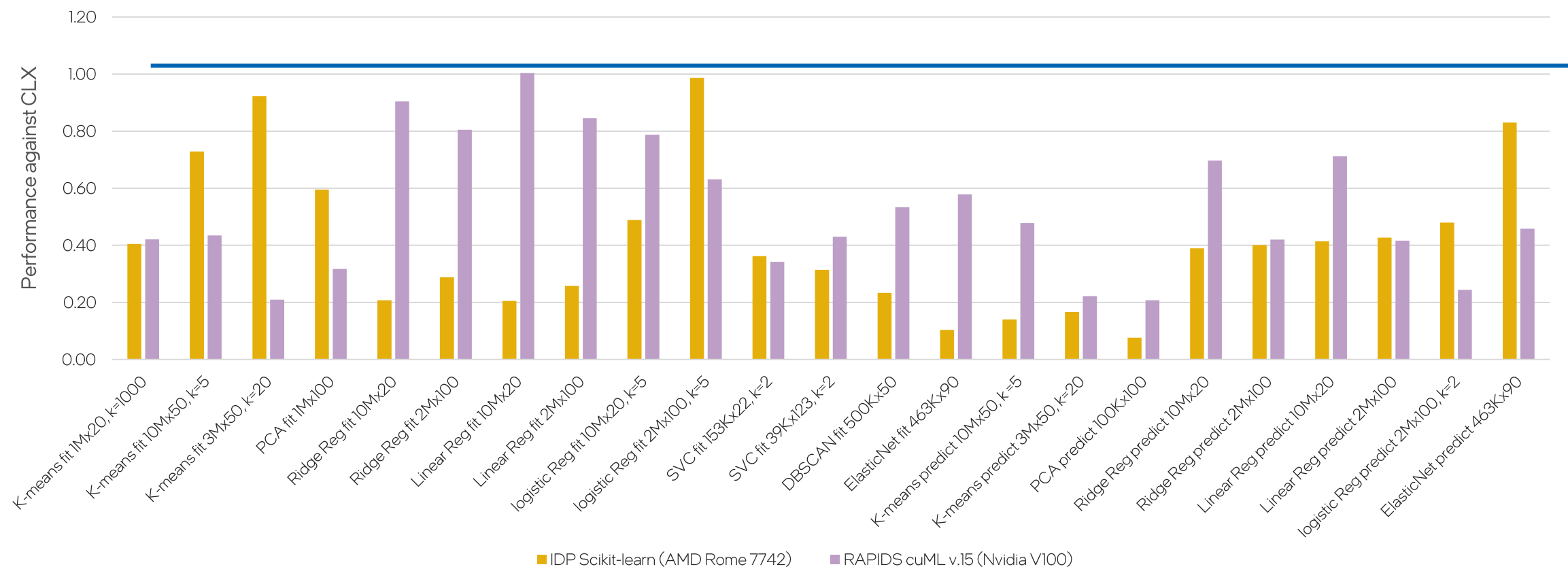


Intel® Extension for scikit-learn (Training)



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.

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. No 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.

oneAPI Data Analytics Library (oneDAL)

Scikit-Learn* **API** Compatible

daal4py

oneDAL

Use directly for

- Scaling to multiple nodes
- Streaming data
- Non-homogeneous dataframes
- Gradient Boosting (for e.g)

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:

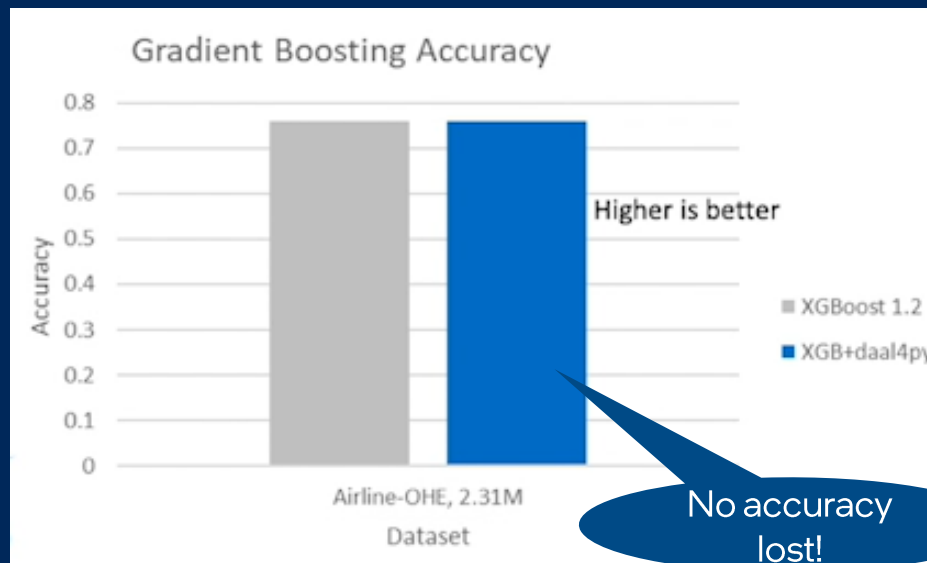
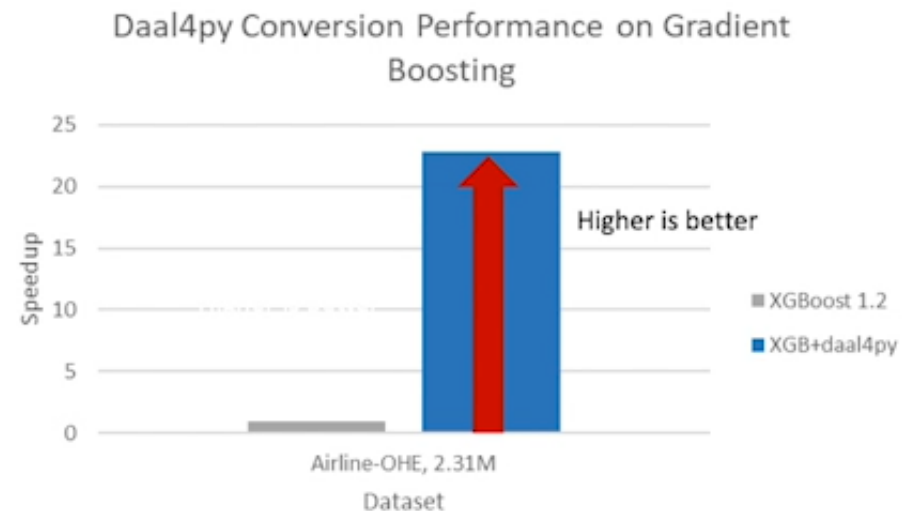
```
# Train common XGBoost model as usual
xgb_model = xgb.train(params, X_train)

import daal4py as d4p

# XGBoost model to DAAL model
daal_model = d4p.get_gbt_model_from_xgboost(xgb_model)

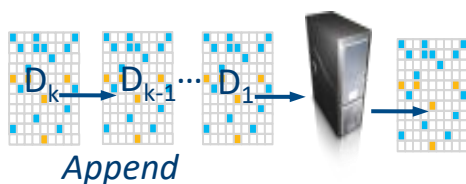
# make fast prediction with DAAL
daal_prediction = d4p.gbt_classification_prediction(...).compute(X_test, daal_model)
```

- Advantages of daal4py GBT model:
 - More efficient model representation in memory
 - AVX-512 instruction set usage
 - Better L1/L2 caches locality



Processing Modes

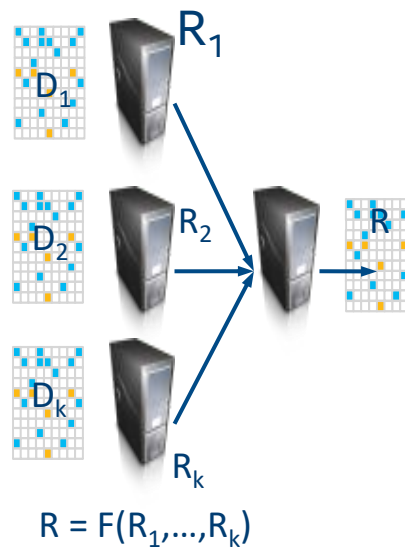
Batch Processing



$$R = F(D_1, \dots, D_k)$$

```
d4p.kmeans_init(10, method="plusPlusDense")
```

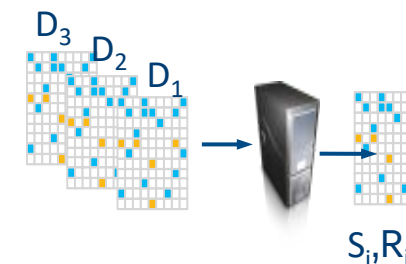
Distributed Processing



$$R = F(R_1, \dots, R_k)$$

```
d4p.kmeans_init(10, method="plusPlusDense",  
distributed="True")
```

Online Processing



$$S_{i+1} = T(S_i, D_i)$$

$$R_{i+1} = F(S_{i+1})$$

```
d4p.kmeans_init(10, method="plusPlusDense",  
streaming="True")
```

Distributed K-Means using daal4py

```
import daal4py as d4p

# initialize distributed execution environment
d4p.daalinit()

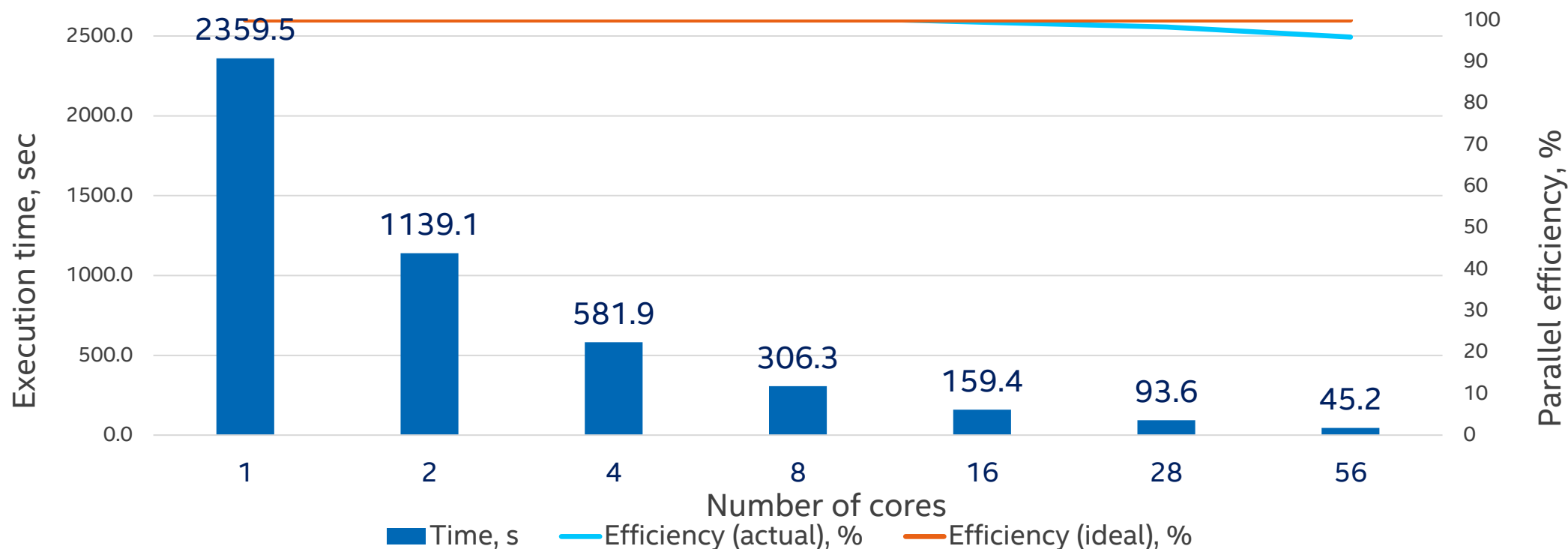
# daal4py accepts data as CSV files, numpy arrays or pandas dataframes
# here we let daal4py load process-local data from csv files
data = "kmeans_dense_{}.csv".format(d4p.my_procid())

# compute initial centroids & kmeans clustering
init = d4p.kmeans_init(10, method="plusPlusDense", distributed=True)
centroids = init.compute(data).centroids
result = d4p.kmeans(10, distributed=True).compute(data, centroids)
```

```
mpirun -n 4 python ./kmeans.py
```

oneDAL K-Means Fit, Cores Scaling

(10M samples, 10 features, 100 clusters, 100 iterations, float32)



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 configuration disclosure for 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.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Configuration: Testing by Intel as of 10/23/2020. Intel® oneAPI Data Analytics Library 2021.1 (oneDAL); Intel® Xeon® Platinum 8280LCPU @ 2.70GHz, 2 sockets, 28 cores per socket, 10M samples, 10 features, 100 clusters, 100 iterations, float32.

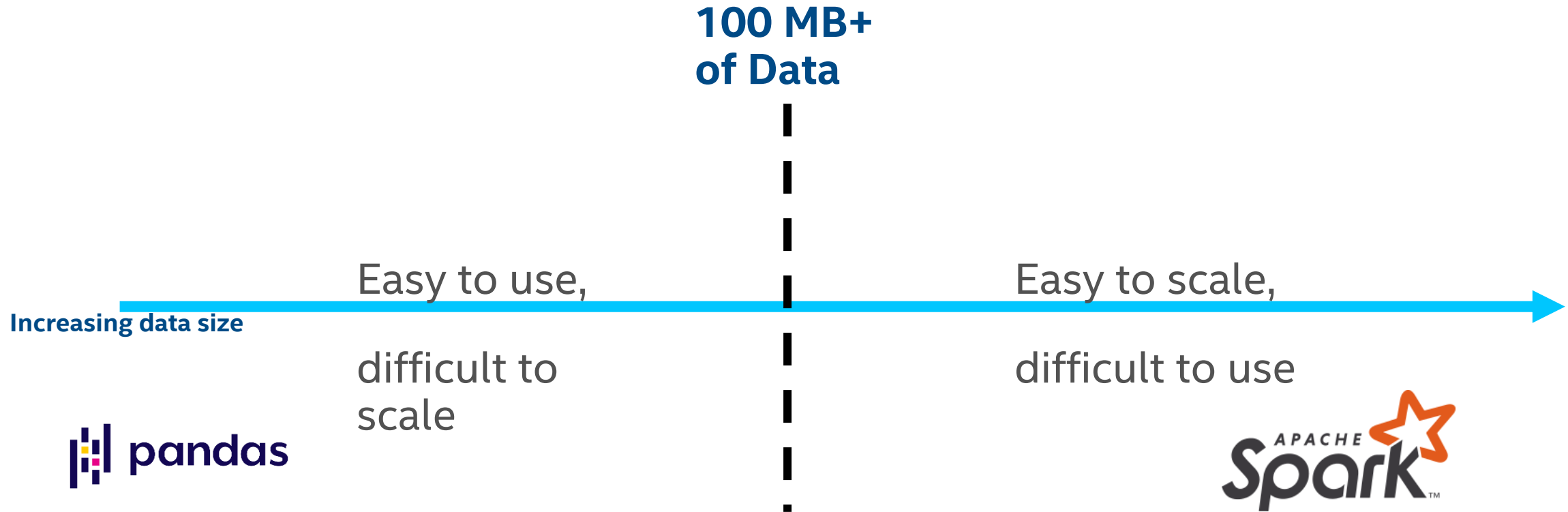
Accelerating Machine Learning with Intel® oneAPI Analytics Toolkit

Intel® Distribution of Modin



Current Data Loading & ETL Landscape

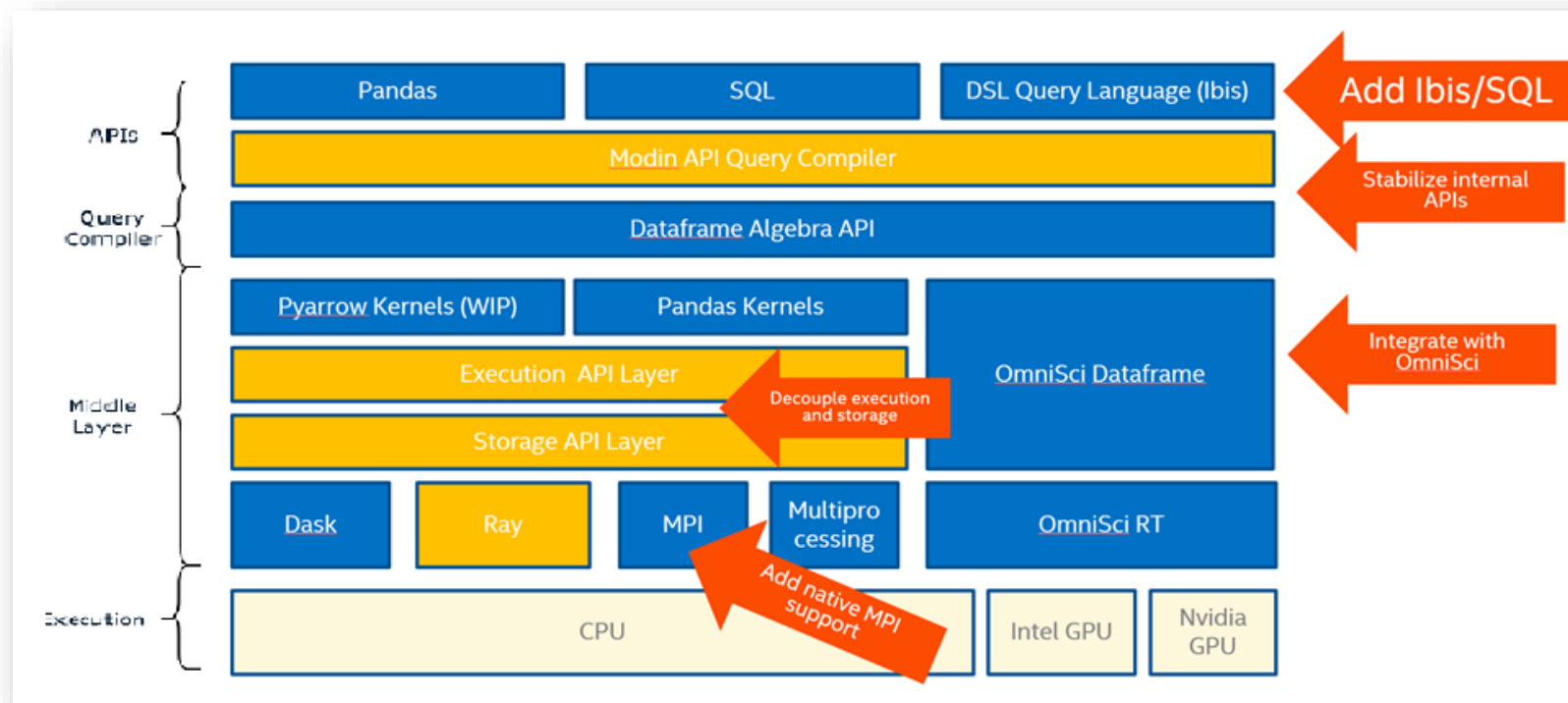
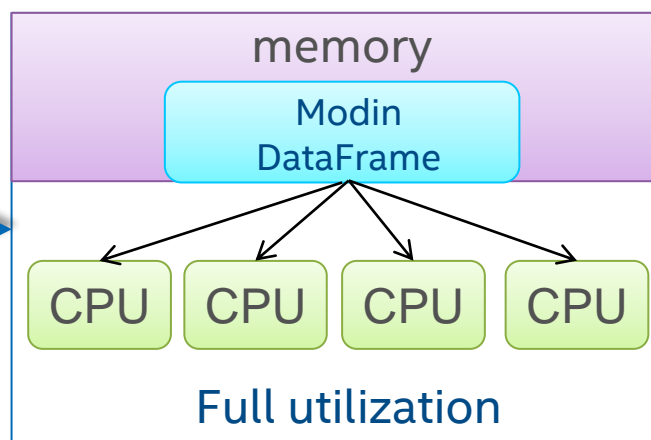
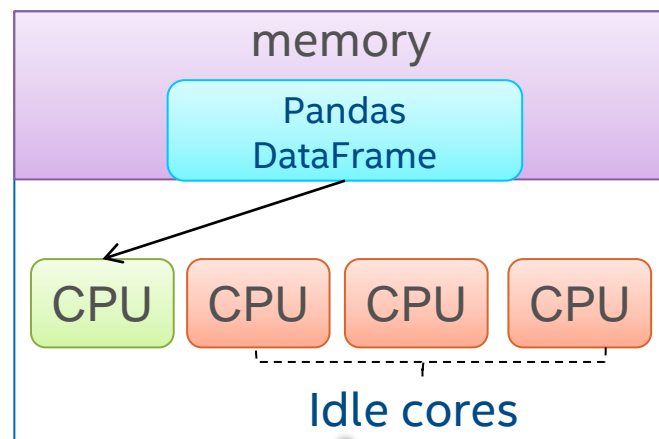
After a certain data size, need to change your API to handle more data



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

Modin

■ Usable and Scalable



To use Modin, replace the pandas import

To install the Intel Distribution of Modin, alter the command to include conda forge dependencies:
`conda create -n aikit-modin intel-aikit-modin -c intel -c conda-forge`

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

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)

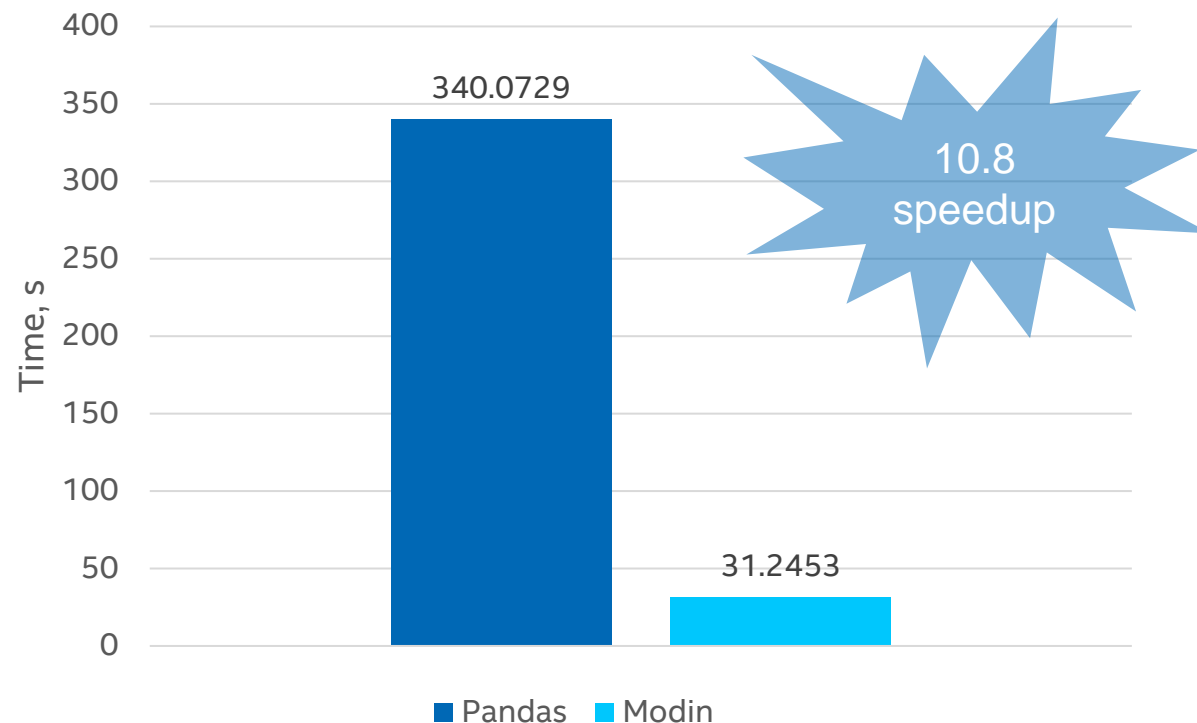
    count_y = df.groupby("column_0")["0"].count()

    return df, count_y

df, count_y = run_etl()
```

- Dataset size: 2.4GB

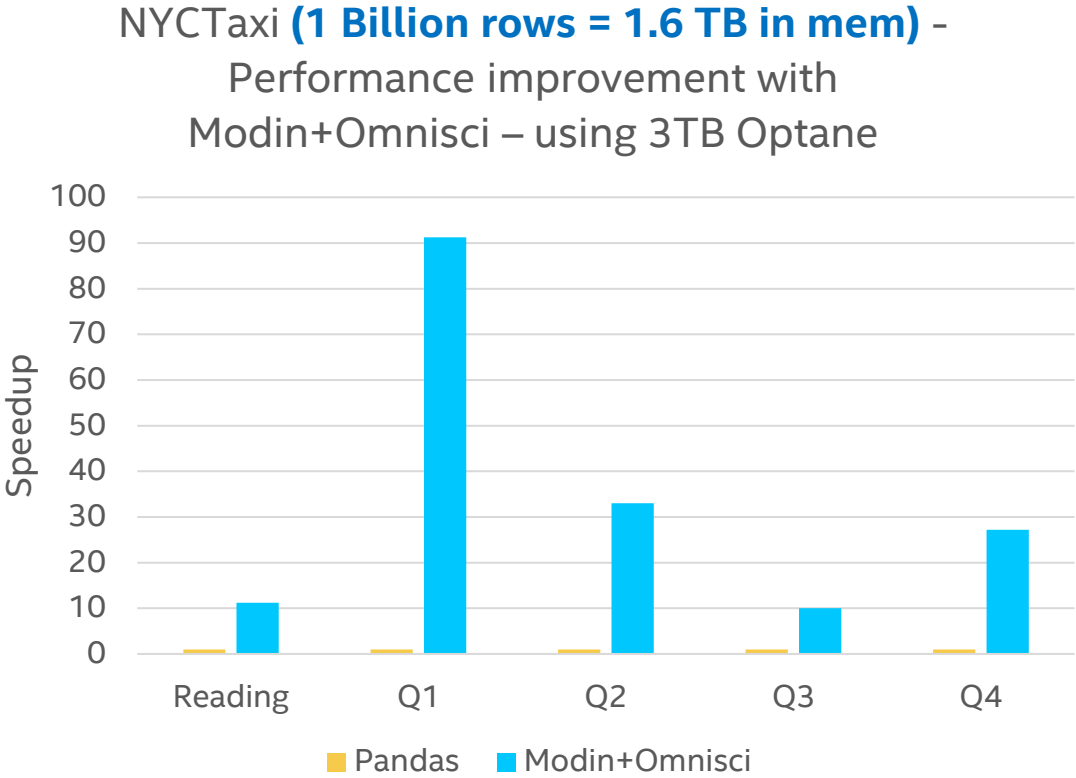
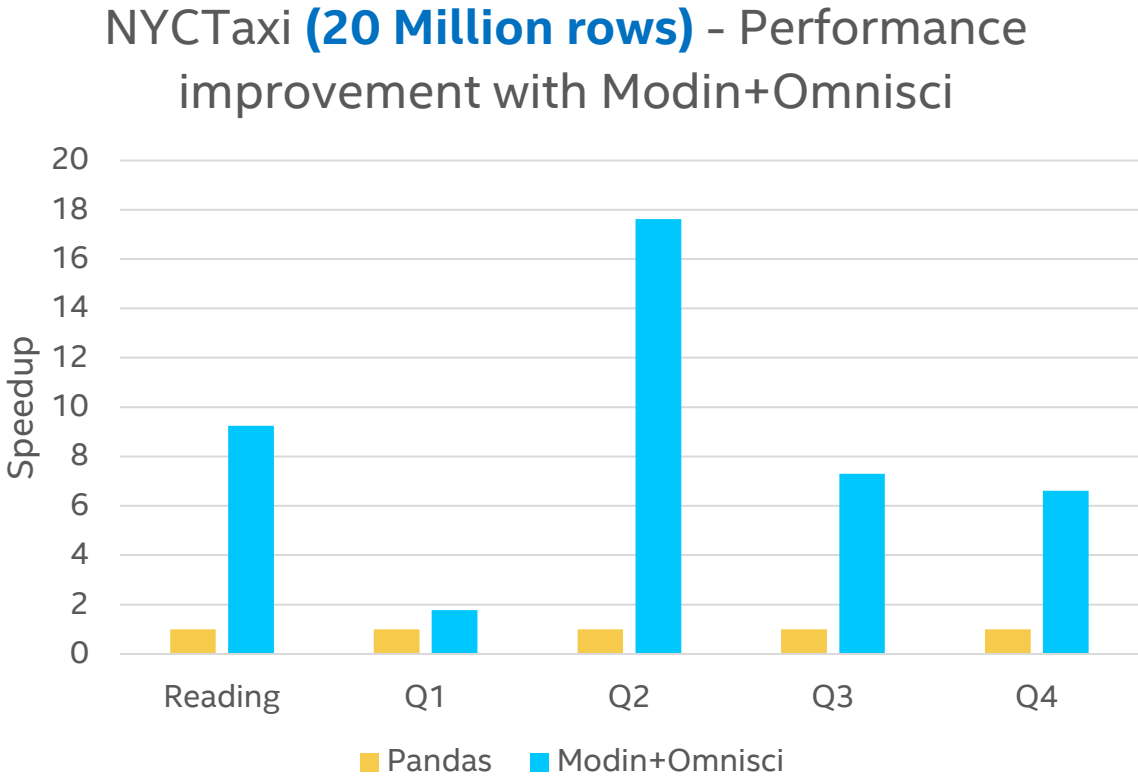
Execution time Pandas vs. Modin[ray]



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

NYCTaxi Workload Performance


Pandas vs Modin – Higher is Better



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

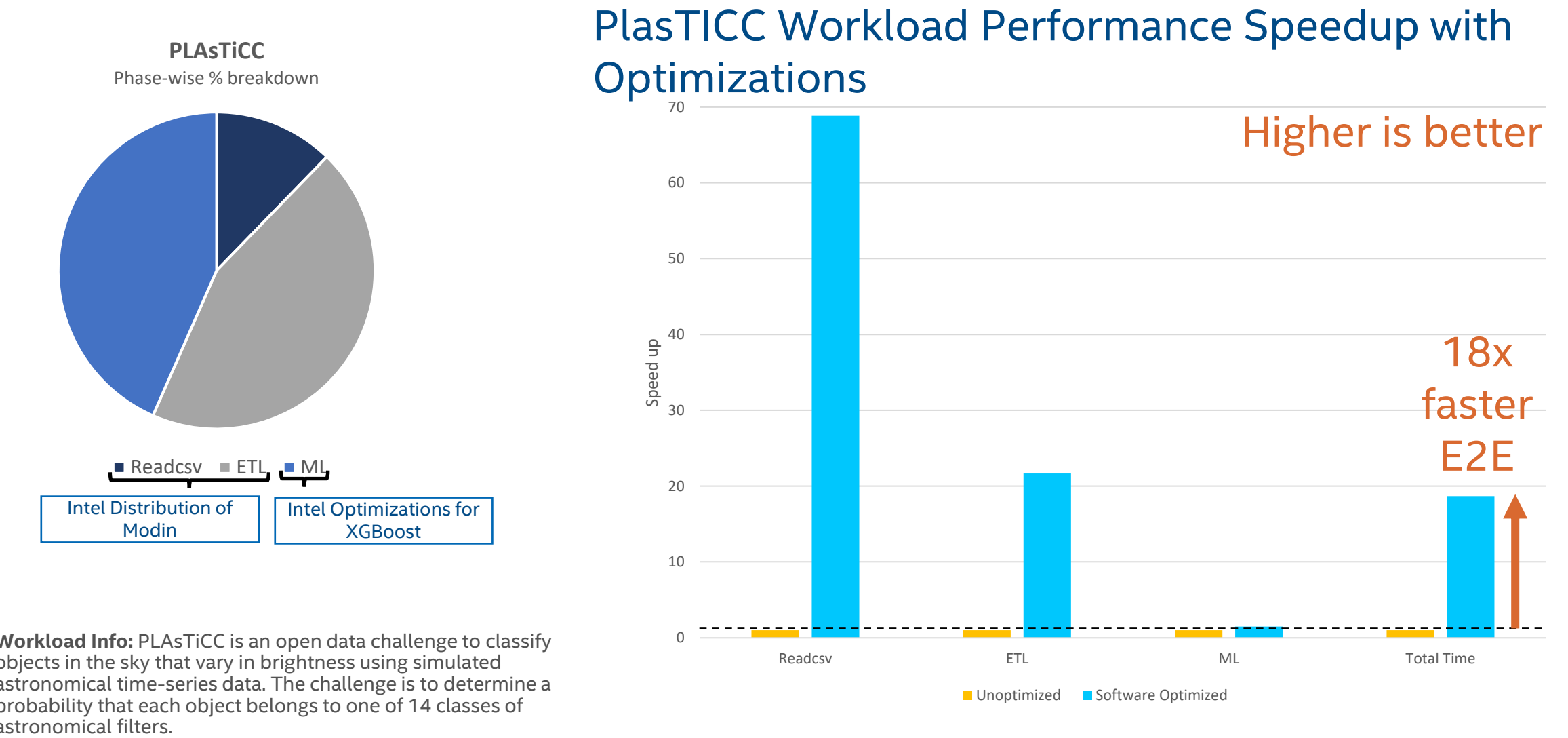
Configurations: For 20 million rows: Dual socket Intel(R) Xeon(R) Platinum 8280L CPUs (S2600WFT platform), 28 cores per socket, hyperthreading enabled, turbo mode enabled, NUMA nodes per socket=2, BIOS: SE5C620.86B.02.01.0013.121520200651, kernel: 5.4.0-65-generic, microcode: 0x4003003, OS: Ubuntu 20.04.1 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

Results have been estimated or simulated and may vary by user configuration and other factors. Learn more at www.intel.com/content/www/us/en/developer/tools/oneapi/appendix-for-configurations.html

 27

Intel Modin + XGBoost E2E workload Intel oneAPI AI toolkit

Combine Intel® oneAPI AI Analytics Toolkit optimizations such as Modin and Intel Optimizations for XGBoost to boost your E2E performance



Develop Fast Neural Networks on Intel® CPUs & GPUs

with Performance-optimized Building Blocks

Accelerating Deep Learning with Intel® oneAPI AI Analytics Toolkit



Develop Fast Neural Networks on Intel® CPUs & GPUs

with Performance-optimized Building Blocks

Intel® oneAPI Deep Neural Network Library (oneDNN)



Intel® oneAPI Deep Neural Network Library (oneDNN)

An **open-source cross-platform** performance library for deep learning applications

- Helps developers create high performance deep learning frameworks
- Abstracts out instruction set and other complexities of performance optimizations
- **Same API for both Intel CPUs and GPUs, use the best technology for the job**
- Supports Linux, Windows and macOS
- Open source for community contributions

More information as well as sources:

<https://github.com/oneapi-src/oneDNN>

Intel® oneAPI Deep Neural Network Library

Basic Information

- Features
 - API: C, C++, SYCL
 - Training: float32, bfloat16⁽¹⁾
 - Inference: float32, bfloat16⁽¹⁾, float16⁽¹⁾, and int8⁽¹⁾
 - MLPs, CNNs (1D, 2D and 3D), RNNs (plain, LSTM, GRU)
- Support Matrix
 - Compilers: Intel, GCC, CLANG, MSVC, DPC++
 - OS: Linux, Windows, macOS
 - CPU
 - Hardware: Intel® Atom, Intel® Core™, Intel® Xeon™
 - Runtimes: OpenMP, TBB, DPC++
 - GPU
 - Hardware: Intel HD Graphics, Intel® Iris® Plus Graphics
 - Runtimes: OpenCL, DPC++

	Intel® oneDNN
Convolution	2D/3D Direct Convolution/Deconvolution, Depthwise separable convolution 2D Winograd convolution
Inner Product	2D/3D Inner Production
Pooling	2D/3D Maximum 2D/3D Average (include/exclude padding)
Normalization	2D/3D LRN across/within channel, 2D/3D Batch normalization
Eltwise (Loss/activation)	ReLU(bounded/soft), ELU, Tanh; Softmax, Logistic, linear; square, sqrt, abs, exp, gelu, swish
Data manipulation	Reorder, sum, concat, View
RNN cell	RNN cell, LSTM cell, GRU cell
Fused primitive	Conv+ReLU+sum, BatchNorm+ReLU
Data type	f32, bfloat16, s8, u8

(1) Low precision data types are supported only for platforms where hardware acceleration is available

Overview of Intel® optimizations for TensorFlow*



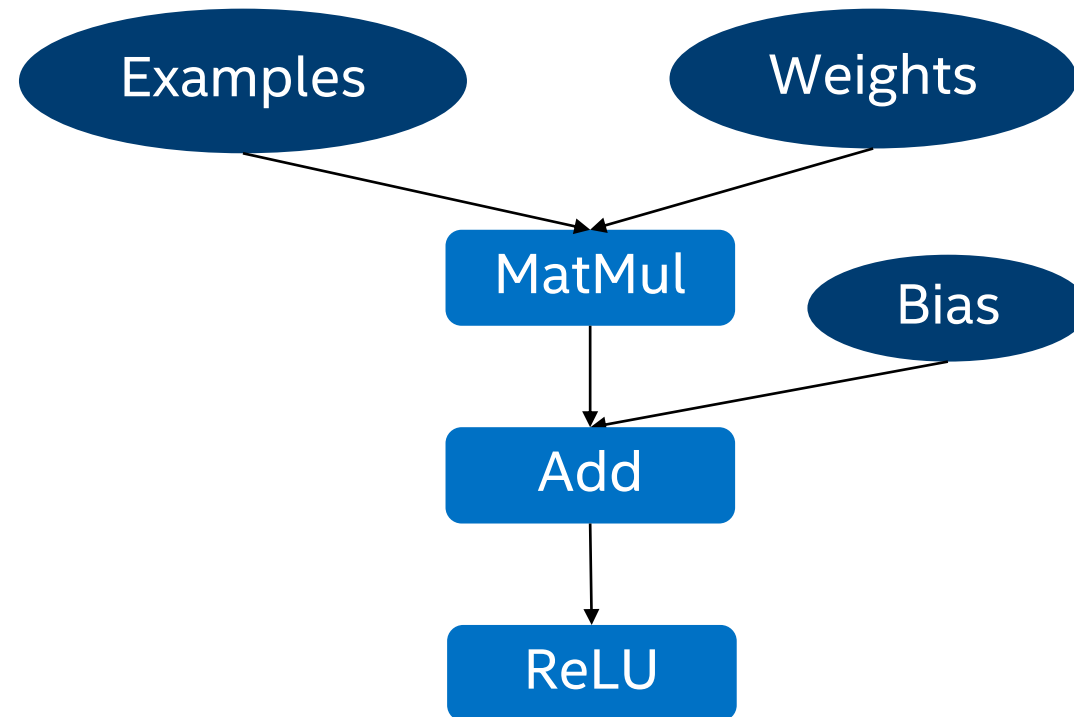
Intel® TensorFlow* optimizations

1. Operator optimizations: Replace default (Eigen) kernels by highly-optimized kernels (using Intel® oneDNN)
2. Graph optimizations: Fusion, Layout Propagation
3. System optimizations: Threading model

Run TensorFlow* benchmark

Operator optimizations

In TensorFlow, computation graph is a data-flow graph.

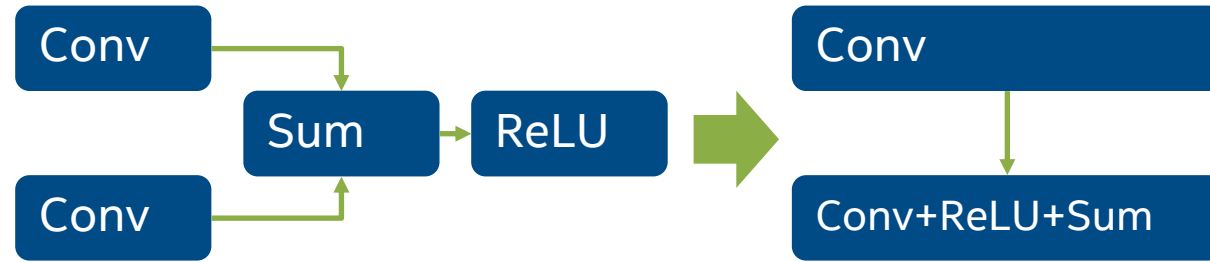


Operator optimizations

- Replace default (Eigen) kernels by highly-optimized kernels (using Intel® oneDNN)
- Intel® oneDNN has optimized a set of TensorFlow operations.
- Library is open-source (<https://github.com/oneapi-src/oneDNN>) and downloaded automatically when building TensorFlow.

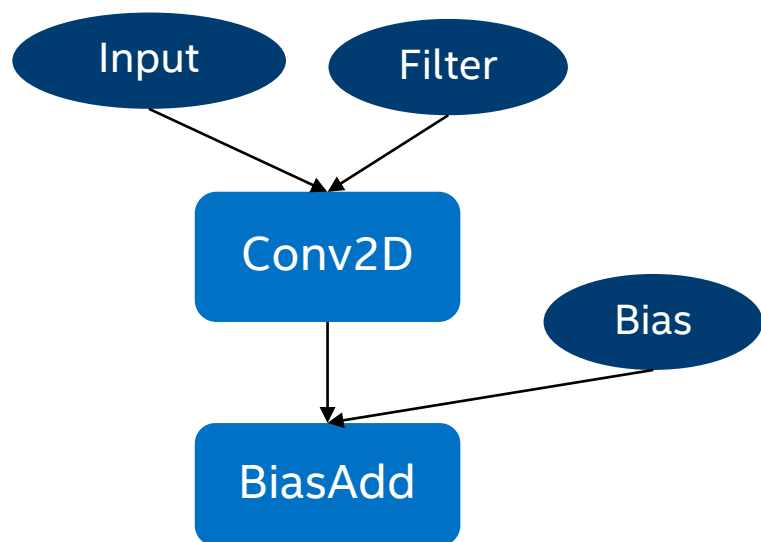
Forward	Backward
Conv2D	Conv2DGrad
Relu, TanH, ELU	ReLUGrad, TanHGrad, ELUGrad
MaxPooling	MaxPoolingGrad
AvgPooling	AvgPoolingGrad
BatchNorm	BatchNormGrad
LRN	LRNGrad
MatMul, Concat	

Fusing computations

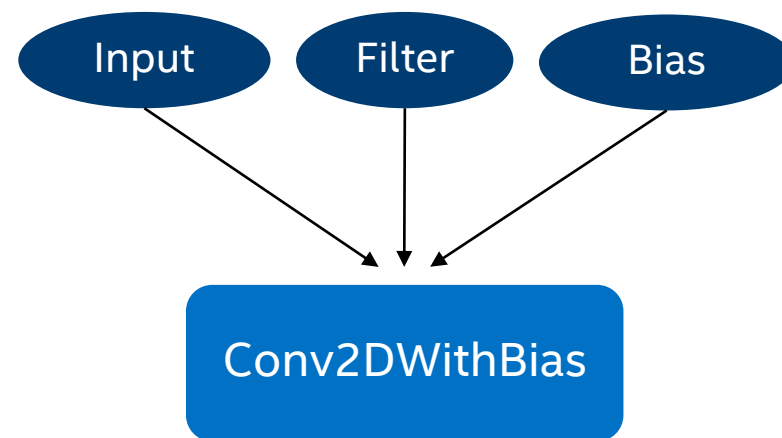


- On Intel processors a high percentation of time is typically spent in BW-limited ops
 - ~40% of ResNet-50, even higher for inference
- The solution is to fuse BW-limited ops with convolutions or one with another to reduce the # of memory accesses
 - Conv+ReLU+Sum, BatchNorm+ReLU, etc
- The frameworks are expected to be able to detect fusion opportunities
 - IntelCaffe already supports this

Graph optimizations: fusion

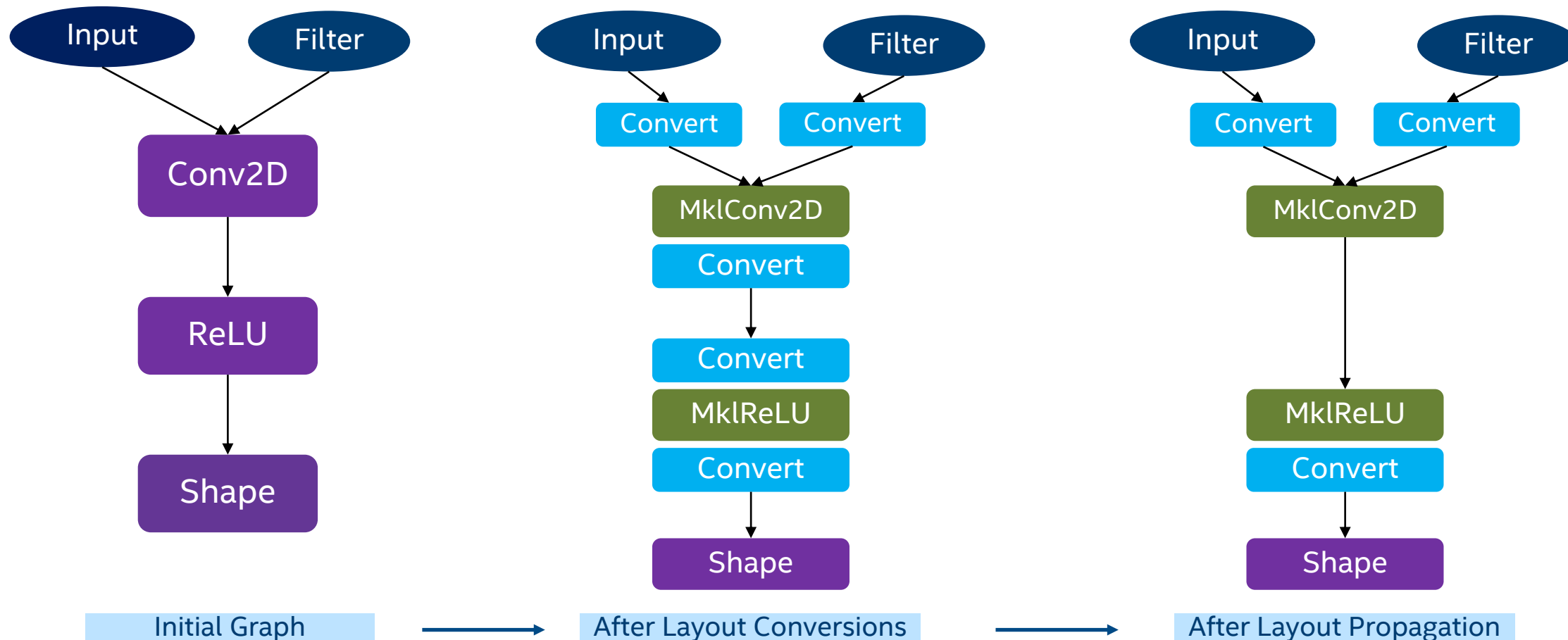


Before Merge



After Merge

Graph optimizations: layout propagation



All oneDNN operators use highly-optimized layouts for TensorFlow tensors.

Data Layout has a BIG Impact

- Continuous access to avoid gather/scatter
- Have iterations in inner most loop to ensure high vector utilization
- Maximize data reuse; e.g. weights in a convolution layer
- Overhead of layout conversion is sometimes negligible, compared with operating on unoptimized layout

21	18	32	6	3	
1	8	92	37	29	44
40	11	9	22	3	26
23	3	47	29	88	1
5	15	16	22	46	12
	29	9	13	11	1

21	18	...	1	..	8	92	..
----	----	-----	---	----	---	----	----

Channel based (NCHW)

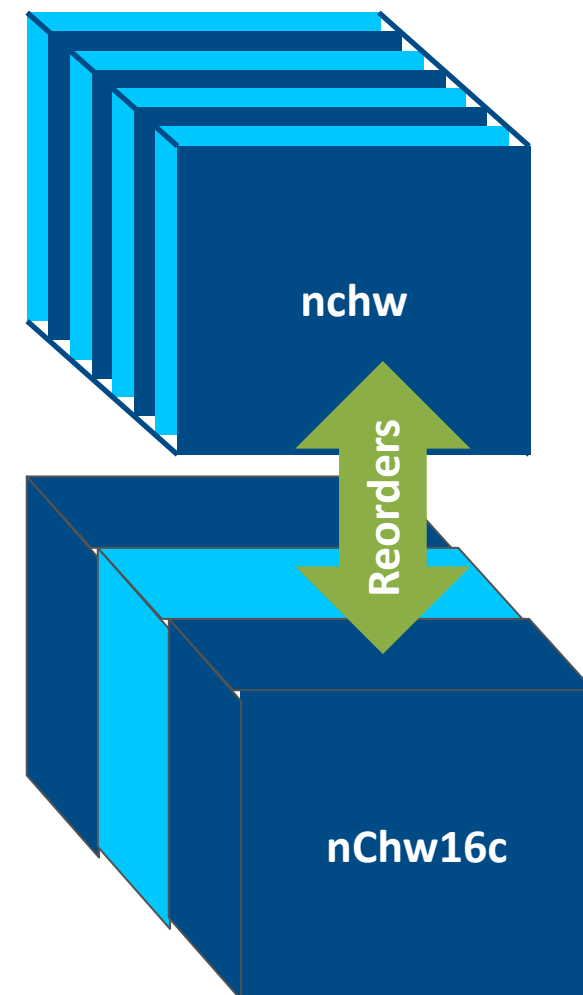
21	8	18	92	..	1	11	..
----	---	----	----	----	---	----	----

Pixel based (NHWC)

```
for i= 1 to N # batch size
  for j = 1 to C # number of channels, image RGB = 3 channels
    for k = 1 to H # height
      for l = 1 to W # width
        dot_product( ...)
```

More on memory channels: Memory layouts

- Most popular memory layouts for image recognition are **nhwc** and **nchw**
 - Challenging for Intel processors either for vectorization or for memory accesses (cache thrashing)
- Intel oneDNN convolutions use blocked layouts
 - Example: **nhwc** with channels blocked by 16 – **nChw16c**
 - Convolutions define which layouts are to be used by other primitives
 - Optimized frameworks track memory layouts and perform reorders **only** when necessary

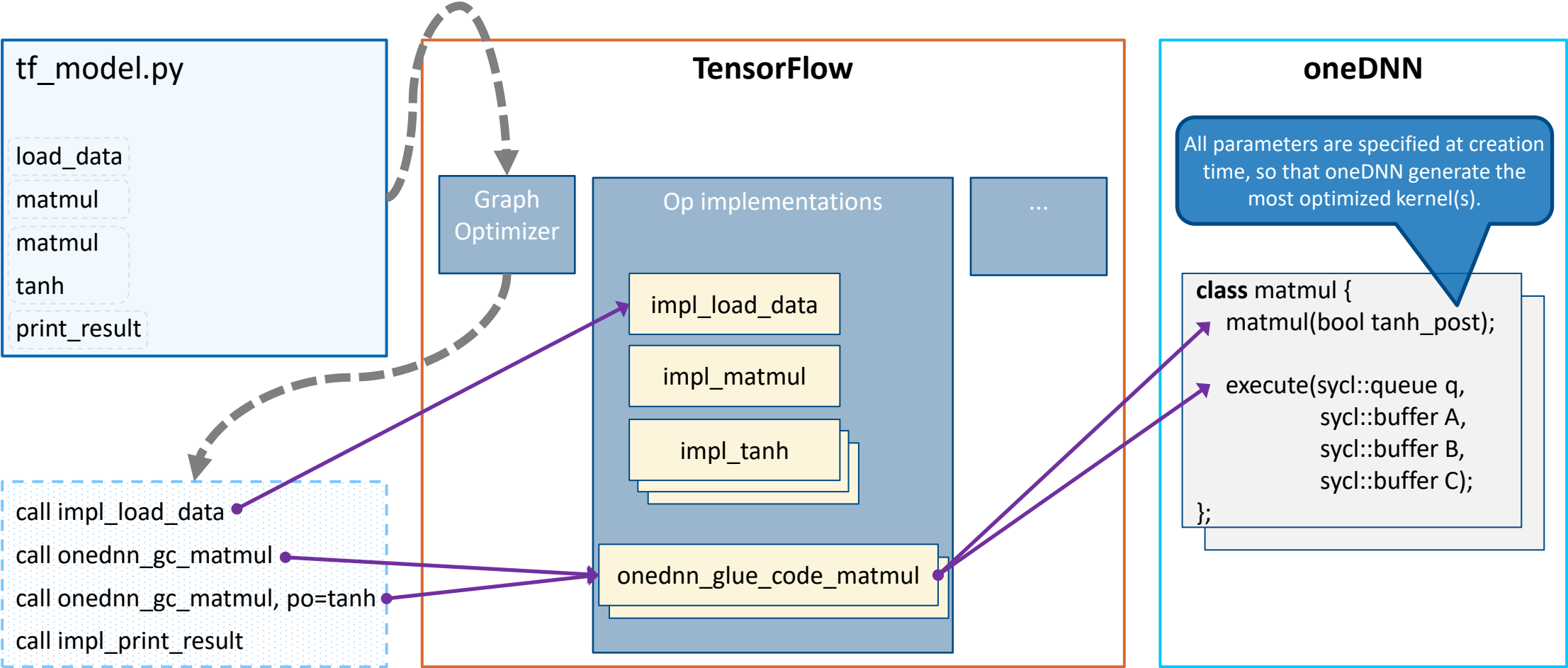


More details: https://oneapi-src.github.io/oneDNN/understanding_memory_formats.html

System optimizations: load balancing

- TensorFlow graphs offer opportunities for parallel execution.
- Threading model
 1. **inter_op_parallelism_threads** = max number of operators that can be executed in parallel
 2. **intra_op_parallelism_threads** = max number of threads to use for executing an operator
 3. **OMP_NUM_THREADS** = oneDNN equivalent of **intra_op_parallelism_threads**

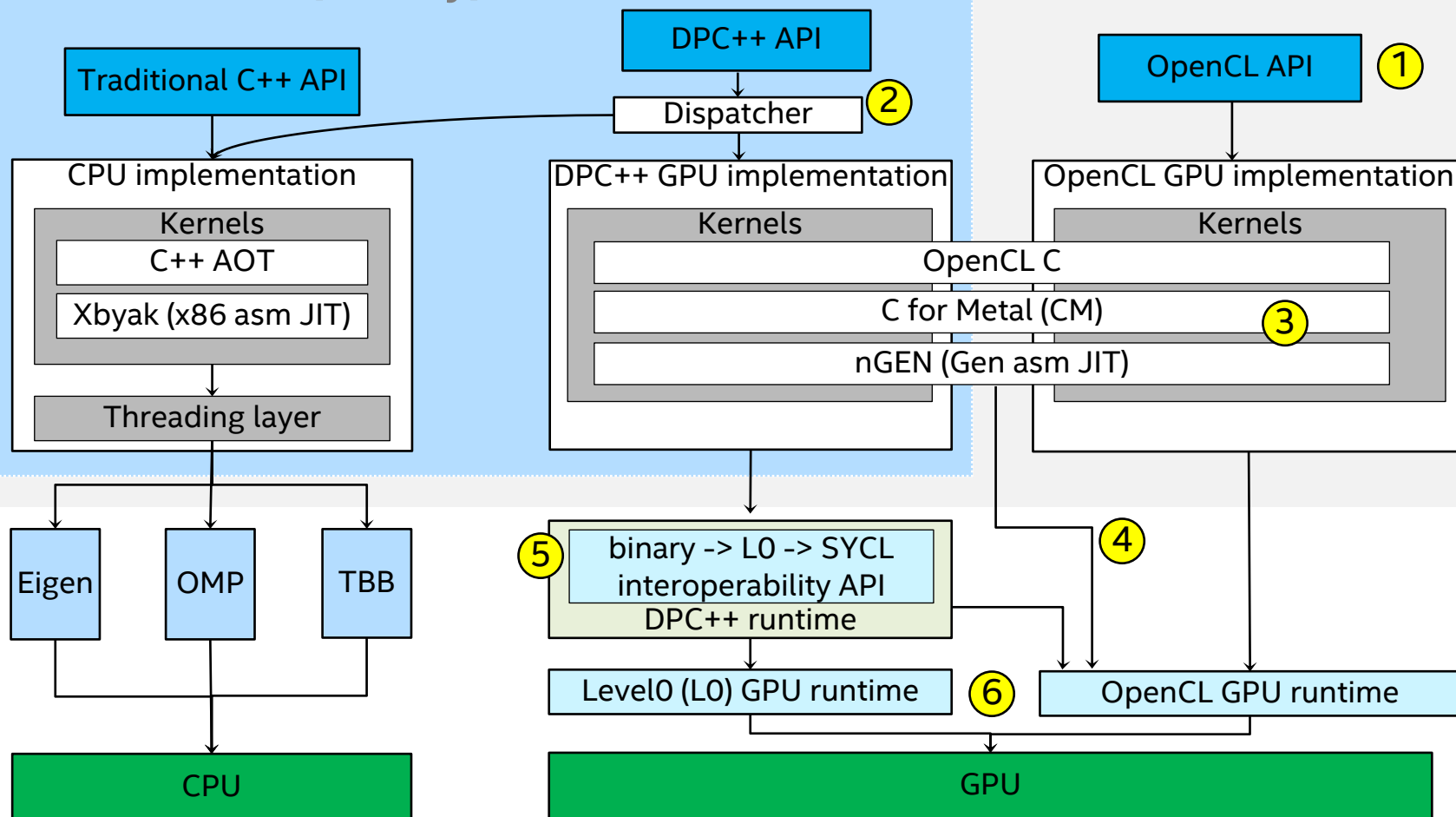
oneDNN <-> Frameworks interaction



oneDNN architecture overview

oneDNN (open source)

Intel oneDNN (binary)



- 1 OpenCL API is not available as part of Intel oneAPI binary distribution
- 2 Dispatching between CPU and GPU is based on the kind of device associated with the DPC++ queue
- 3 All GPU kernels are compiled in runtime. CM and nGEN support is not available publicly yet. Adding/migrating to DPC++ kernels is under consideration
- 4 OpenCL GPU RT is always needed to compile OpenCL C and CM kernels
- 5 In case of DPC++ and LO, binary kernels need to be wrapped to LO modules to create SYCL kernels eventually
- 6 Under DPC++ API/runtime, users can run on GPU via either OpenCL or LO GPU runtime: it should be specified in compile time, but can be checked during execution time

Develop Fast Neural Networks on Intel® CPUs & GPUs

with Performance-optimized Building Blocks

Intel® optimizations for PyTorch



PyTorch Optimization Strategy

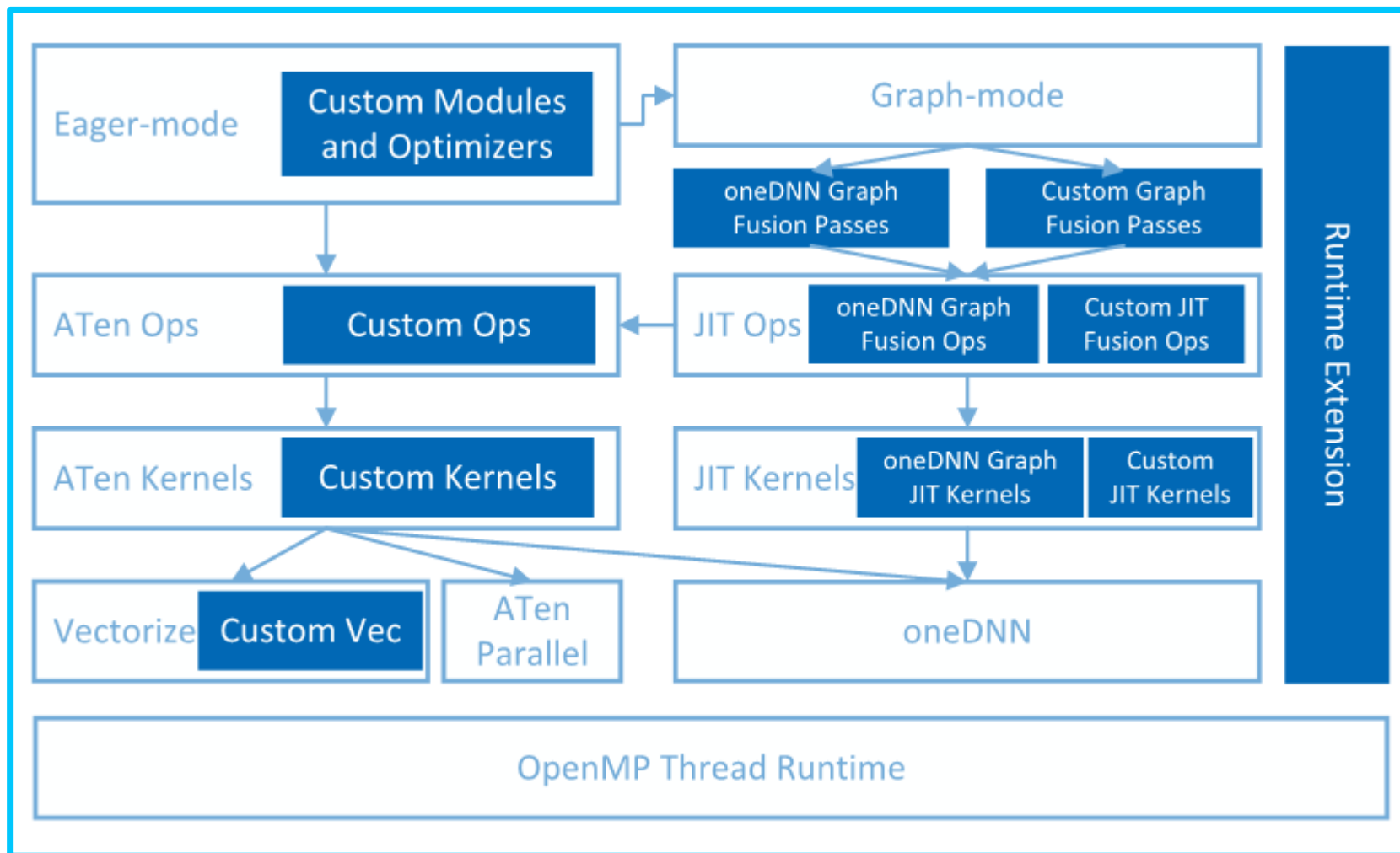


Intel[®] Extension
for PyTorch
(IPEX)

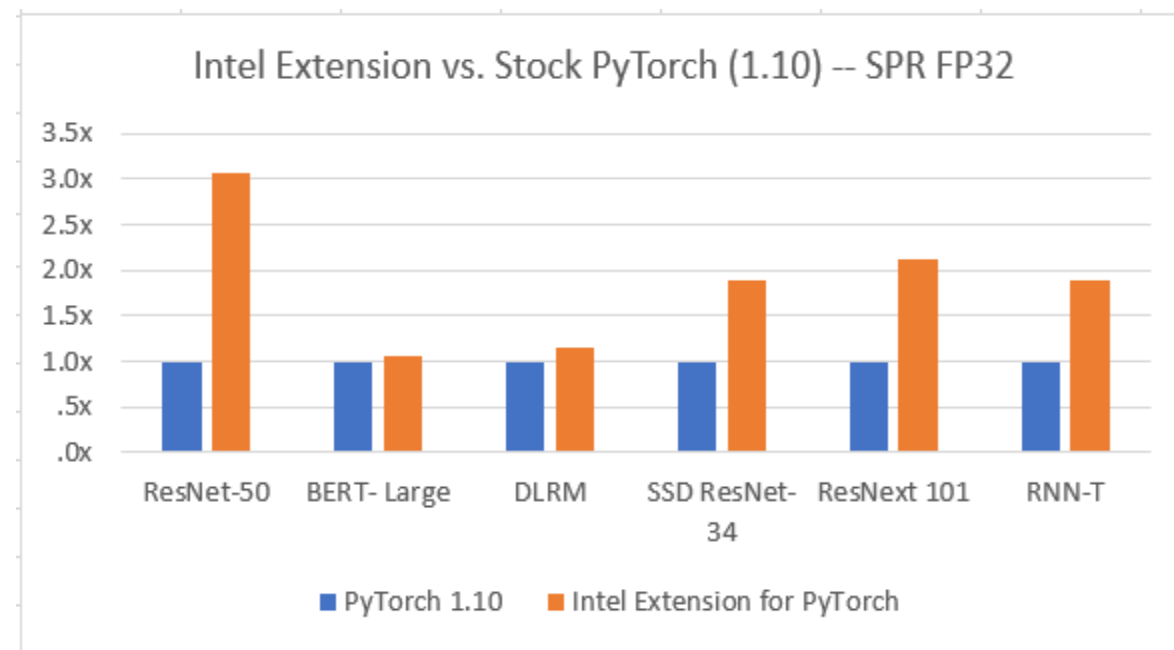
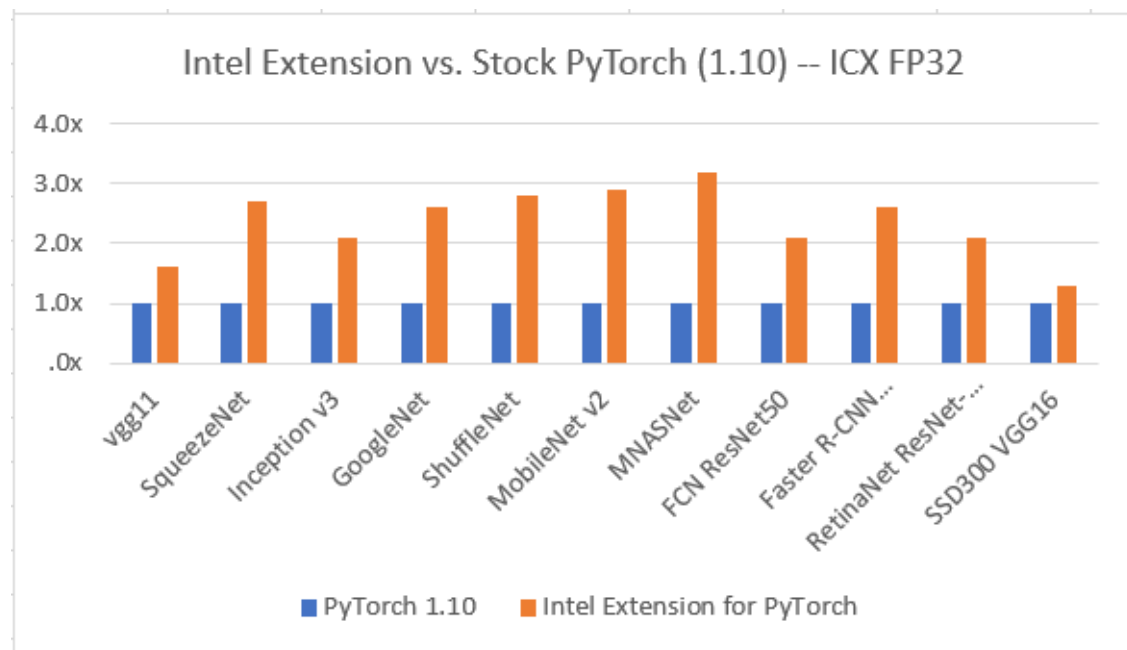
- General IA feature enabling and performance optimizations in stock PyTorch.
- Early access/adoption of aggressive optimizations through IPEX.
- Minimal line of code changes to get benefit from IPEX.

PyTorch-IPEX Demo

IPEX Architecture



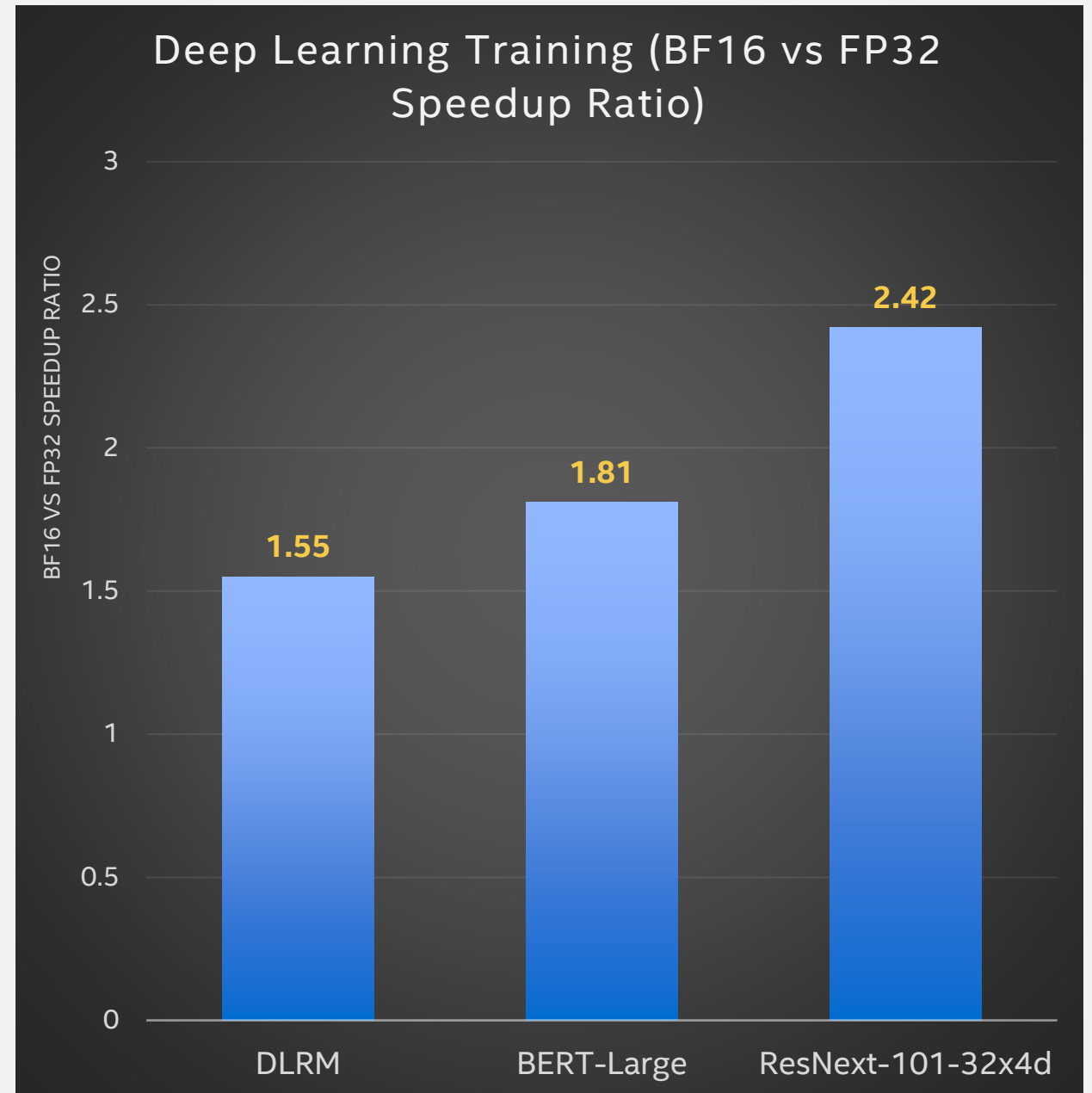
IPEX Performance boosts (FP32)





Training BF16 speedup vs FP32 with IPEX

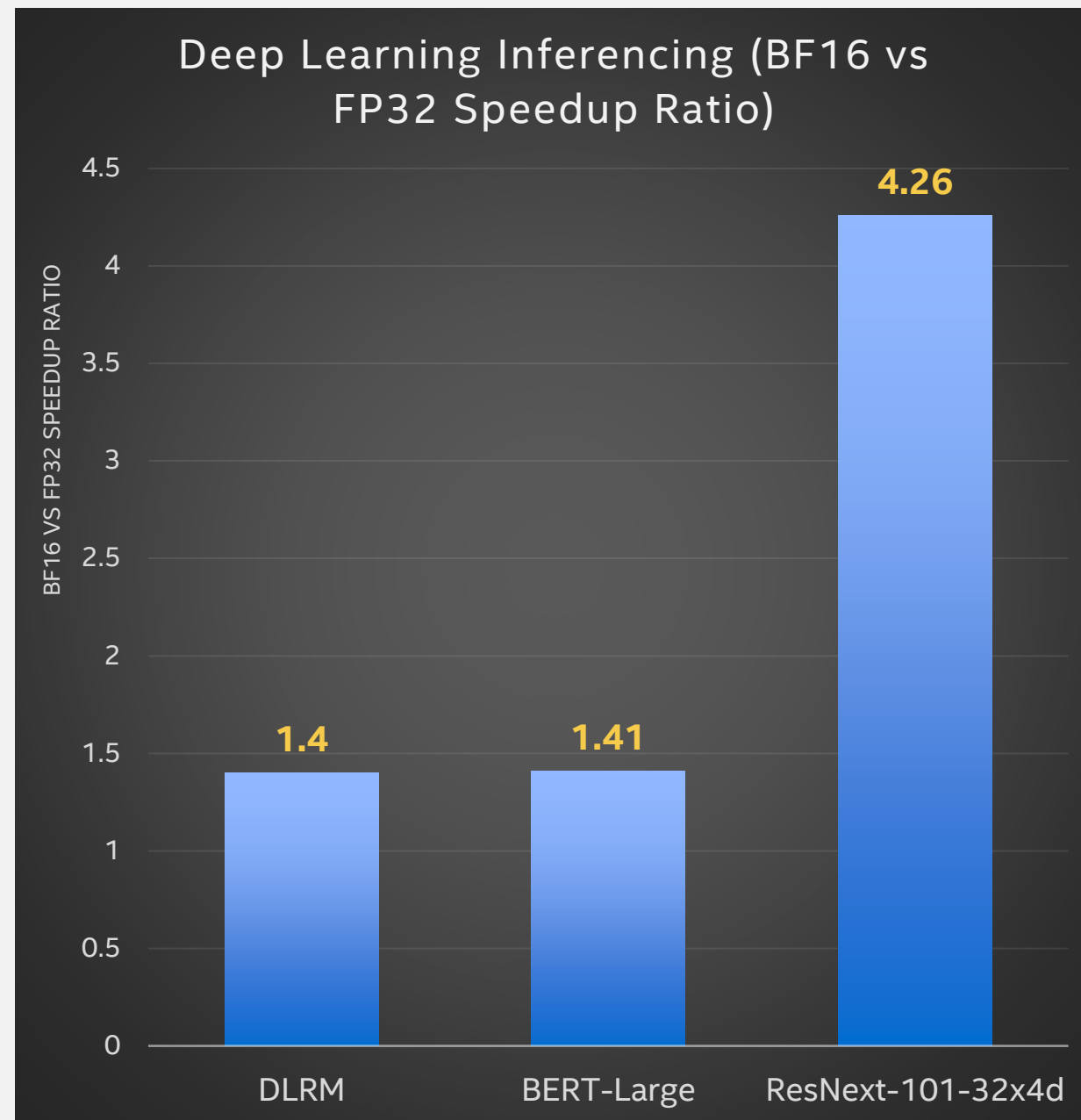
Single instance BF16 training performance gains over baseline (FP32 with Intel® Math Kernel Library for DLRM and BERT-Large, FP32 with Intel® oneDNN for ResNext-101-32x4d), measured on single-socket Intel(R) Xeon(R) Platinum 8380H Processor with 28 cores. The DLRM model uses 2K mini-batch-size with Criteo terabyte dataset, and the hyper-parameters use the MLPerf configuration. The BERT-large model uses 24 mini-batch-size with WikiText dataset. The ResNext-101-32x4d uses 128 mini-batch-size with ILSVRC2012 dataset.





Inferencing BF16 speedup vs FP32 with IPEX

Multi-instance BF16 inference performance gains over baseline (FP32 with Intel® Math Kernel Library for DLRM and BERT-Large, FP32 with Intel® oneDNN for ResNext-101-32x4d), measured on 8-socket Intel(R) Xeon(R) Platinum 8380H Processor with 28 cores per socket. The DLRM model uses 64 mini-batch-size per instance with Criteo terabyte dataset, and the hyper-parameters use the MLPerf configuration. The BERT-large model uses 1 mini-batch-size with WikiTest dataset. The ResNext-101-32x4d uses 1 mini-batch-size with ILSVRC2012 dataset.



Distributed deep learning with Intel[®] oneCCL



Intel® oneAPI Collective Communications Library

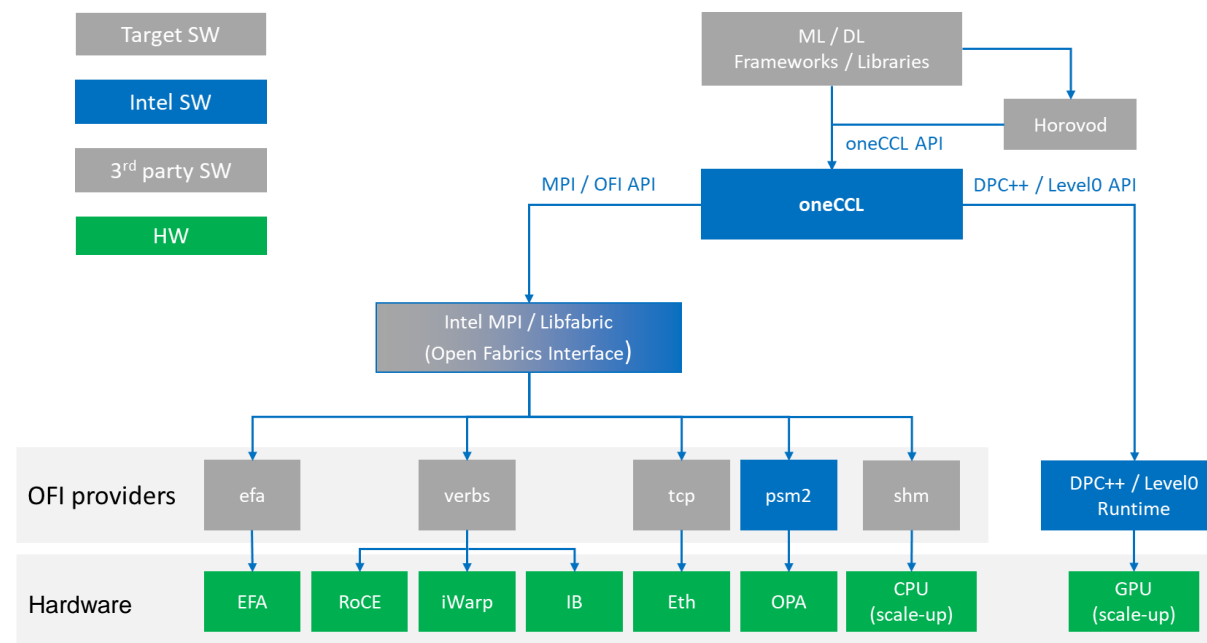
Optimize Communication Patterns

oneCCL provides optimized communication patterns for high performance on Intel CPUs & GPUs to distribute model training across multiple nodes

Transparently supports many interconnects, such as Intel® Omni-Path Architecture, InfiniBand, & Ethernet

Built on top of lower-level communication middleware-MPI & libfabrics

Enables efficient implementations of collectives used for deep learning training-all-gather, all-reduce, and reduce-scatter



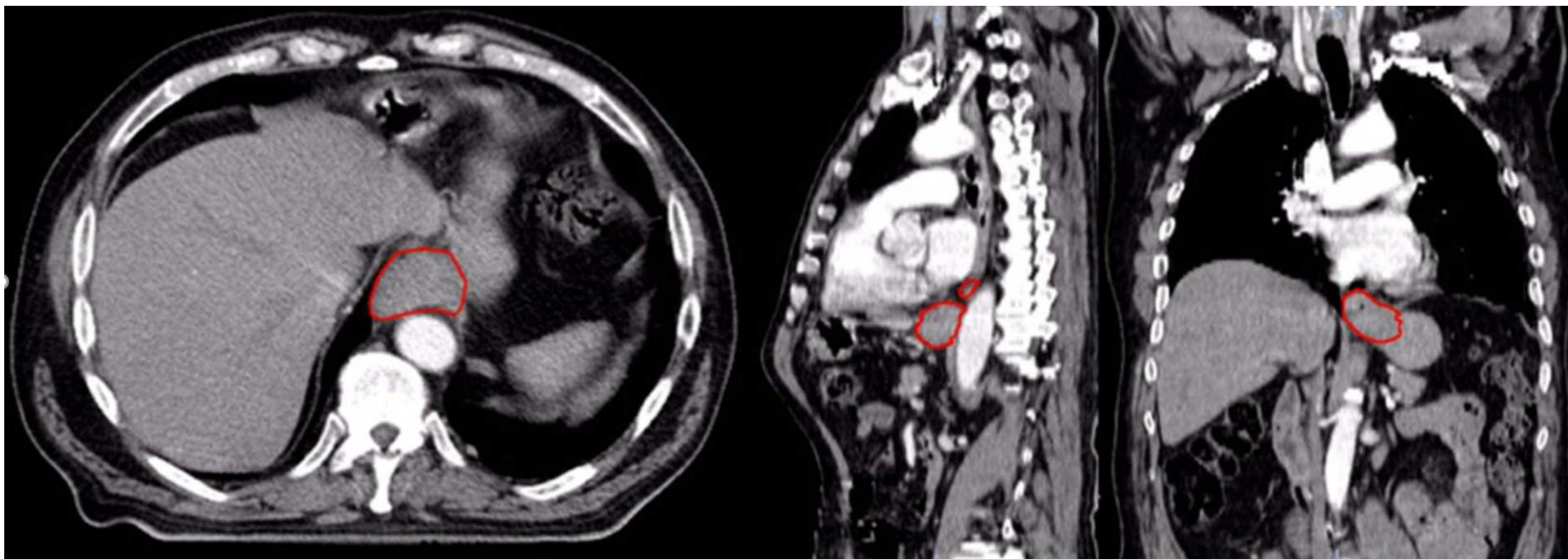
oneCCL is included in the
[Intel® oneAPI Base Toolkit](#)

[oneCCL Product Page](#)



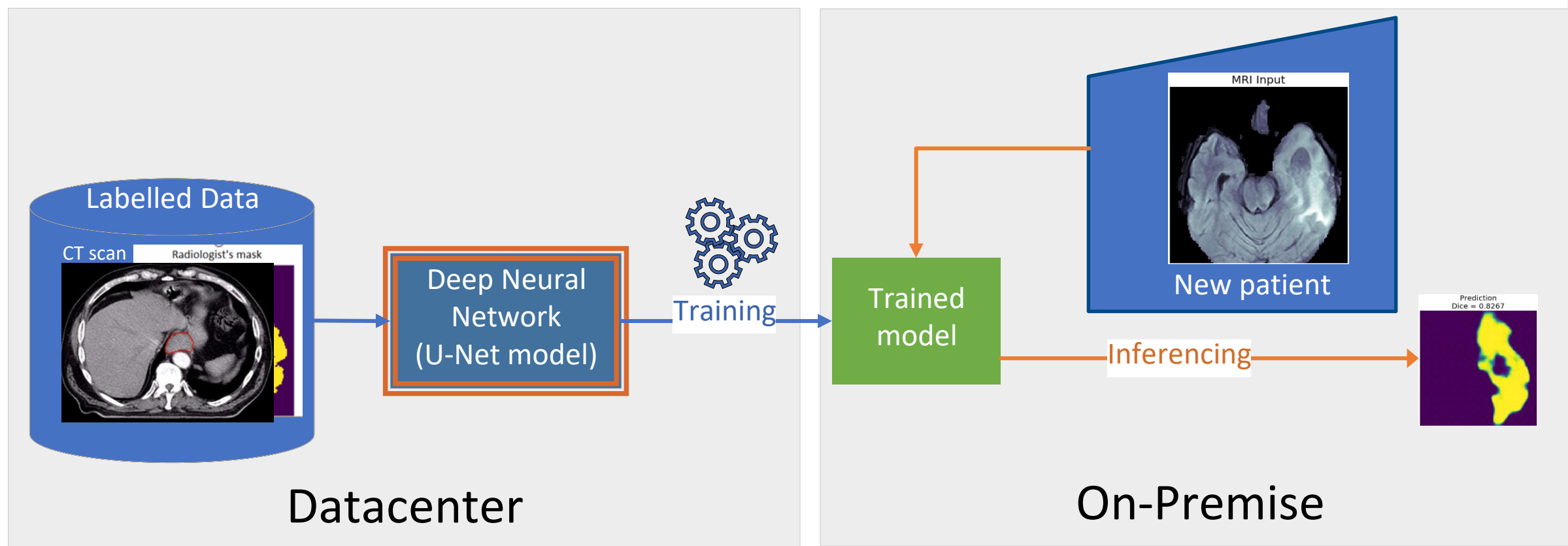
Distributed AI on a real-world use-case

Problem Statement: AI Algorithm to **segment*** the Metabolic Tumour Volume (MTV) in oesophageal cancer



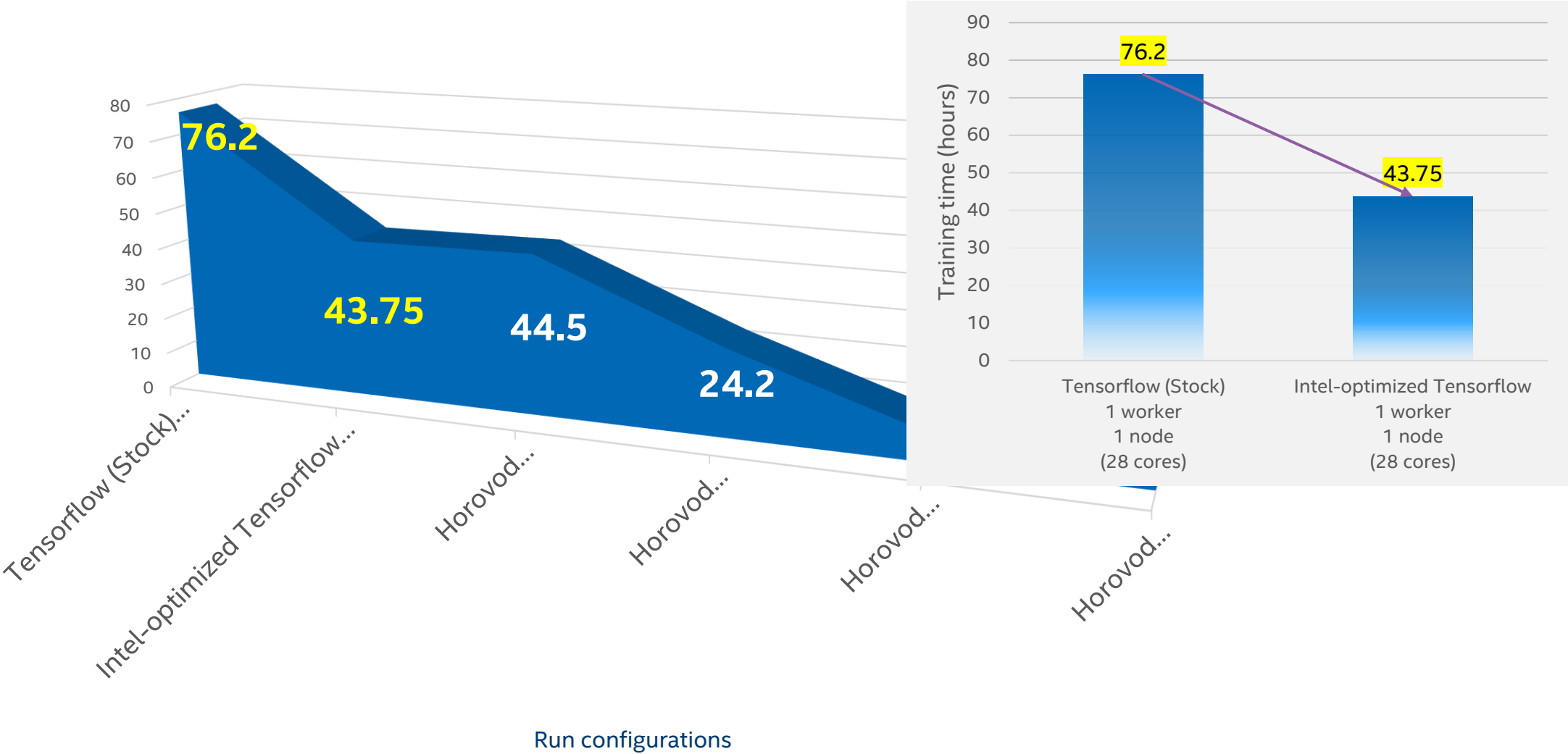
* **Segmentation:** Find the contour of the tumor on the CT scan

AI solution to for tumor segmentation:



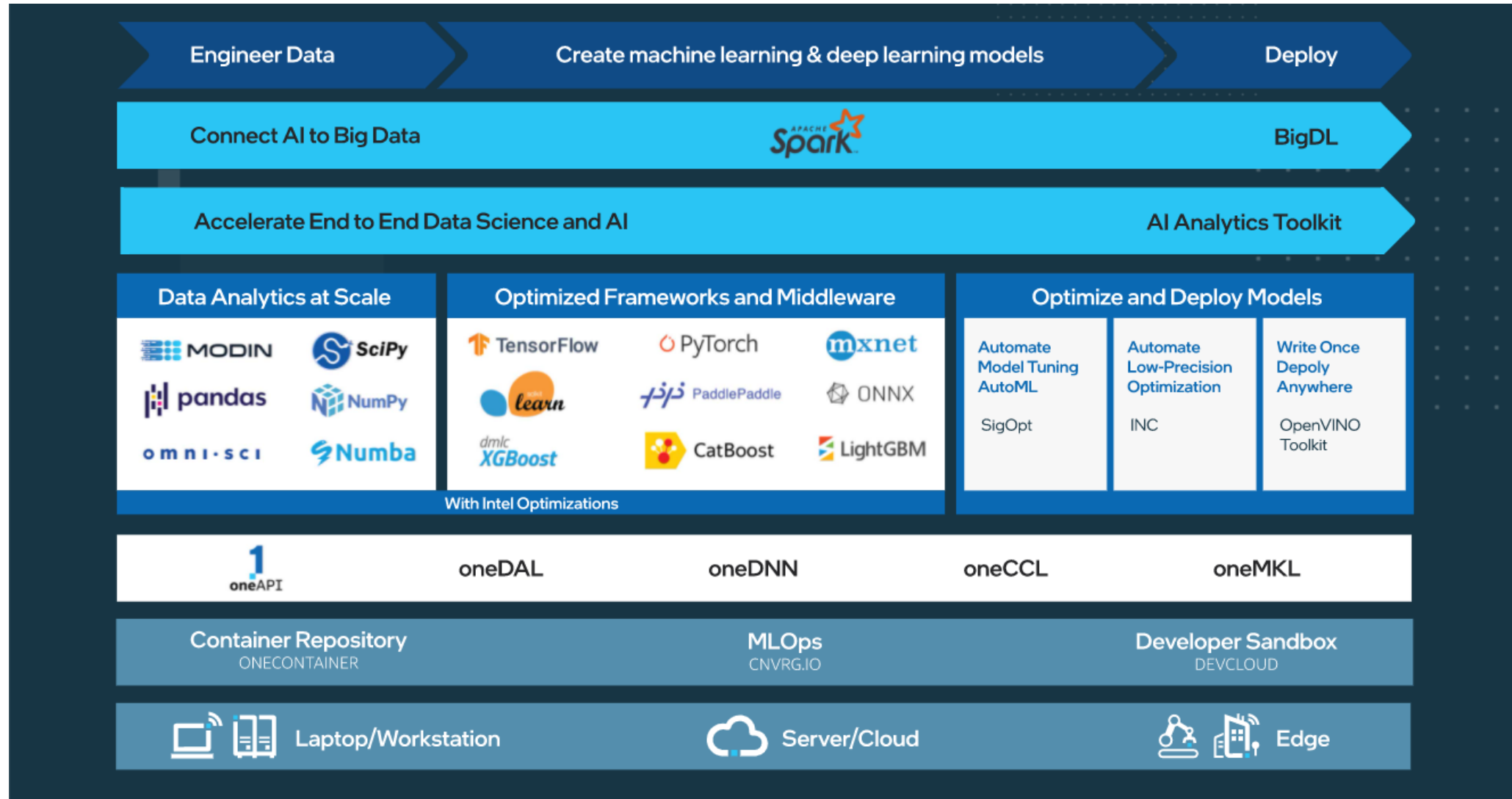
Performance results

Training time (hours) - lower is better



AI Software Stack for Intel® XPU

Intel offers a Robust Software Stack to Maximize Performance of Diverse Workloads





Thank you

