张建宇

AI软件解决方案工程师

英特尔

使用 Intel® 优化的 AI 框架 释放Intel® XPU 的动能，加速 AI

- 随着AI应用对算力需求的增强，CPU，GPU和专用加速芯片都可以通过异构运算的方式，协同工作，增强系统算力。我们统称它们为XPU;
- Intel不断的在新推出的CPU和GPU里，加入增强AI运算的新特性，比如Intel® Deep Learning Boost， Intel® Advanced Matrix Extensions。这些新特性可以极大的加速AI的训练，推理的性能;
- 为了方便用户使用到这些硬件特性，Intel同时也推出了Intel® AI Analytics Toolkit软件工具包，包含了针对硬件新特性优化的AI框架和中间件，供用户免费使用。

专注于Intel平台（CPU，GPU）上，AI解决方案和性能优化。硕士毕业于西北工业大学模式识别和人工智能专业。具有丰富的AI，虚拟化，通讯行业和嵌入式软件开发经验。

使用Intel优化的AI框架，释放Intel® XPU的动能，加速AI

Jianyu Zhang
Intel AI Software Solution Engineer, AIA, SATG

intel®

Agenda

- Intel® XPU Features to Accelerate AI
  - Xeon
  - Xᵉ GPU
- Intel® oneAPI® Power AI
- Intel® Optimization for Frameworks
  - Intel® Extension for PyTorch*
  - Intel® Extension for Tensorflow*
  - Intel® Neural Compressor
  - Intel® Extension for Scikit-Learn*
  - XGBoost
  - oneDAL

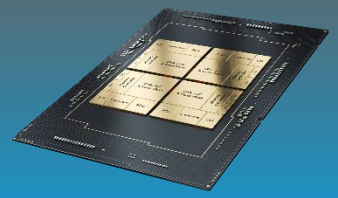
Intel® XPU Features to Accelerate AI

XPU

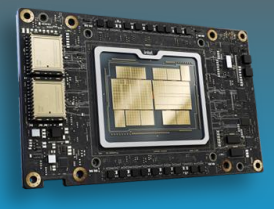
Diverse accelerators needed to meet today's performance requirements:

48% of developers target heterogeneous systems that use more than one kind of processor or core[1]

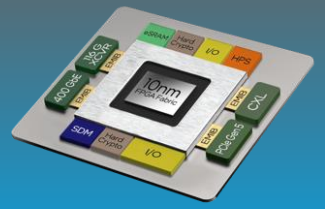
CPU



GPU



FPGA



Other Accelerators

Developer Challenges: Multiple Architectures, Vendors, and Programming Models

oneAPI

Open, Standards-based, Multiarchitecture Programming

1. *Evans Data Global Development Survey Report 22.1, June 2022*

The 4th Xeon - Sapphire Rapids

Sapphire Rapids

PCI Express Gen 5.0, Compute Express Link 1.1, DDR5

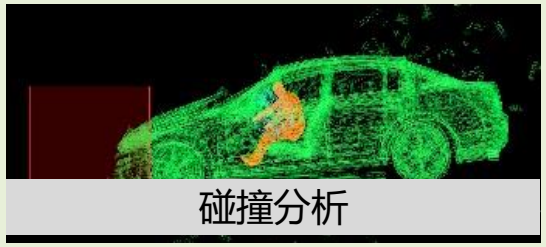
高级矩阵计算扩展 (AMX)，加速 AI

发布

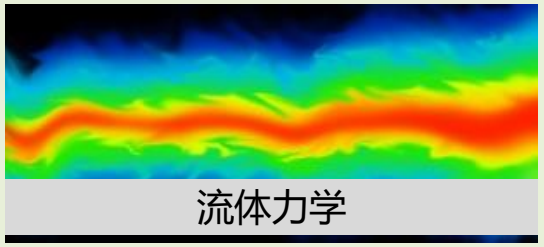
Sapphire Rapids (Intel Xeon Max CPU) 内置高带宽内存HBM型号

加速内存带宽需求型HPC负载

模式支持: HBM only, Flat, Cache



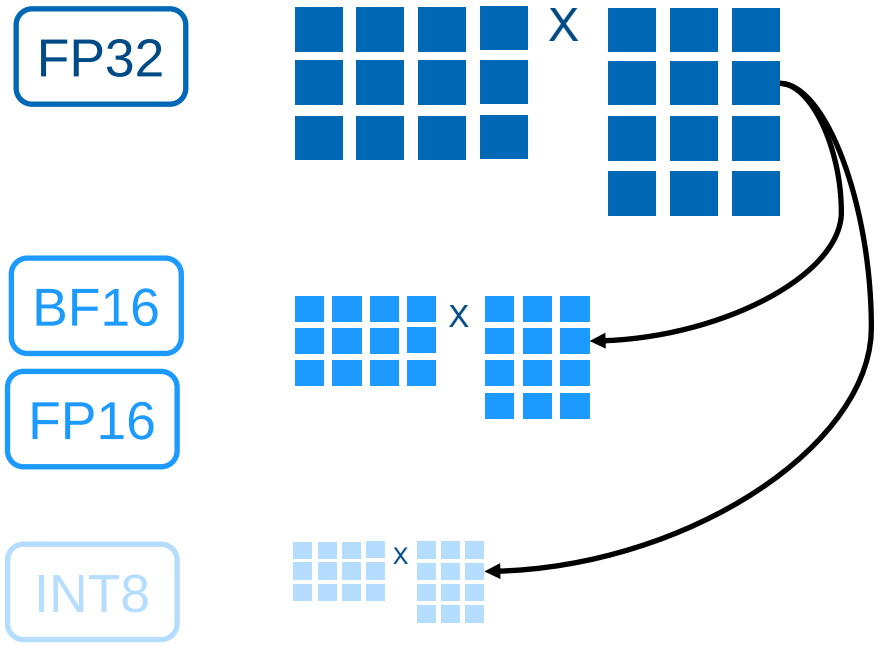
碰撞分析

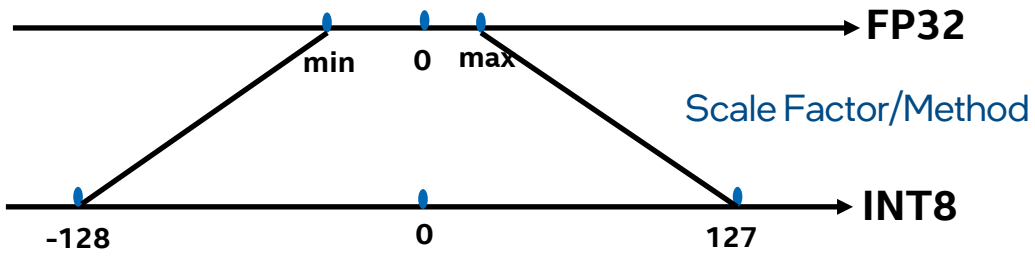


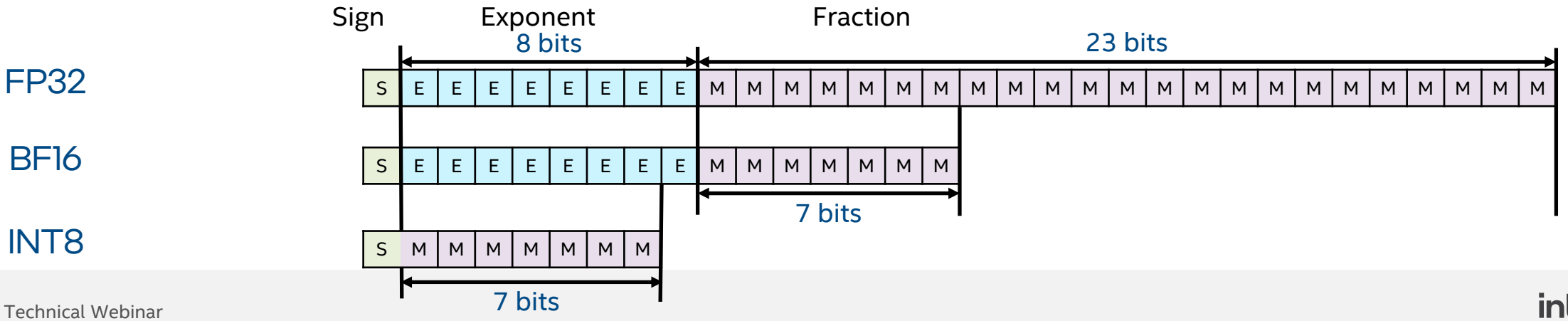
流体力学

深度学习训练

Low Precision Inference & Quantization







Hardware Instruction Accelerate AI

Matrix

Matrix Data Type	Intel Intrinsics
BF16	AMX_BF16
INT8	AMX_INT8

AVX512_VNNI = 3 * AVX512

AMX_INT8 = 8 *AVX512_VNNI

Vector

Vector Data Type	Intel Intrinsics
FP32	AVX512
BF16	AVX512_BF16
INT8	AVX512_VNNI

Accelerate AI in Xeon

Generation Intel® Xeon® Scalable Processors	Code Name	AVX512	AVX512_VNNI	AVX512_BF16	AMX_INT8	AMX_BF16
2nd	Cascade Lake	✓	✓			
3rd	Cooper Lake	✓		✓		
3rd	Ice Lake	✓	✓			
4th	Sapphire Rapids	✓	✓	✓	✓	✓
		FP32	INT8	BF16	INT8	BF16

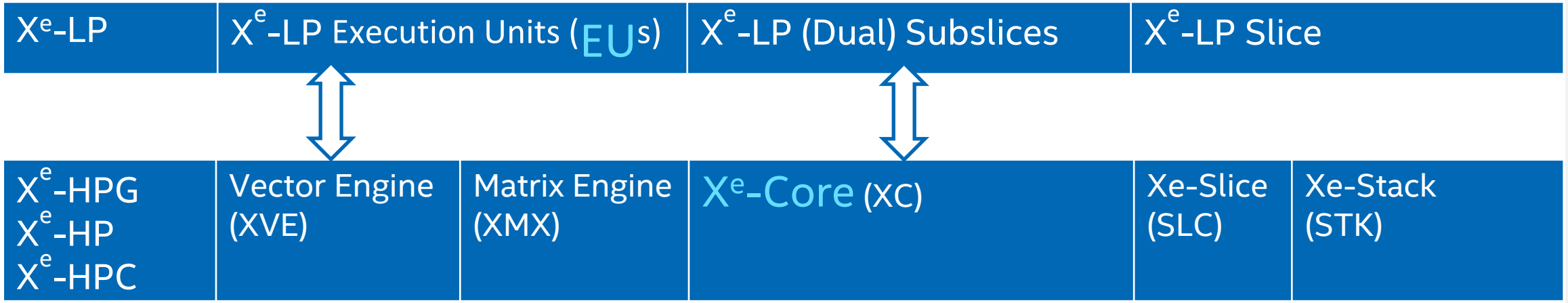
Vector

Matrix

Intel® Xᵉ GPU Architecture

Name	Range	Product
X^e-LP	integrated/low power	Gen12 in 11th Core
X^e-HPG	enthusiast/high performance gaming	Arc 380, Arc 750, Arc 770
X^e-HP	data center/AI	Data Center Flex Series (ATS-M)
X^e-HPC	high performance computing	Data Center Max Series (PVC)

Compute Unit



Media Supercomputer

Intel® Data Center GPU Flex Series

Sampling Today | Shipping Mid 2022

75-150W | Xe-HPG | GDDR6 memory

Video Transcode

8 Simultaneous 4K Streams

30+ Simultaneous 1080p Streams

Virtual Desktop Infrastructure

60+ Virtualized Functions

Cloud Gaming

30+ Game Streams[1]

Inference

150 TOPS[2]

[1]720p30 Android Games | [2] INT 8

Industry's Only	Open-Source Media Solution Stack
Industry's First	GPU with HW AV1 Encode

Xe-core

Compute Building Block of Xe HPG-based GPUs

16
Vector Engines

256 bit
per engine

16
Matrix Engines

1024 bit
per engine

Intel® oneAPI Power AI

oneAPI Industry Initiative

Break the Chains of Proprietary Lock-in

Freedom to Make Your Best Choice

- One programming model for multiple architectures and vendors
- Cross-architecture code reuse for freedom from vendor lock-in

Realize all the Hardware Value

- Performance across CPU, GPUs, FPGAs, and other accelerators
- Expose and exploit cutting-edge features of the latest hardware

Develop & Deploy Software with Peace of Mind

- Open industry standards provide a safe, clear path to the future
- Compatible with existing languages and programming models including C, C++ with SYCL, Python, OpenMP, Fortran, and MPI
- Powerful libraries for acceleration of domain-specific functions

oneAPI

The productive, smart path to freedom for accelerated computing from the economic and technical burdens of proprietary programming models

Intel® oneAPI Toolkits

oneAPI

Intel® oneAPI Base Toolkit

A core set of high-performance libraries and tools for building C++, SYCL, C/OpenMP, and Python applications

Add-on Domain-specific Toolkits

For HPC developers

Intel® oneAPI Tools for HPC
Deliver fast Fortran, OpenMP & MPI applications that scale

For Edge & IoT developers

Intel® oneAPI Tools for IoT
Build efficient, reliable solutions that run at network's edge

For visual creators, scientists, and engineers

Intel® oneAPI Rendering Toolkit
Create performant, high-fidelity visualization applications

Toolkits powered by oneAPI

For AI developers and data scientists

Intel® AI Analytics Toolkit
Accelerate machine learning & data science pipelines end-to-end with optimized DL & ML frameworks & high-performing Python libraries

OpenVINO™

For deep learning inference developers

Intel® OpenVINO™ toolkit
Deploy high performance inference & applications from edge to cloud

Download at intel.com/oneAPI
Or visit Intel® Developer Cloud

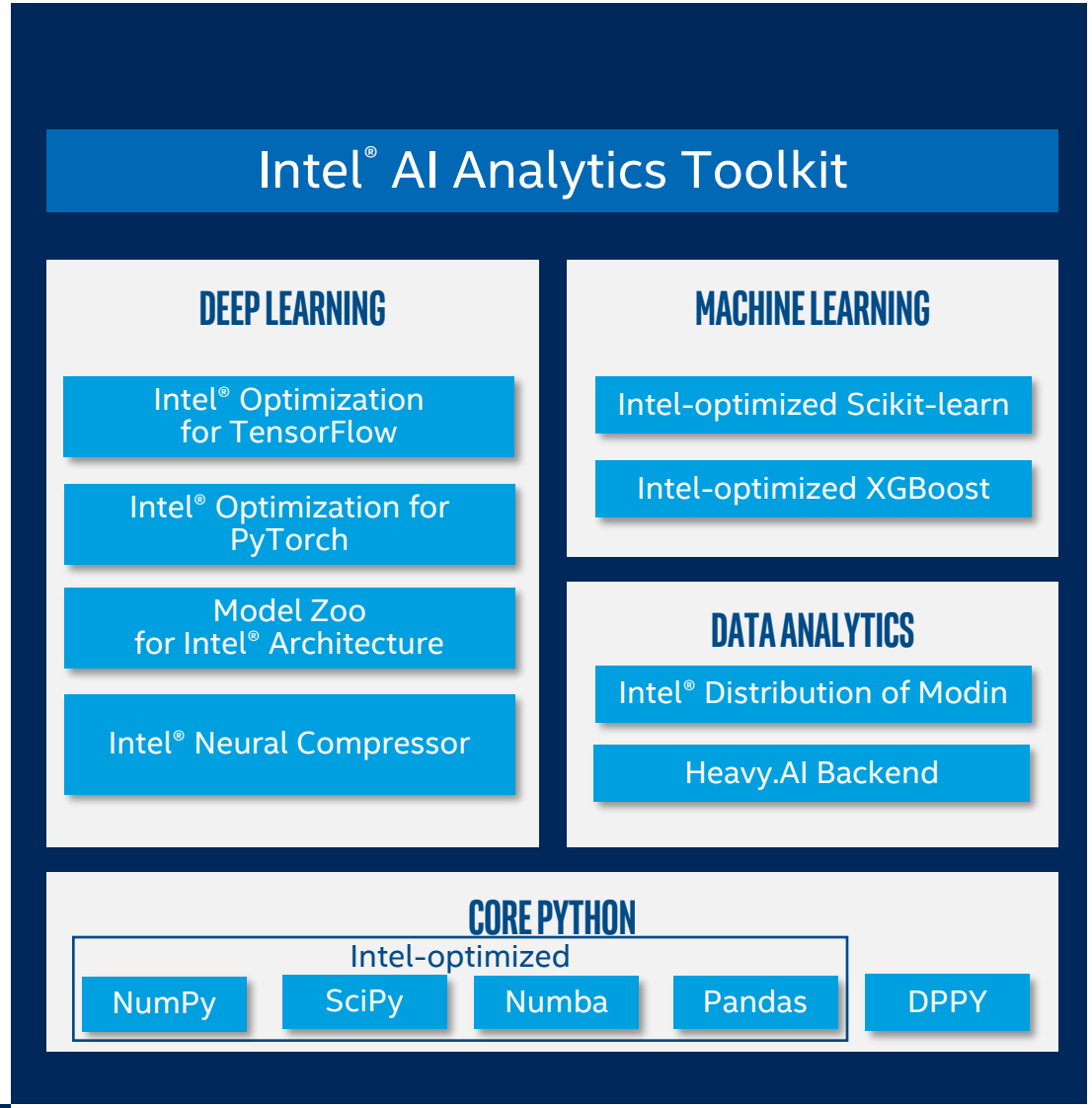
Intel® AI Analytics Toolkit

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: Here



Deep Learning Training & Inference Performance *

Uses Intel® Optimization for PyTorch with 3rd Gen Intel® Xeon® Scalable Processors

Training	# Cores per instance	# Instances	BF16 (samples/s)	FP32 (samples/s)	Speedup Ratio
DLRM	28	1	99321	71061	**1.40**
ResNet-50	28	4	399	243	**1.64**
ResNeXt-101 32x4d	28	4	193	120	**1.60**

Table 1. BF16 training performance gains over baseline (FP32 with Intel oneDNN)

Inference	# Cores per instance	# Instances	INT8 (samples/s)	FP32 (samples/s)	Speedup Ratio
DLRM	1	28	611082	214559	**2.85**

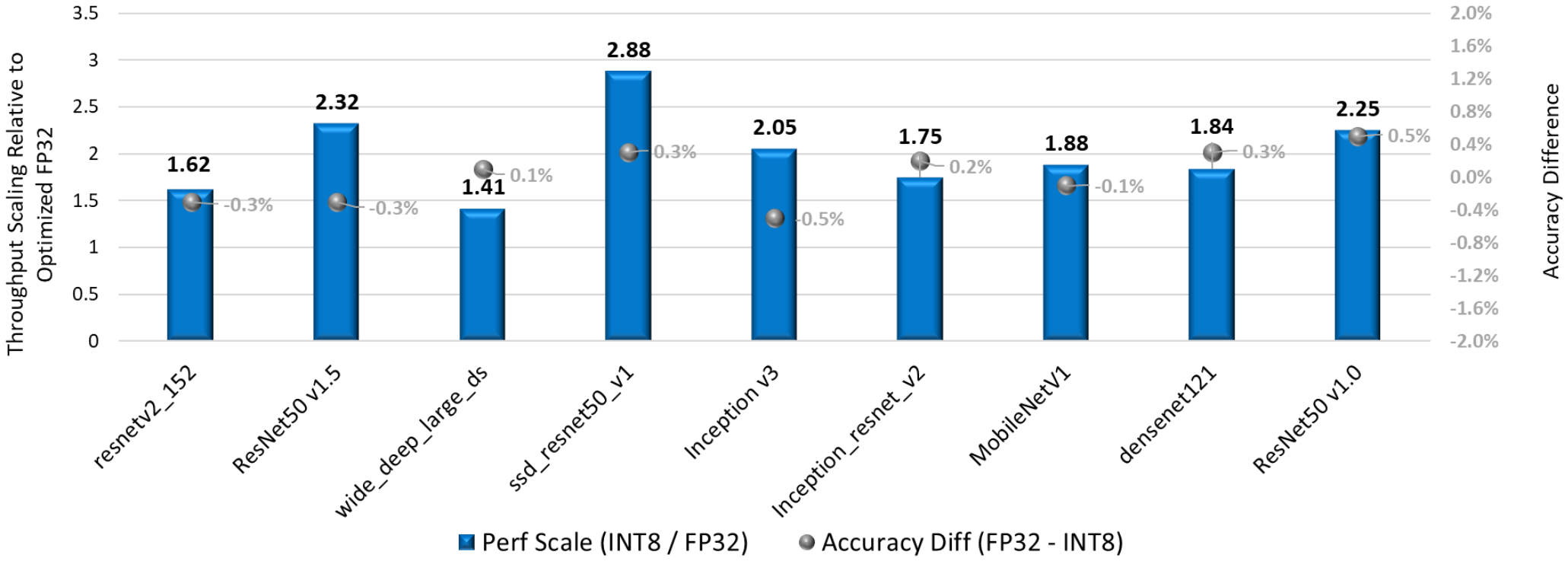
Table 2. INT8 inference performance gains over baseline (FP32 with Intel oneDNN)

*Measured March 2021

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

INT8 Quantized Inference Performance *

Uses Intel® Optimization for TensorFlow & Intel® Neural Compressor



INT8 Inference Throughput Scaling up to **3.8x** & Accuracy Drop within **0.6%**

*Measured March 2021

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

Intel® Optimization for Frameworks

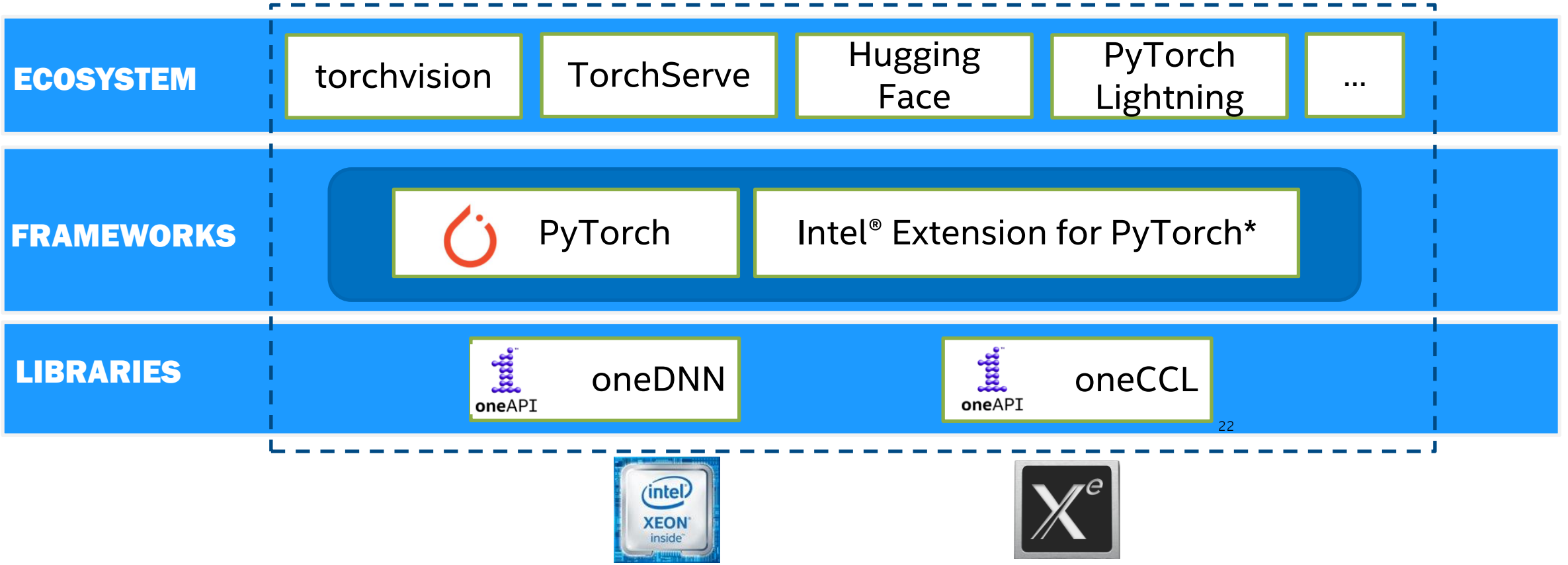
oneAPI Deep Neural Network Library (oneDNN)

- Supports FP32, FP16, Bfloat16, and int8.
- Leverages Intel® DL Boost, AVX512 instructions and processor capabilities
- Fused operations for optimized performance

- Compilers: Intel® oneAPI DPC++ / C++ Compilers
- OS: Linux, Windows, macOS
- CPU: Intel Atom, Intel® Core™, Intel® Xeon®, Intel® Xeon® Scalable processors
- GPU: Intel® Processor Graphics Gen9, Intel® Processor Graphics Gen 12, Intel® Data Center GPU Flex Series

Category	Functions
Compute intensive operations	• (De-)Convolution • Inner Product • RNN (Vanilla, LSTM, GRU) • GEMM
Memory bandwidth limited operations	• Pooling • Batch Normalization • Local Response Normalization • Layer Normalization • Elementwise • Binary elementwise • Softmax • Sum • Concat • Shuffle
Data manipulation	• Reorder

Intel Optimization for PyTorch

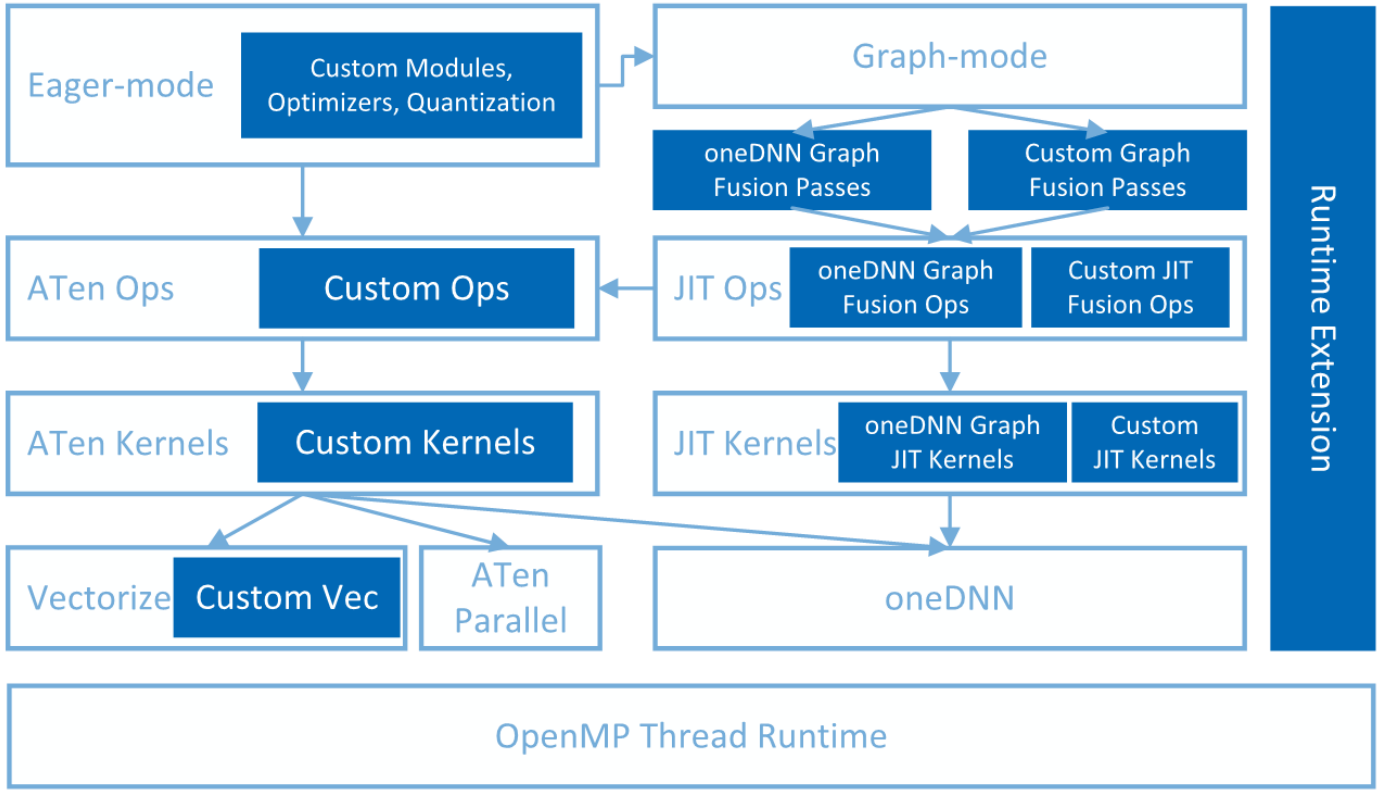


Major Optimization Methodologies

Operator Optimization

Overview

- Extends PyTorch with optimizations for extra performance boost on Intel hardware.
- Open sourced on Github.
- Optimized operators and kernels are registered through PyTorch dispatching mechanism.
- Loaded dynamically in Python script.
- Dynamically linked in CPP executables.

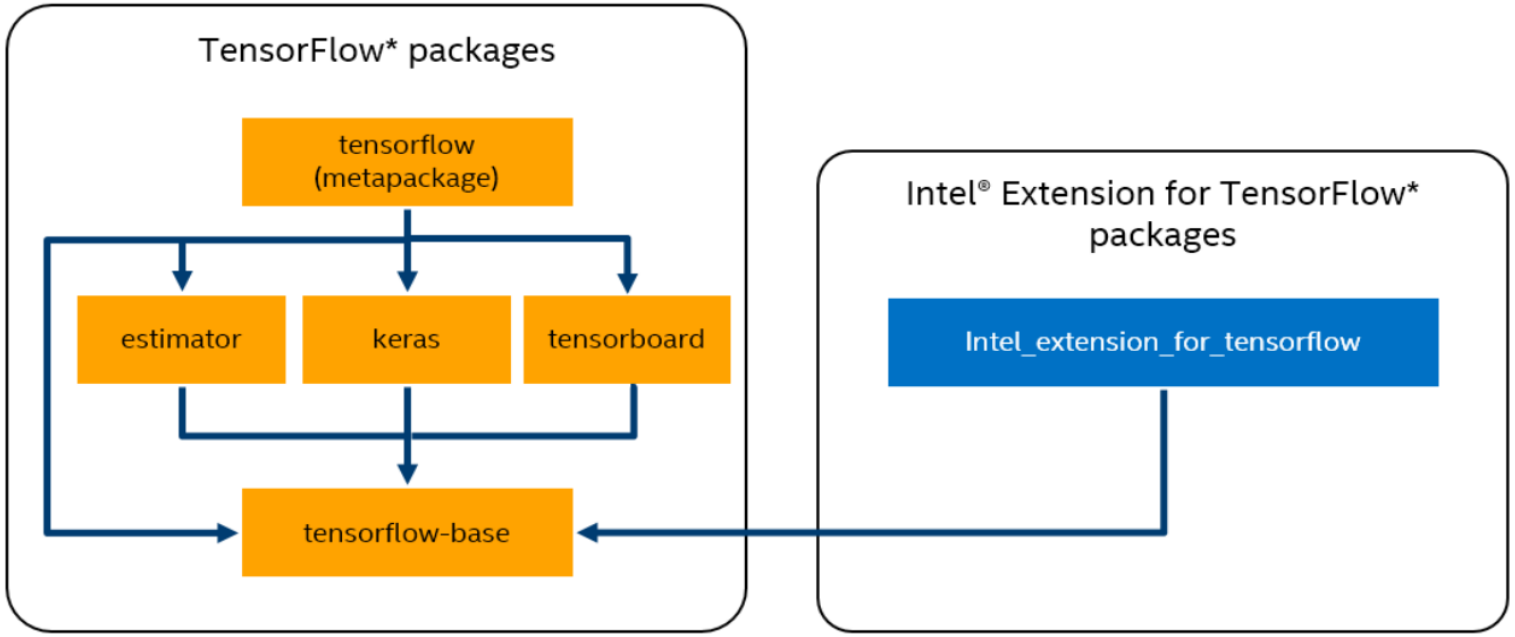


https://intel.github.io/intel-extension-for-pytorch/index.html

https://github.com/intel/intel-extension-for-pytorch

Intel® Extension for TensorFlow*

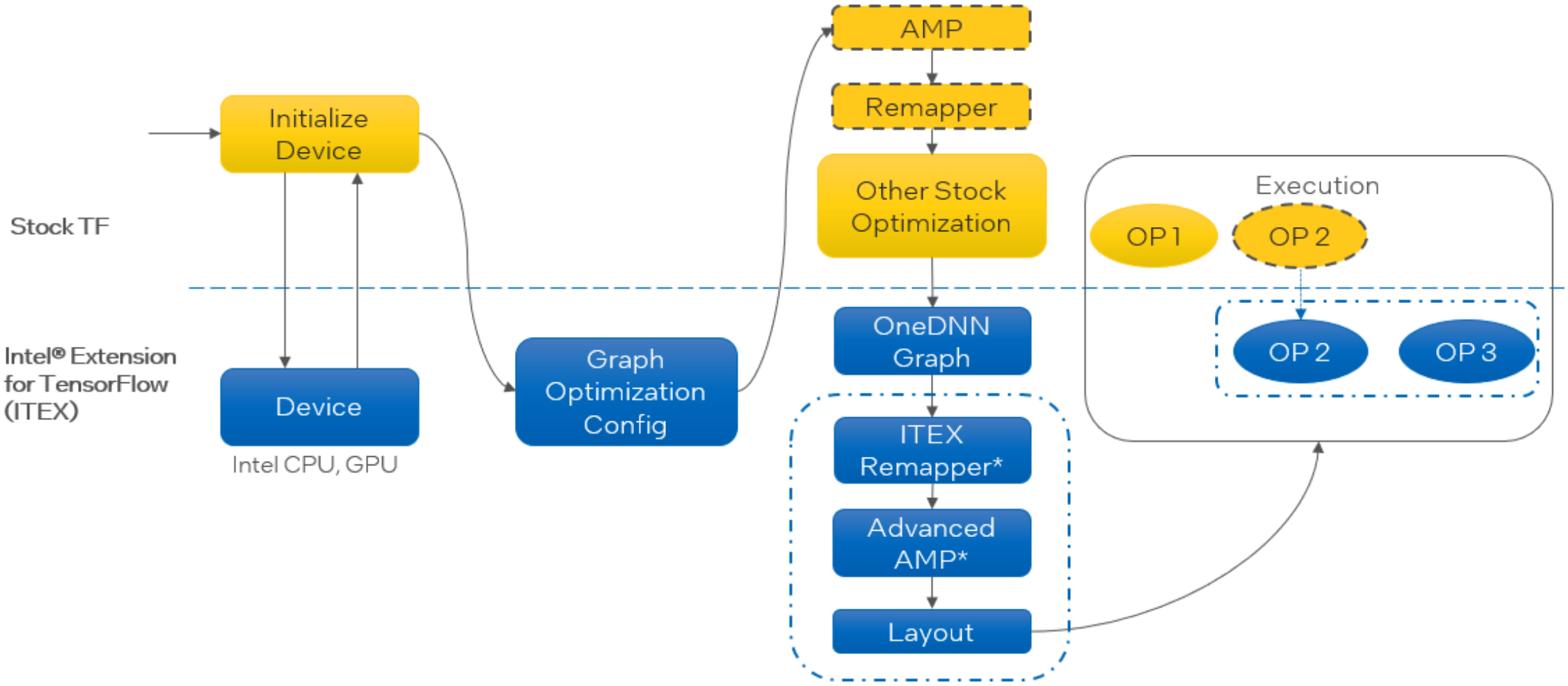
- Provide Intel XPU implementation to benefit Intel GPU and CPU users with the best performance and user experience.
- Provide Full Functionality (100% op converge) / General Optimization (OOB) for Intel GPU based on DPC++ software stack.



Intel® Extension for TensorFlow* pip packages and dependencies

Architecture

AMP: Auto Mixed Precision

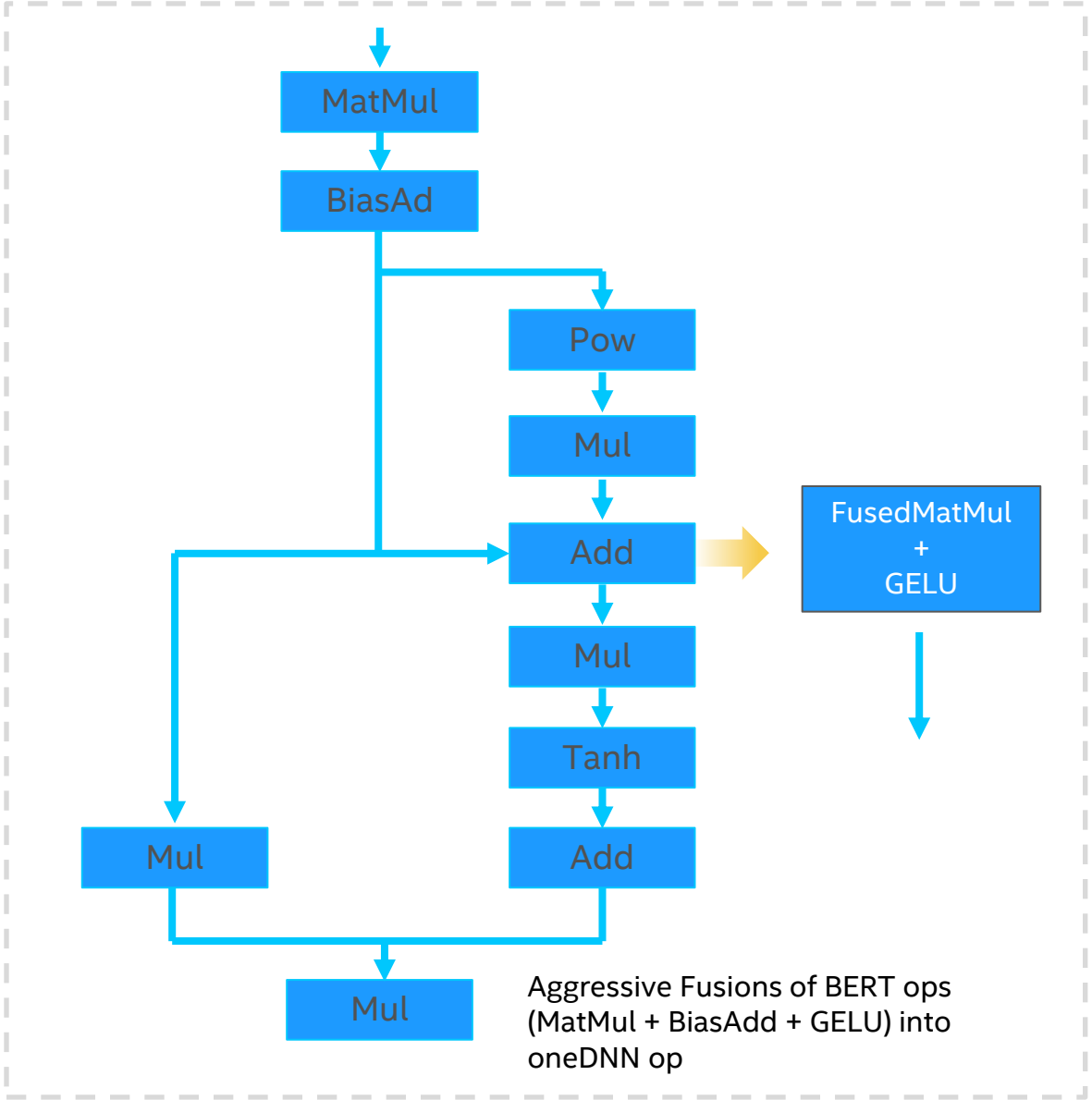


Intel® Optimizations for TensorFlow*

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

oneDNN Integration with TensorFlow

- Replaces compute-intensive standard TF ops with highly optimized custom oneDNN ops
- Aggressive op fusions to improve performance of Convolutions and Matrix Multiplications
- Primitive caching to reduce overhead of calling oneDNN



How to use IPEX - Infer

CPU

FP32

```
import intel_extension_for_pytorch as ipex
model = ipex.optimize(model)
```

BF16

```
import intel_extension_for_pytorch as ipex
model, optimizer = ipex.optimize(model,
    optimizer=optimizer, dtype=torch.bfloat16)

for batch_idx, (data, target) in enumerate(train_loader):
    optimizer.zero_grad()
    with torch.cpu.amp.autocast():
        output = model(data)
        ...
```

FP16

GPU

```
import intel_extension_for_pytorch as ipex
model = model.to("xpu")
data = data.to("xpu")
model = ipex.optimize(model)
```

```
model = model.to("xpu")
data = data.to("xpu")
model = ipex.optimize(model, dtype=torch.bfloat16)

with torch.no_grad():
    with torch.xpu.amp.autocast(enabled=True, dtype=torch.bfloat16):
        model(data)
```

```
model = model.to("xpu")
data = data.to("xpu")
model = ipex.optimize(model, dtype=torch.float16)

with torch.no_grad():
    with torch.xpu.amp.autocast(enabled=True, dtype=torch.float16):
        model(data)
```

How to use IPEX - Train

CPU

FP32

```
import intel_extension_for_pytorch as ipex
model = ipex.optimize(model)
```

BF16

```
import intel_extension_for_pytorch as ipex
model = ipex.optimize(model, dtype=torch.bfloat16)

with torch.no_grad():
  with torch.cpu.amp.autocast():
    model(data)
```

GPU

```
import intel_extension_for_pytorch as ipex
model = model.to("xpu")
criterion = criterion.to("xpu")
model, optimizer = ipex.optimize(model, optimizer=optimizer)

for batch_idx, (data, target) in enumerate(train_loader):
    data = data.to("xpu")
    target = target.to("xpu")
```

```
import intel_extension_for_pytorch as ipex
model = model.to("xpu")
criterion = criterion.to("xpu")
model, optimizer = ipex.optimize(model, optimizer=optimizer, dtype=torch.bfloat16)

for batch_idx, (data, target) in enumerate(train_loader):
    optimizer.zero_grad()
    data = data.to("xpu")
    target = target.to("xpu")
    with torch.xpu.amp.autocast(enabled=True, dtype=torch.bfloat16):
        output = model(data)
        loss = criterion(output, target)
```

How to use ITEX

ITEX

Explicit

FP32

```
import intel_extension_for_tensorflow as itex
backend = "GPU" #CPU, GPU or AUTO
itex.set_backend(backend)
```

BF16

Python API	Environment Variable

`import intel_extension_for_tensorflow as itex`
 	
`auto_mixed_precision_options = itex.AutoMixedPrecosionOptions()`
`auto_mixed_precision_options.data_type = itex.BFLOAT16`
 	
`graph_options = itex.GraphOptions()`
`graph_options.auto_mixed_precision_options=auto_mixed_precision_options`
`graph_options.auto_mixed_precision = itex.ON`
 	
`config = itex.ConfigProto(graph_options=graph_options)`
`itex.set_backend("gpu", config)`
	`export ITEX_AUTO_MIXED_PRECISION=1`
	`export ITEX_AUTO_MIXED_PRECISION_DATA_TYPE="BFLOAT16"`

How to Check Running on CPU or GPU

Run command: **DNNL_VERBOSE=1** python itex_sample.py

```
onednn_verbose,info,oneDNN v2.5.0 (commit e008c47c7f2e839ff64c206a21c82059a227717c)
onednn_verbose,info,cpu,runtime:DPC++
onednn_verbose,info,cpu,isa:Intel 64
onednn_verbose,info,gpu,runtime:DPC++
onednn_verbose,info,cpu,engine,0,backend:OpenCL,name:Genuine Intel(R) CPU $0000%@,driver_version:2021.13.11
onednn_verbose,info,gpu,engine,0,backend:Level Zero,name:Intel(R) Graphics [0x020a],driver_version:1.1.20495
onednn_verbose,info,gpu,engine,1,backend:Level Zero,name:Intel(R) Graphics [0x020a],driver_version:1.1.20495
onednn_verbose,info,prim_template:operation,engine,primitive,implementation,prop_kind,memory_descriptors,attribu
tes,auxiliary,problem_desc,exec_time
onednn_verbose,info,gpu,binary_kernels:enabled
onednn_verbose,exec,gpu:18446744073709551615,reorder,ocl:ref:any,undef,src_f32::blocked:cdba:f0
dst_f32:p:blocked:Acdb16a:f0,,,3x3x2x2,1.42993
onednn_verbose,exec,gpu:18446744073709551615,convolution,ocl:gen9:blocked,forward_training,src_f32::blocked:acdb
:f0 wei_f32:p:blocked:Acdb16a:f0 bia_undef::undef::f0 dst_f32::blocked:acdb:f0,attr-scratchpad:user
,alg:convolution_direct,mb1_ic3oc3_ih5oh5kh2sh1dh0ph0_iw5ow5kw2sw1dw0pw0,0.352051
onednn_verbose,exec,gpu:18446744073709551615,eltwise,ocl:ref:any,forward_training,data_f32::blocked:abcd:f0
diff_undef::undef::f0,attr-scratchpad:user ,alg:eltwise_relu alpha:0 beta:0,1x5x5x3,0.297852
```

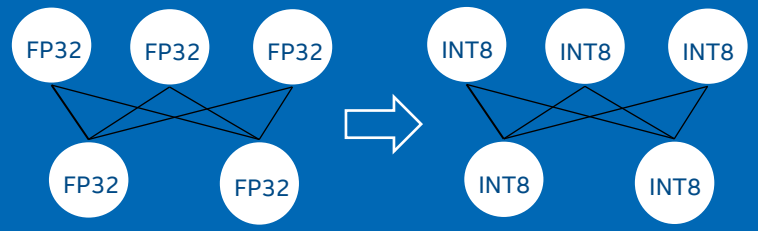
Installation

	ITEX
Common	pip install --upgrade tensorflow
CPU	pip install --upgrade intel-extension-for-tensorflow[cpu]
GPU	pip install --upgrade intel-extension-for-tensorflow[gpu]

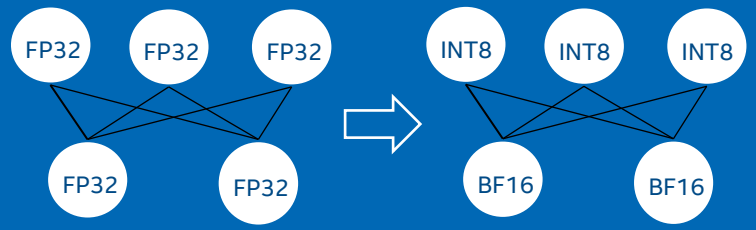
	IPEX
CPU	pip install torch torchvision torchaudio --extra-index-url https://download.pytorch.org/whl/cpu pip install intel_extension_for_pytorch
GPU	pip install torch==2.0.1a0 torchvision==0.15.2a0 intel_extension_for_pytorch==2.0.110+xpu -f https://developer.intel.com/ipex-whl-stable-xpu-idp

Deep Learning Inference Optimization

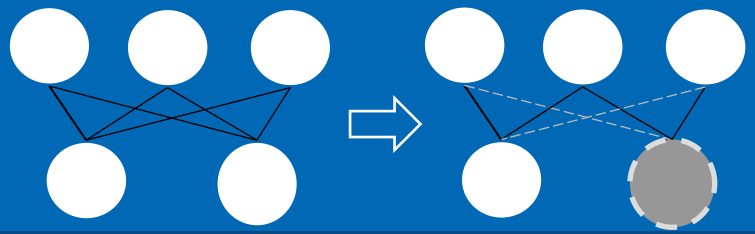
Quantization



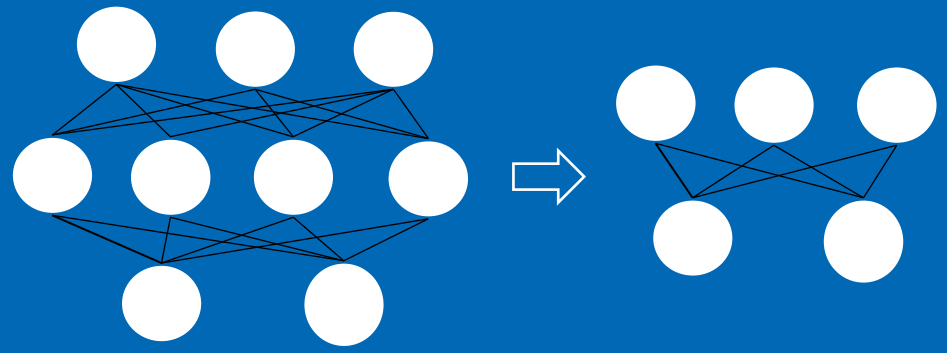
Mix Precision Graphic Optimization



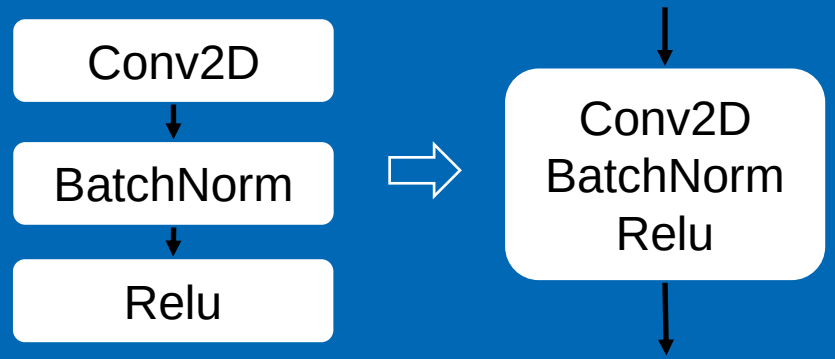
Pruning



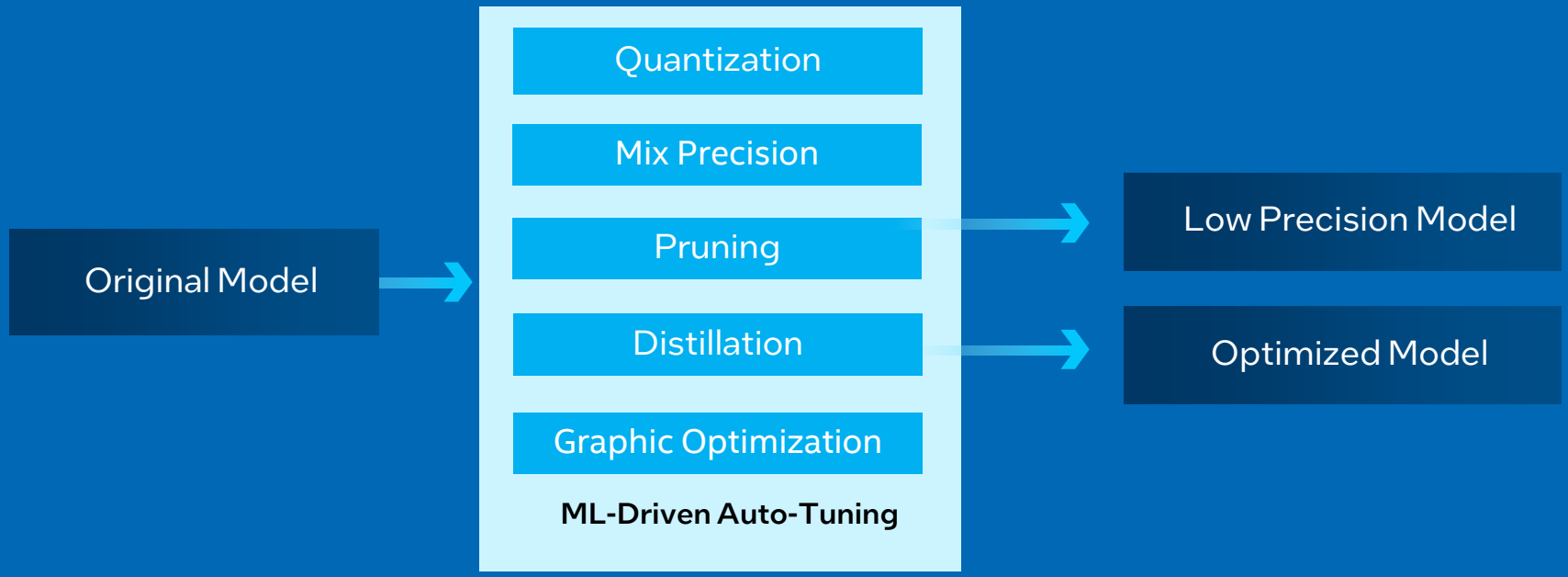
Knowledge Distillation



Graphic Optimization



Intel® Neural Compressor



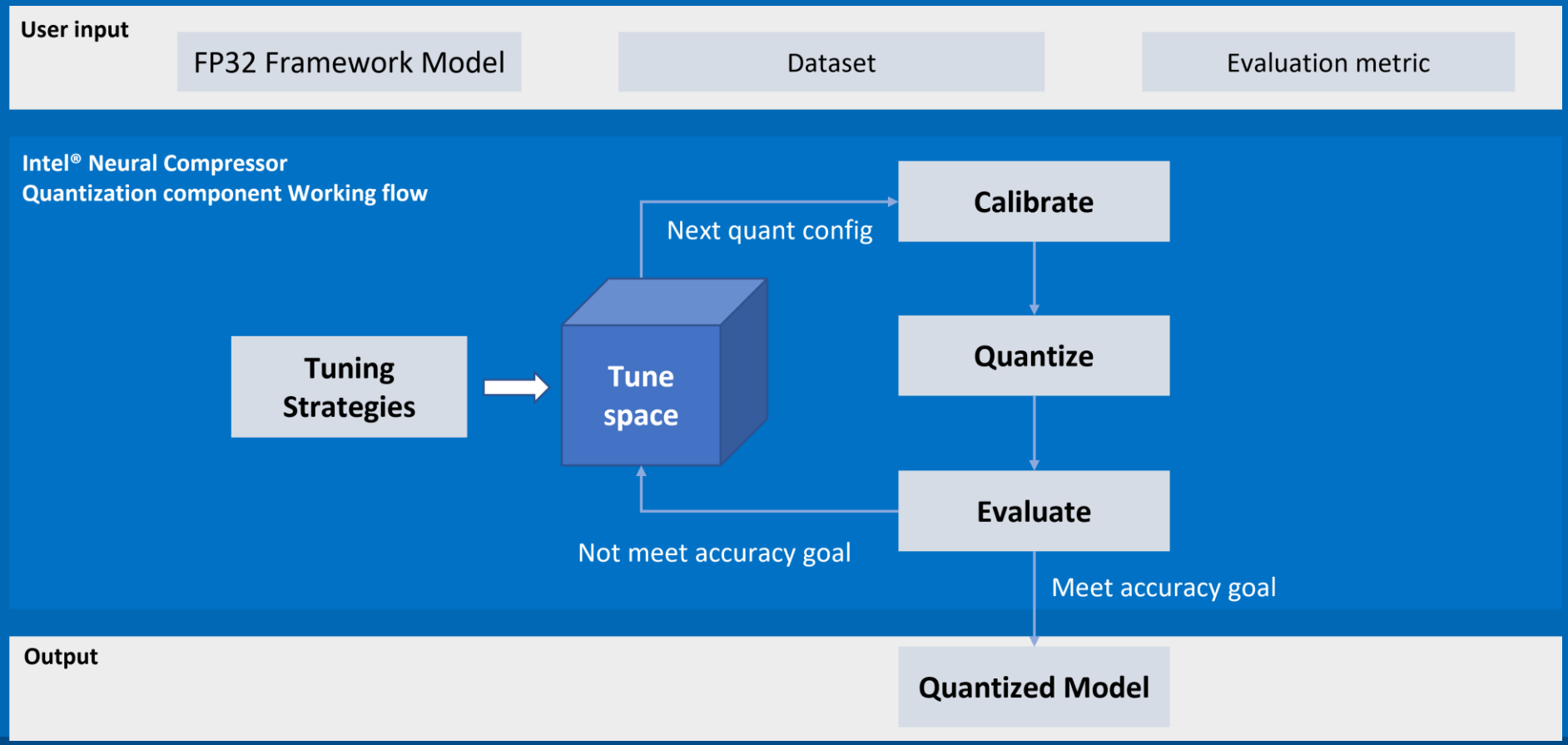
~~Accuracy Loss~~

~~Operation Complex~~

Intel® Neural Compressor (https://github.com/intel/neural-compressor) is designed to use automatic accuracy-aware tuning strategies to help user **easily & quickly** find out the best optimization methods above.

Workflow

- **High efficiency** for quantization with automatic accuracy-driven tuning strategy.
  - Less time to achieve the quantization goals comparing with handcraft work.
  - Tuning strategy: basic, Bayesian, random, Sigopt.
- **Flexible** workflow.
  - Dataset & metric & evaluation function not only support built-in but also customized one.



30x AI 性能提升，基于优化软件和下一代至强处理器

TensorFlow

oneAPI one DNN

Intel® Neural Compressor

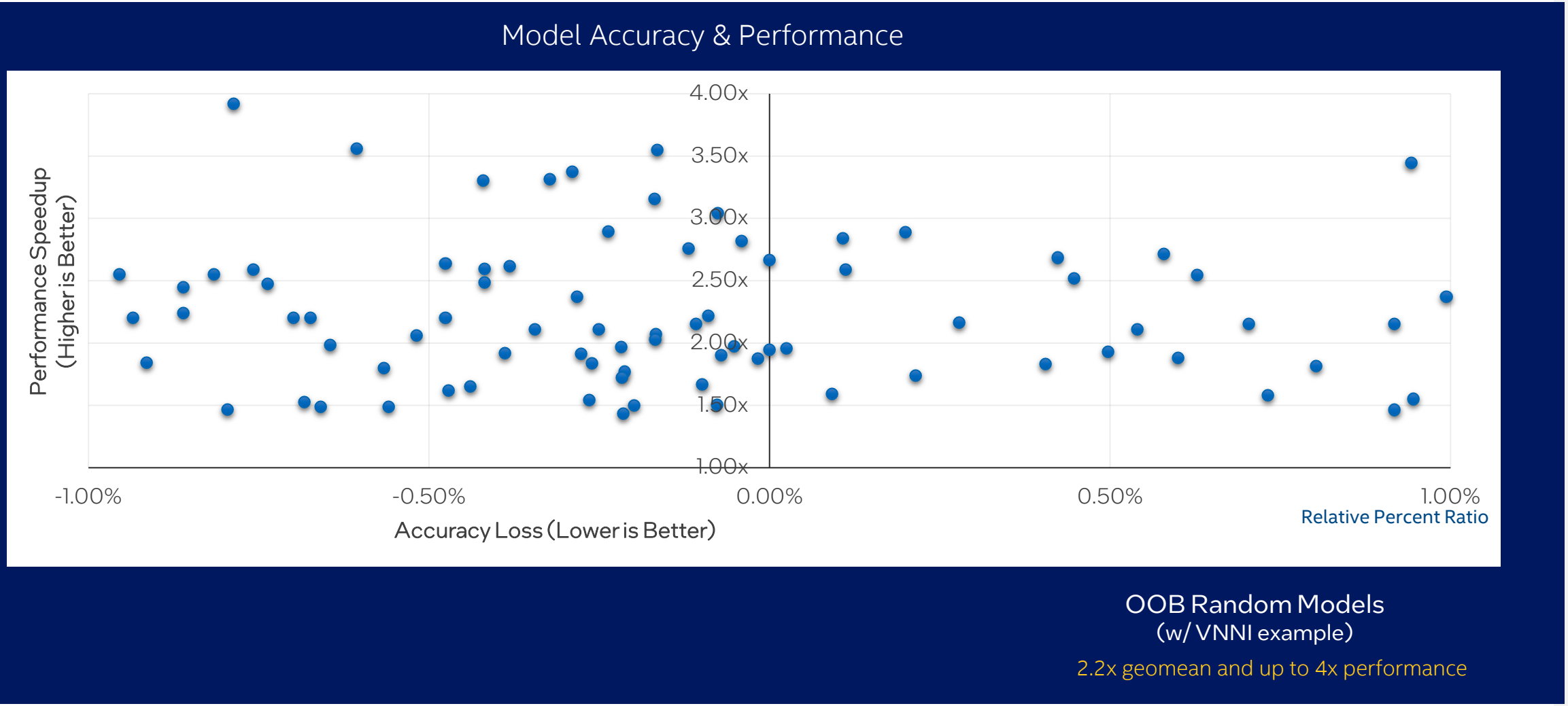
13.06	20.54	81.66	394
Baseline	1.5x	3.9x	4.8x
(FP32) Official TensorFlow on 3rd Gen Intel® Xeon® Scalable Processor	(FP32) TensorFlow with oneDNN enabled	(int8) Model quantization with Intel® Neural compressor	(int8) Intel® AMX optimization on Sapphire Rapids

3rd Gen Intel® Xeon™ Scalable Processors — 4th Gen Xeon

Results may vary. See www.intel.com/InnovationEventClaims for workloads and configurations.

SSD-ResNet-34 Inference Throughput (Batch Size =1)
For workloads and configurations visit www.intel.com/InnovationEventClaims. Results may vary.

Performance



Status

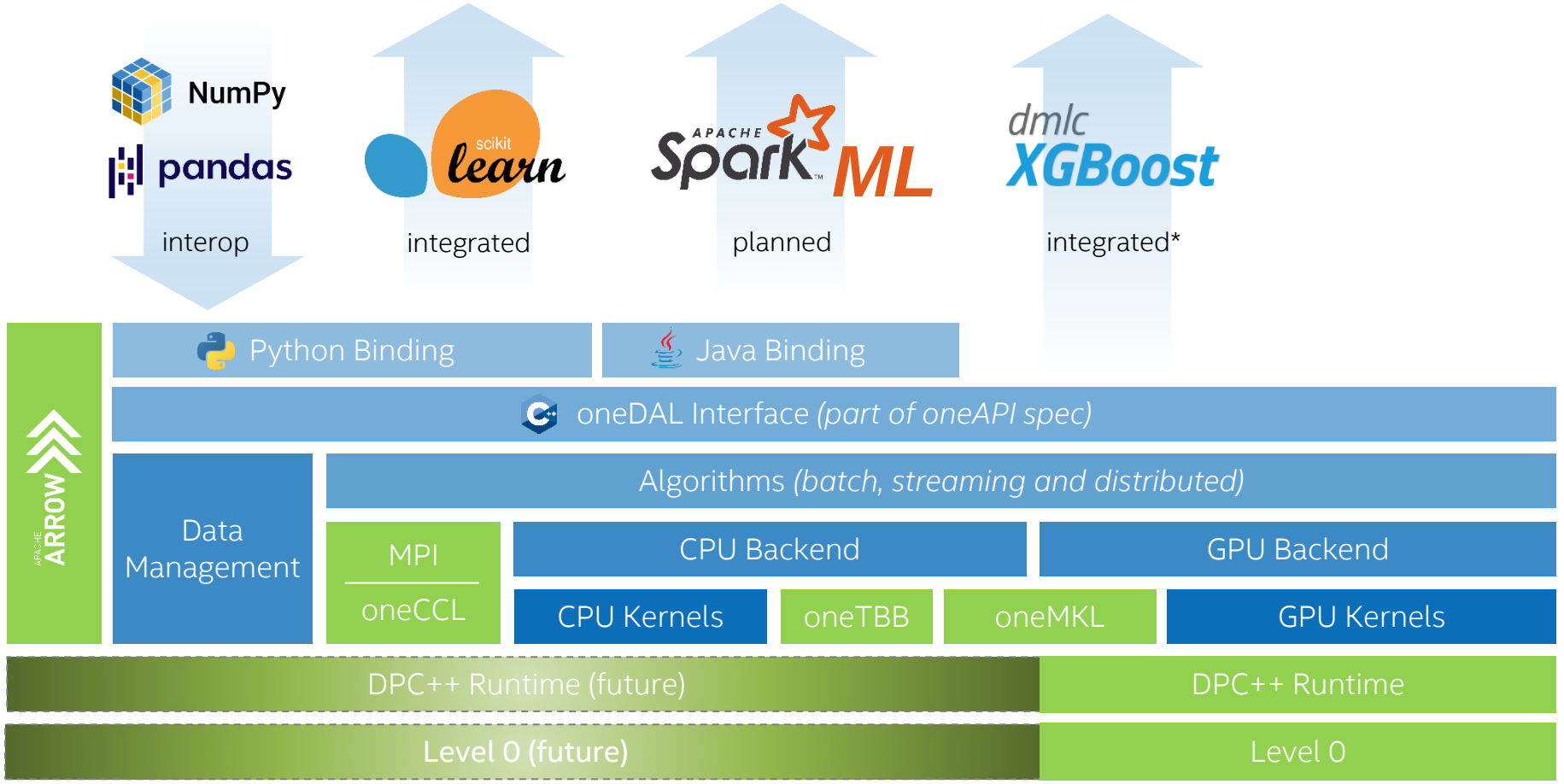
- Open source: https://github.com/intel/neural-compressor
- 2.3 is released.
- Automatically Tuning to reduce the accuracy loss.
- Support frameworks: Tensorflow, Pytorch, MXNet, ONNX RT
- 420+ public models are verified with good performance improvement and limited accuracy loss; 10000+ random models including Stable Diffusion, large language models
- Adaption: Huggingface/Optimum, ONNX model Zoo, PyTorch ecosystem.
- Customer: Microsoft, Alibaba, Tencent, ByteDance, etc.
- Publication (19+): https://github.com/intel/neural-compressor/blob/master/docs/source/publication_list.md

Quantization with Accuracy Aware Tuning

```
from neural_compressor.quantization import fit
from neural_compressor.config import PostTrainingQuantConfig, TuningCriterion, AccuracyCriterion

eval_dataloader = …
eval_func = …

tuning_criterion = TuningCriterion(max_trials=600)
accuracy_criterion = AccuracyCriterion(
      higher_is_better=True,
      criterion='relative',
      tolerable_loss=0.01
      )

conf = PostTrainingQuantConfig(
                              backend="ipex",
                              approach="static",
                              tuning_criterion=tuning_criterion,
                              accuracy_criterion = accuracy_criterion)
q_model = fit(model, conf=conf, calib_dataloader=eval_dataloader, eval_func=eval_func)
q_model.save("path/to/save")
```

Intel® oneAPI Data Analytics Library (oneDAL)

Framework Interfaces & Software Stack

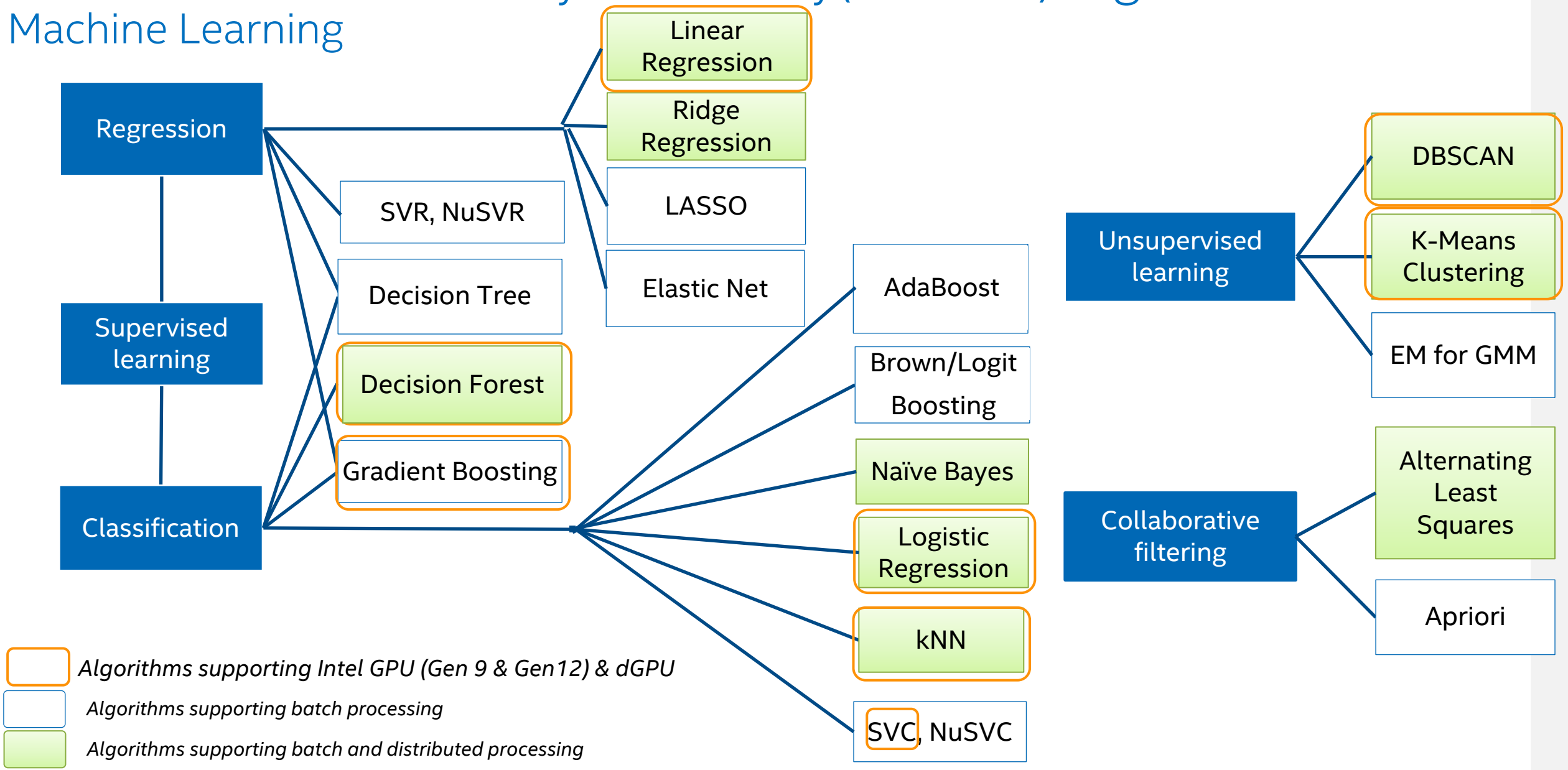


API designed to be hardware and vendor independent

Relies on Data Parallel C++ (DPC++) and C++17

Intel® oneAPI Data Analytics Library(oneDAL) Algorithms

Machine Learning



Intel® Extension for Scikit-learn

Common Scikit-learn

```
from sklearn.svm import SVC

X, Y = get_dataset()

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

Scikit-learn mainline

Scikit-learn with Intel CPU opts

```
from sklearnex import patch_sklearn
patch_sklearn()

from sklearn.svm import SVC

X, Y = get_dataset()

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

Intel® extension for 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

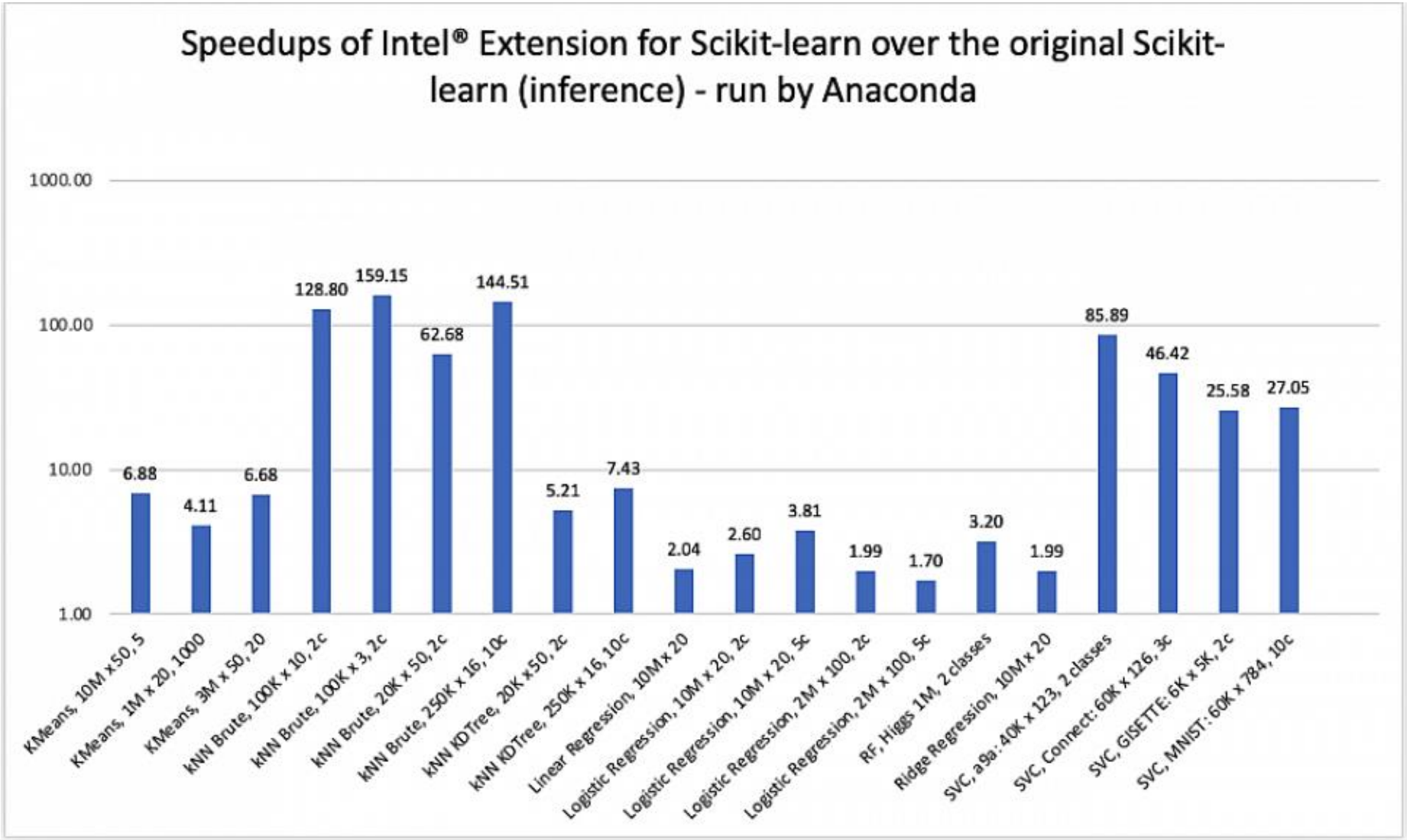
- Directly from the script:

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

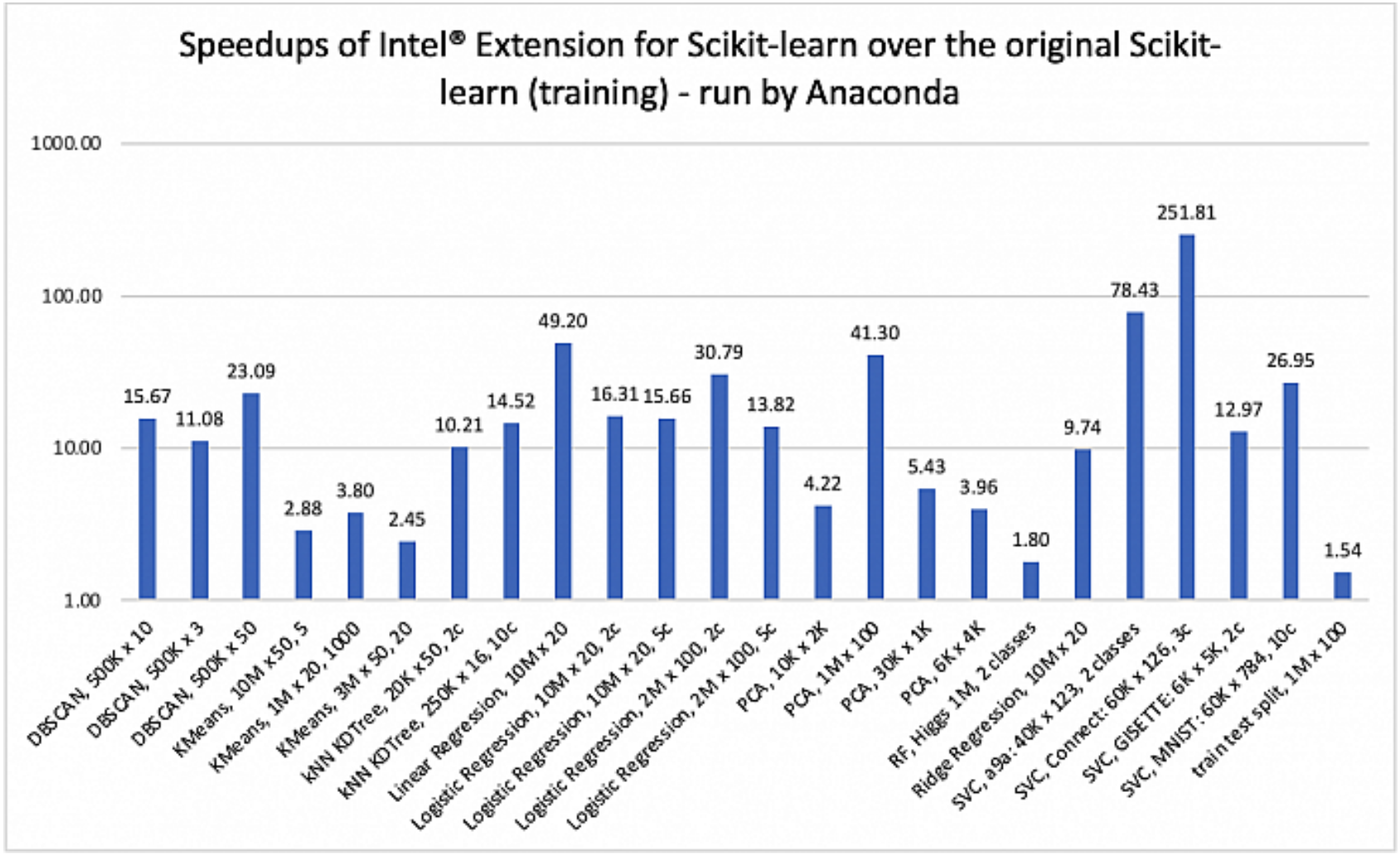
- Through global patching to enable patching for your scikit-learn installation for all further runs:

```
python sklearnex.glob patch_sklearn
```

Intel® Extension for scikit-learn (Inference)



Intel® Extension for scikit-learn (Training)



ML Performance with Intel-optimized XGBoost *

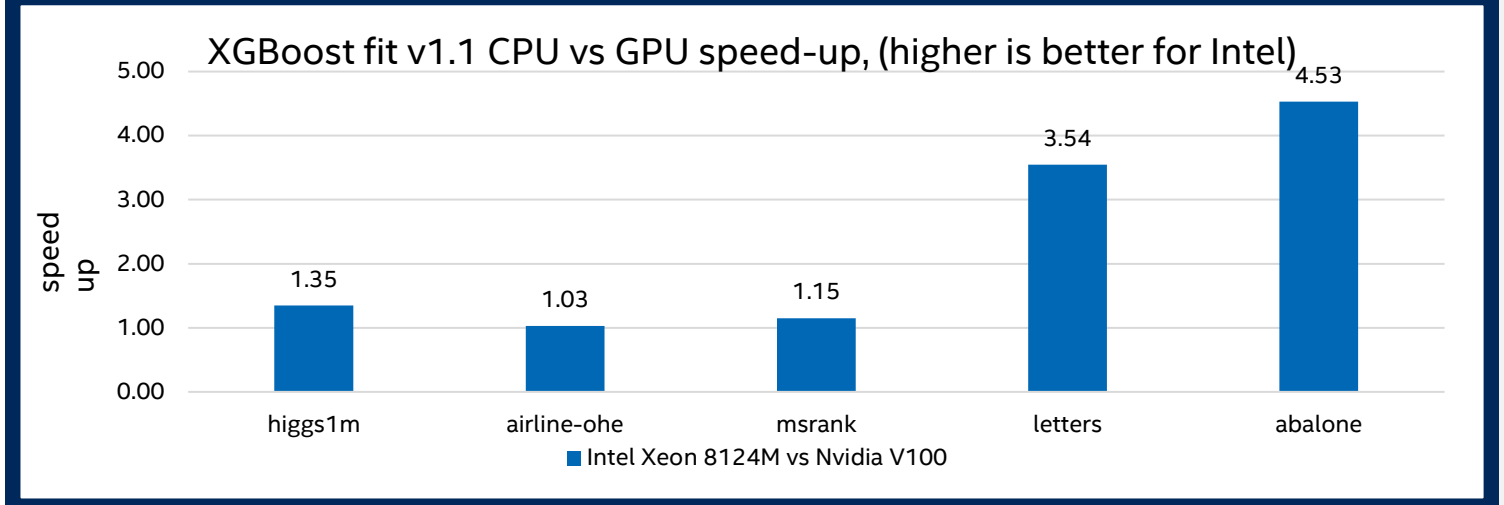
- Intel's contribution to XGBoost project on GitHub https://github.com/dmlc/xgboost
- Memory prefetching, nestled and advanced parallelism, usage of uint8

+ Reducing memory consumption

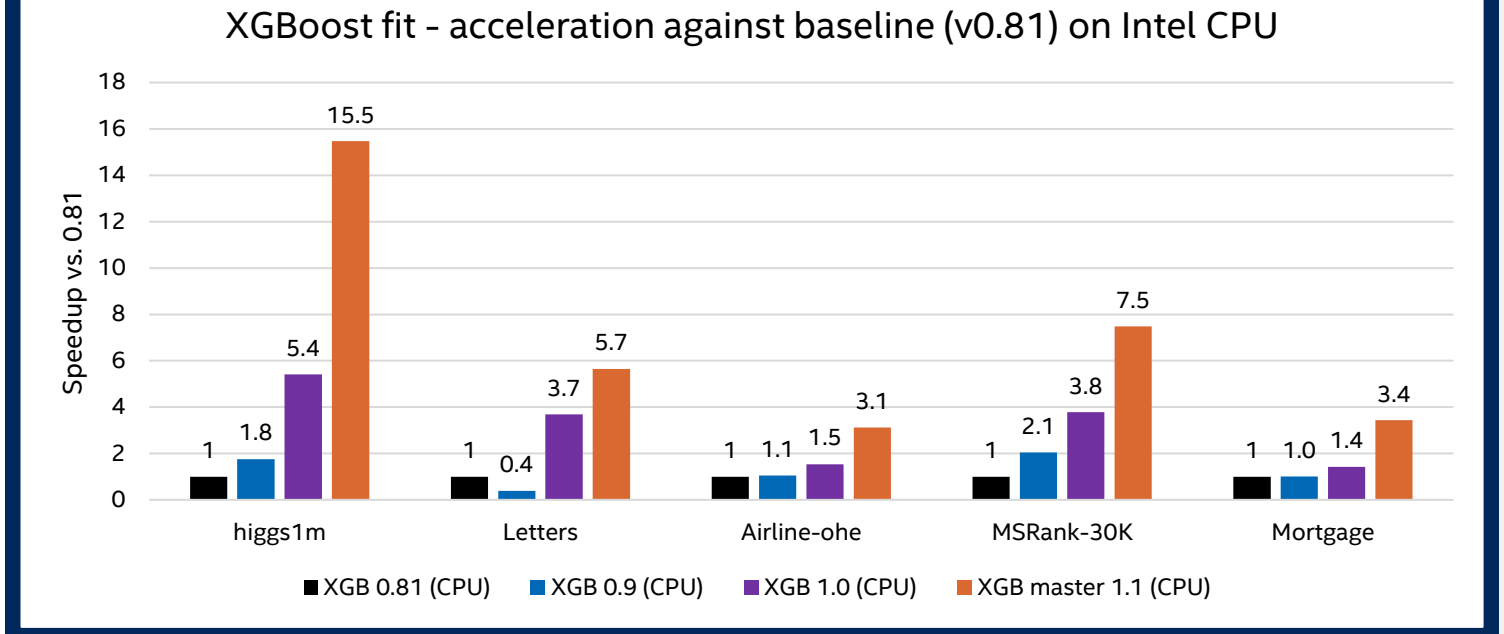
memory, Kb	Airline	Higgs1m
Before	28311860	1907812
#5334	16218404	1155156
reduced:	1.75	1.65

*Measured March 2021

XGBoost CPU vs. GPU

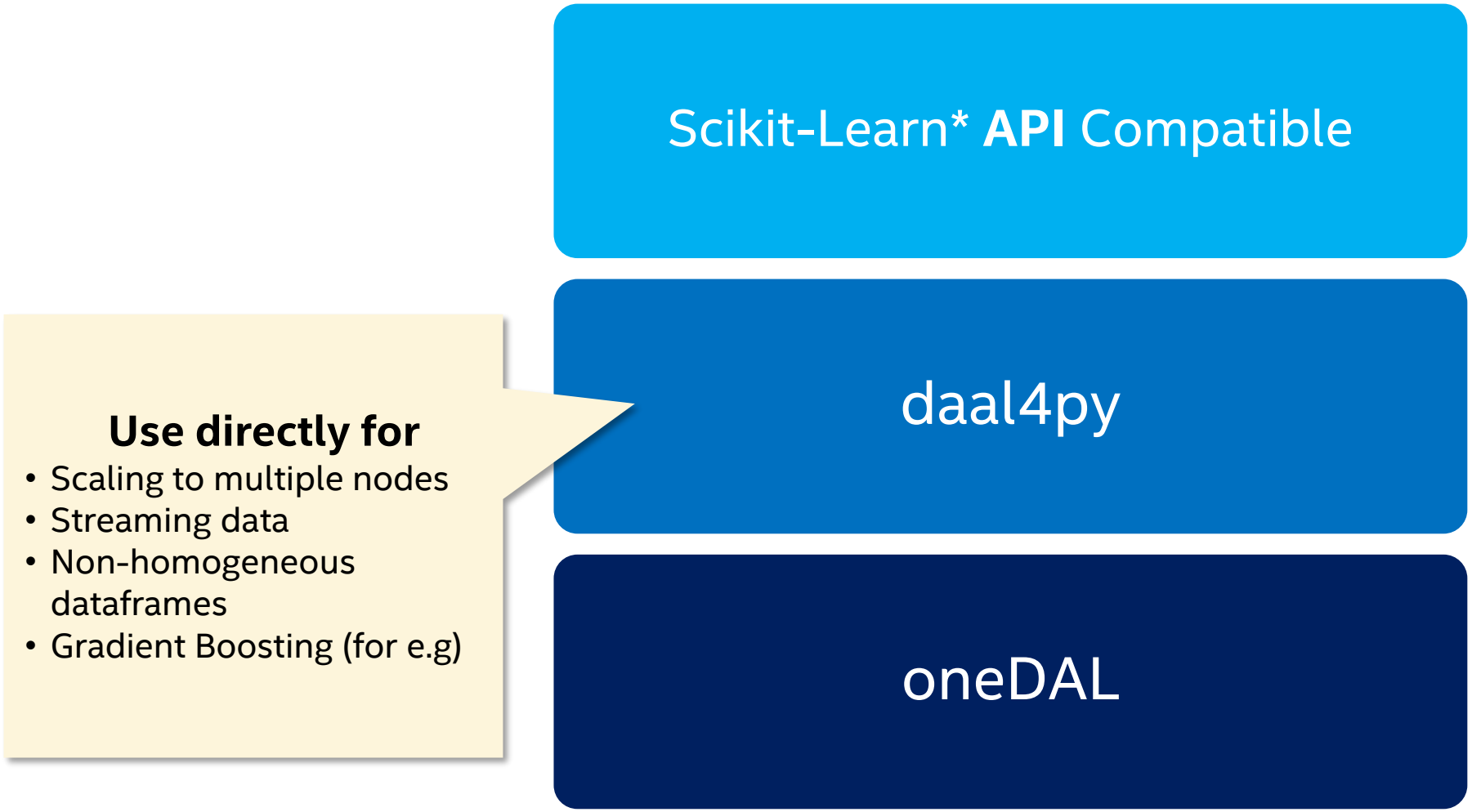


XGBoost fit CPU acceleration ("hist" method)



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

oneAPI Data Analytics Library (oneDAL)

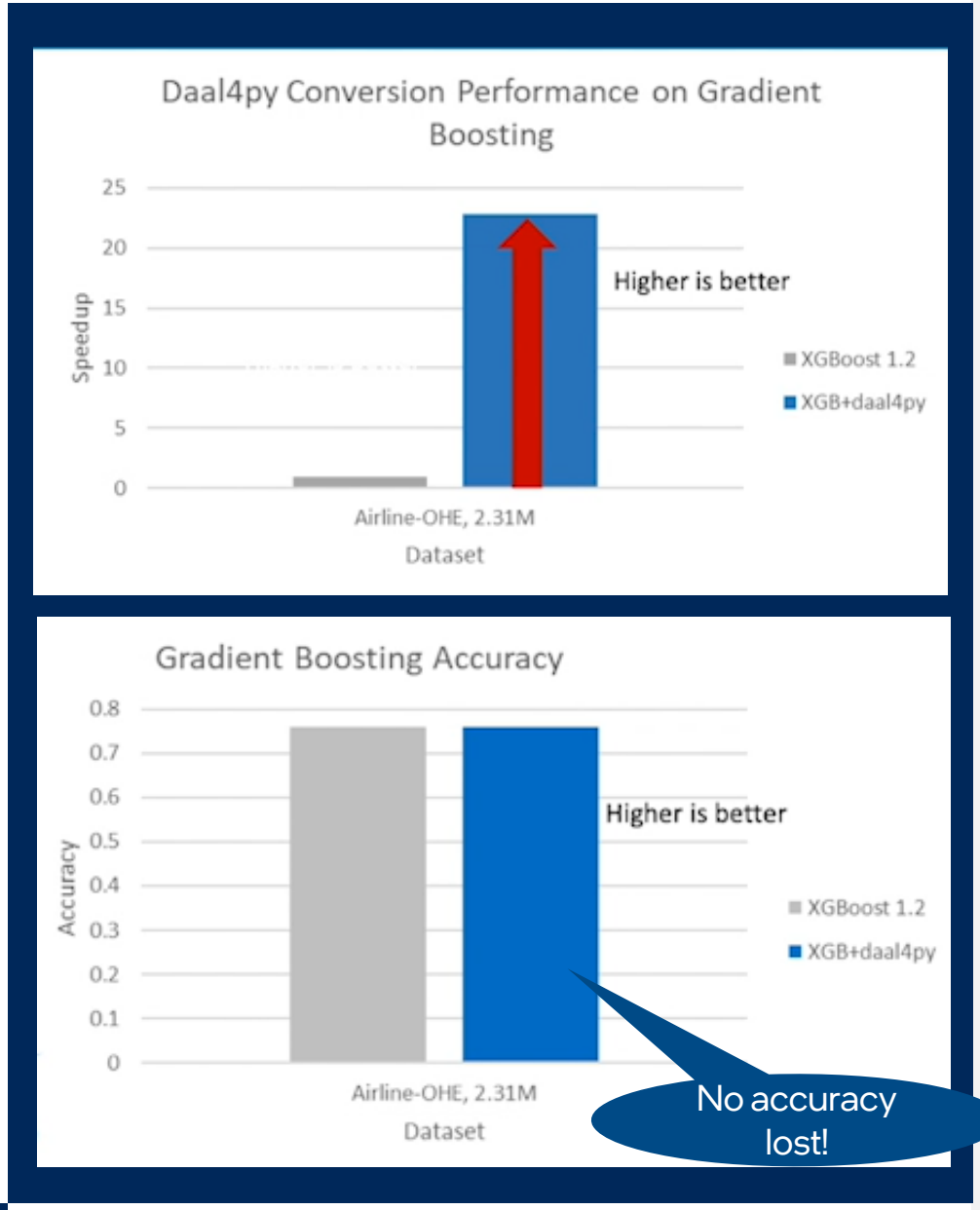


XGBoost and LightGBM Prediction Acceleration with Daal4Py

- 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



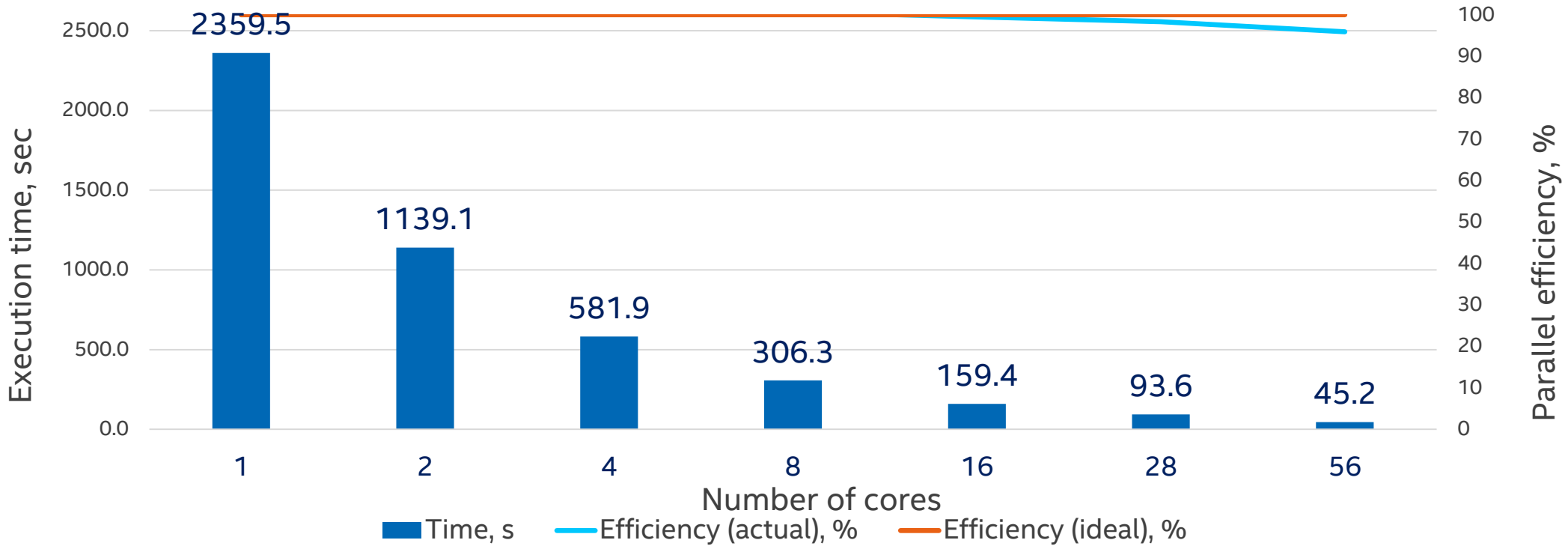
Distributed K-Means using daal4py

```
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

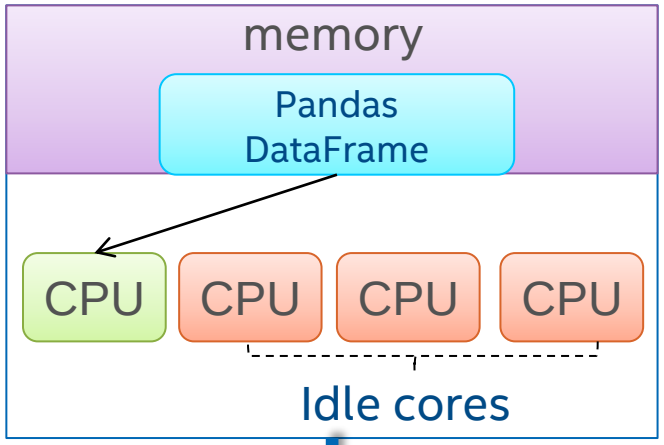
(10M samples, 10 features, 100 clusters, 100 iterations, float32)

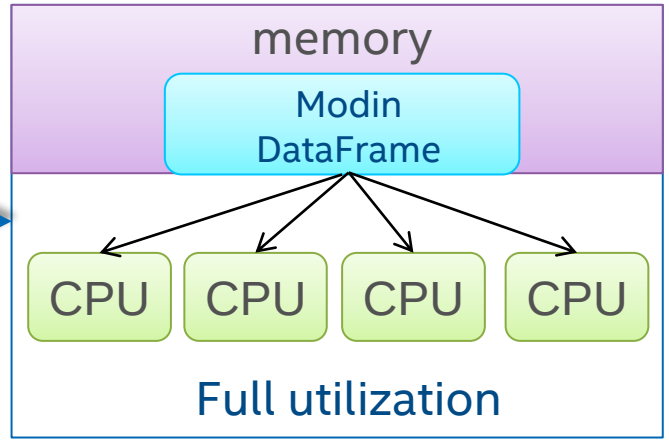


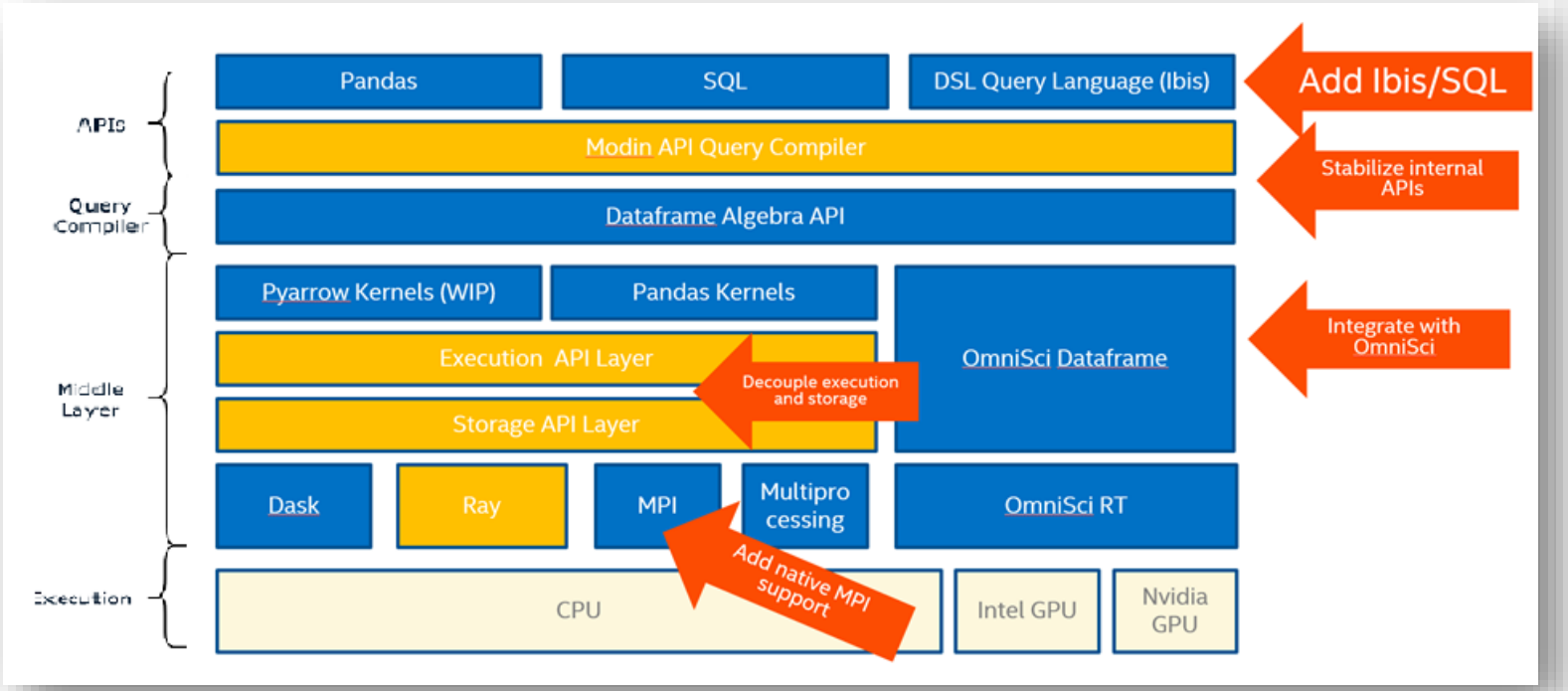
Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex.
Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component 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.

Modin

- Usable and Scalable







To use Modin, replace the pandas import

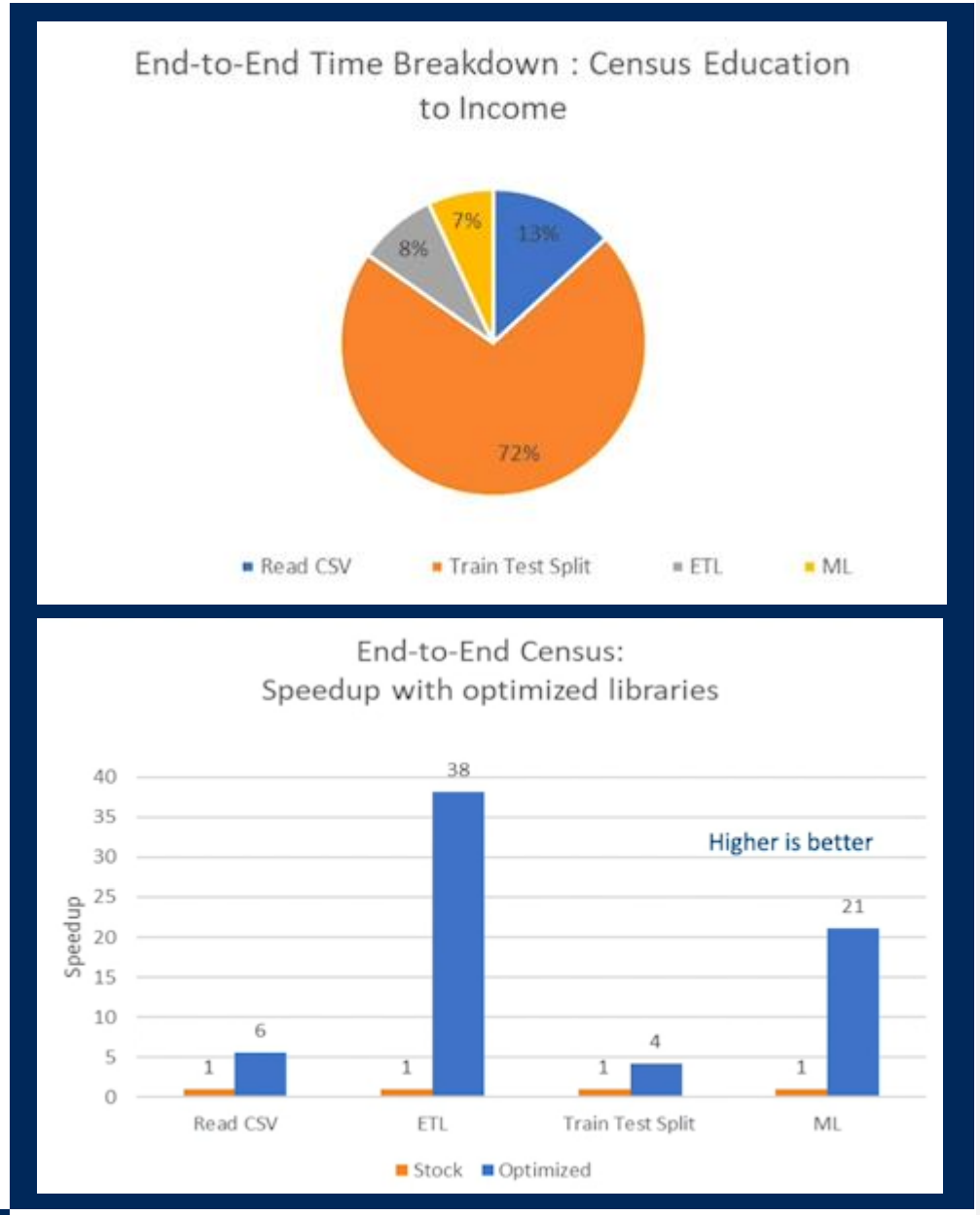
To install the Intel Distribution of Modin, alter the command to include conda forge dependencies:
`conda create -n aikit-modin intel-aikit-modin -c intel -c conda-forge`

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

End-to-End Data Pipeline Acceleration*

- **Workload:** Train a model using 50 years of Census dataset from IPUMS.org to predict education based on income
- **Solution:** Intel Modin for data ingestion and ETL, Daal4Py and Intel scikit-learn for model training and prediction
- **Performance 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**

*Measured March 2021

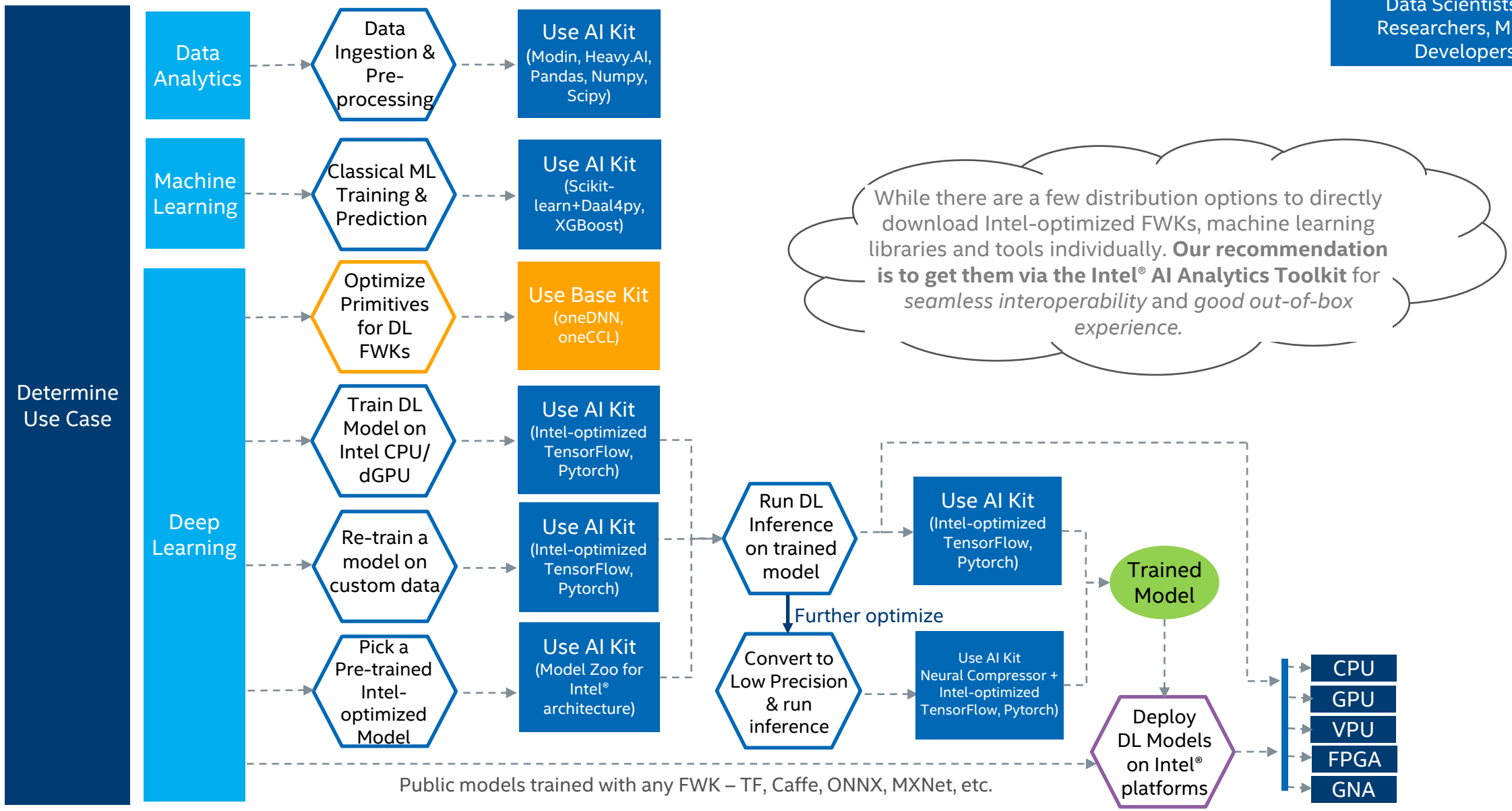


For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.
See backup for configuration details.

AI Development Workflow

Native Code Developers, Framework Developers

Data Scientists, AI Researchers, ML/DL Developers



While there are a few distribution options to directly download Intel-optimized FWKs, machine learning libraries and tools individually. **Our recommendation is to get them via the Intel® AI Analytics Toolkit** for *seamless interoperability* and *good out-of-box experience.*

AI Kit = Intel® AI Analytics Toolkit
Base Kit = Intel® oneAPI Base Toolkit

- 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.
- Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.
- Your costs and results may vary.
- Intel technologies may require enabled hardware, software or service activation.
-


intel