Choose the Best Accelerated Technology

AI optimizations on Intel Architecture (IA)

Mecit Gungor – DL Software Engineer
Feb 11 2022

All information provided in this deck is subject to change without notice.
Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Intel's oneAPI Ecosystem

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

oneAPI

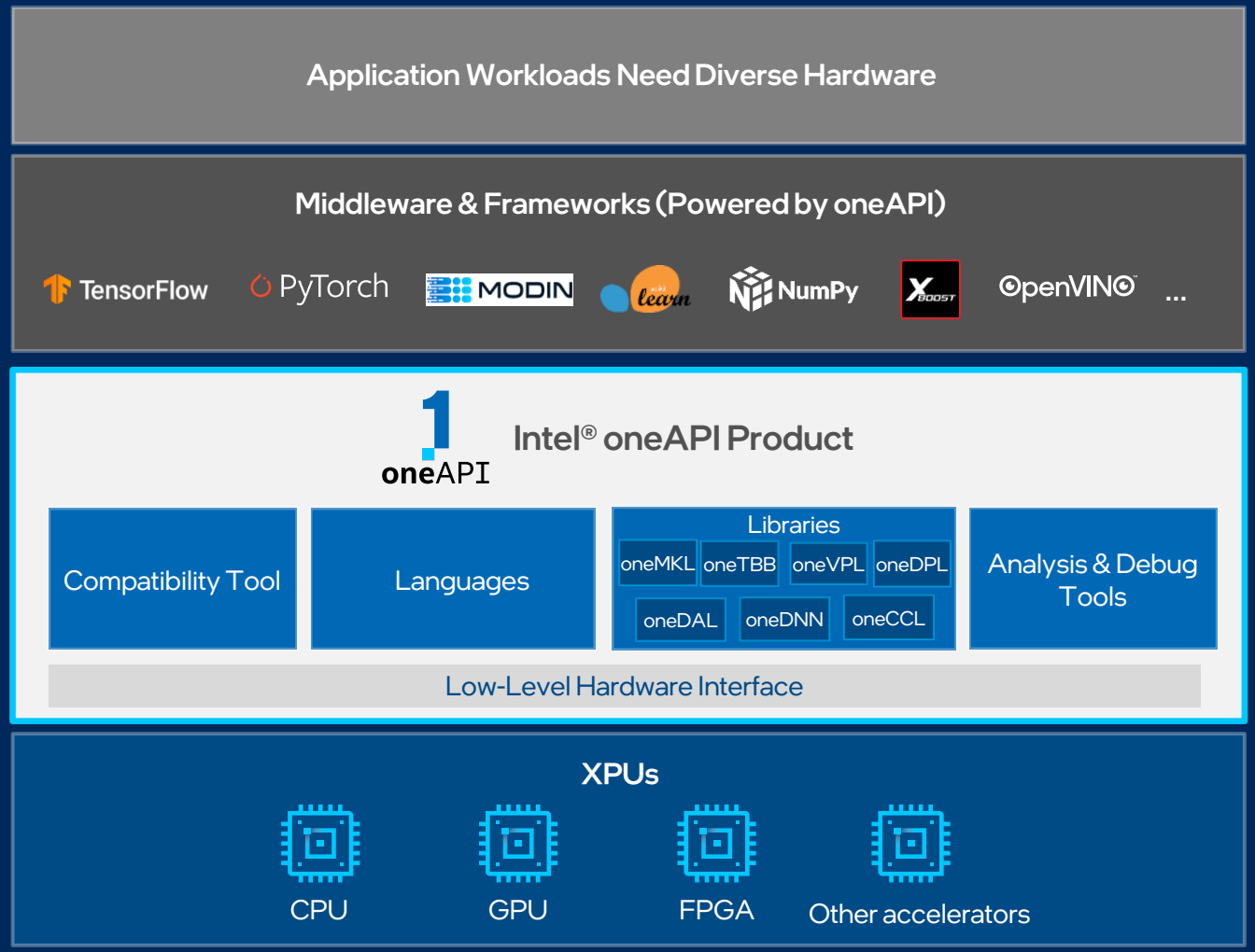
A cross-architecture language based on C++ and SYCL standards

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.

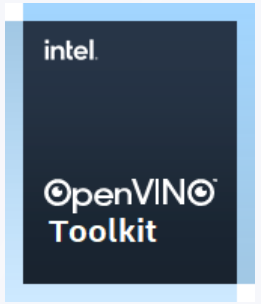
Intel® Software Development Toolkits for AI & Analytics in oneAPI Ecosystem

Toolkits **Powered by oneAPI**

Intel® AI Analytics Toolkit

Accelerate machine learning & data science pipelines with optimized DL frameworks & high-performing Python libraries

Data Scientists, AI Researchers, DL/ML Developers

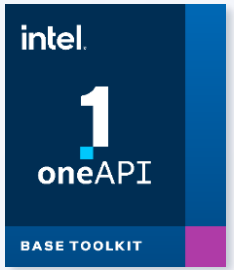


Intel® Distribution of OpenVINO™ Toolkit

Deploy high performance inference & applications from edge to cloud

AI Application, Media, & Vision Developers

Intel® oneAPI Toolkits



Intel® oneAPI Base Toolkit

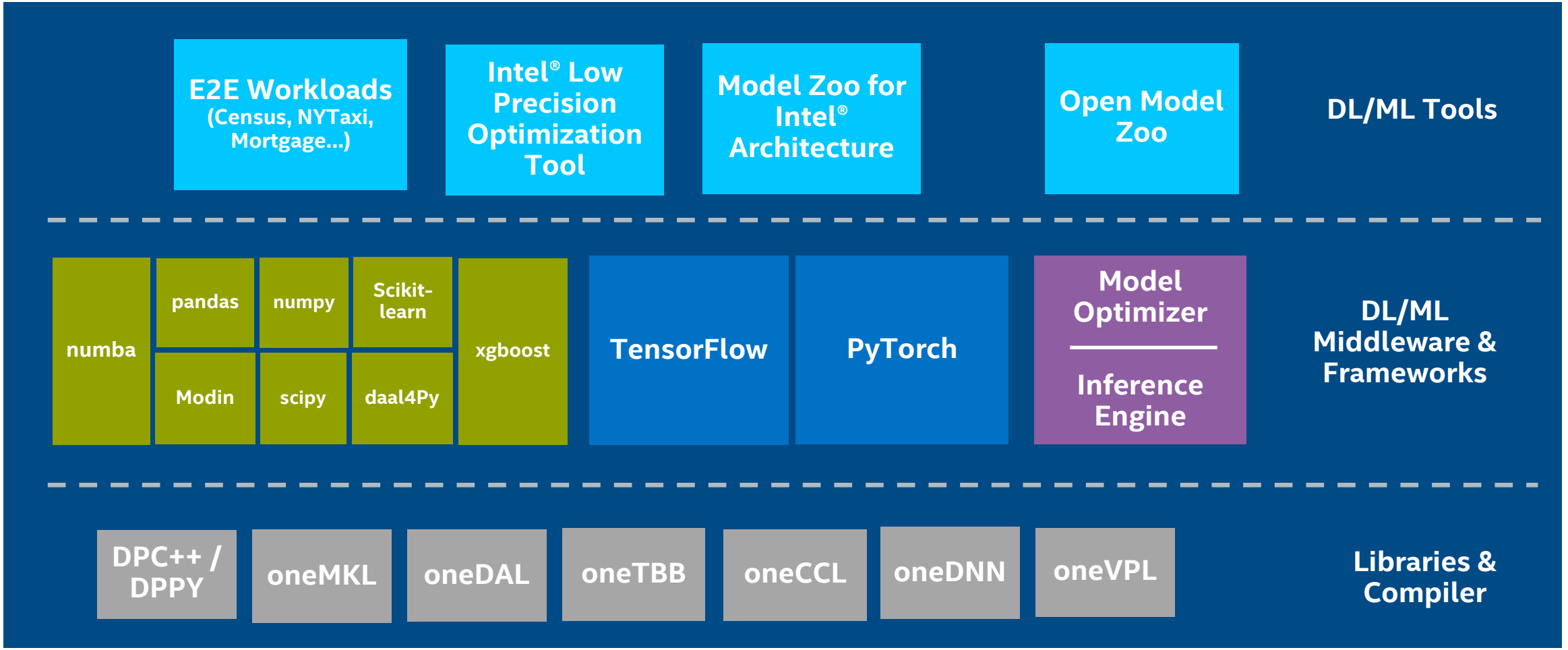
Includes oneDNN, oneCCL & oneDAL

Optimize primitives for algorithms and framework development

DL Framework Developers - Optimize algorithms for Machine Learning & Analytics

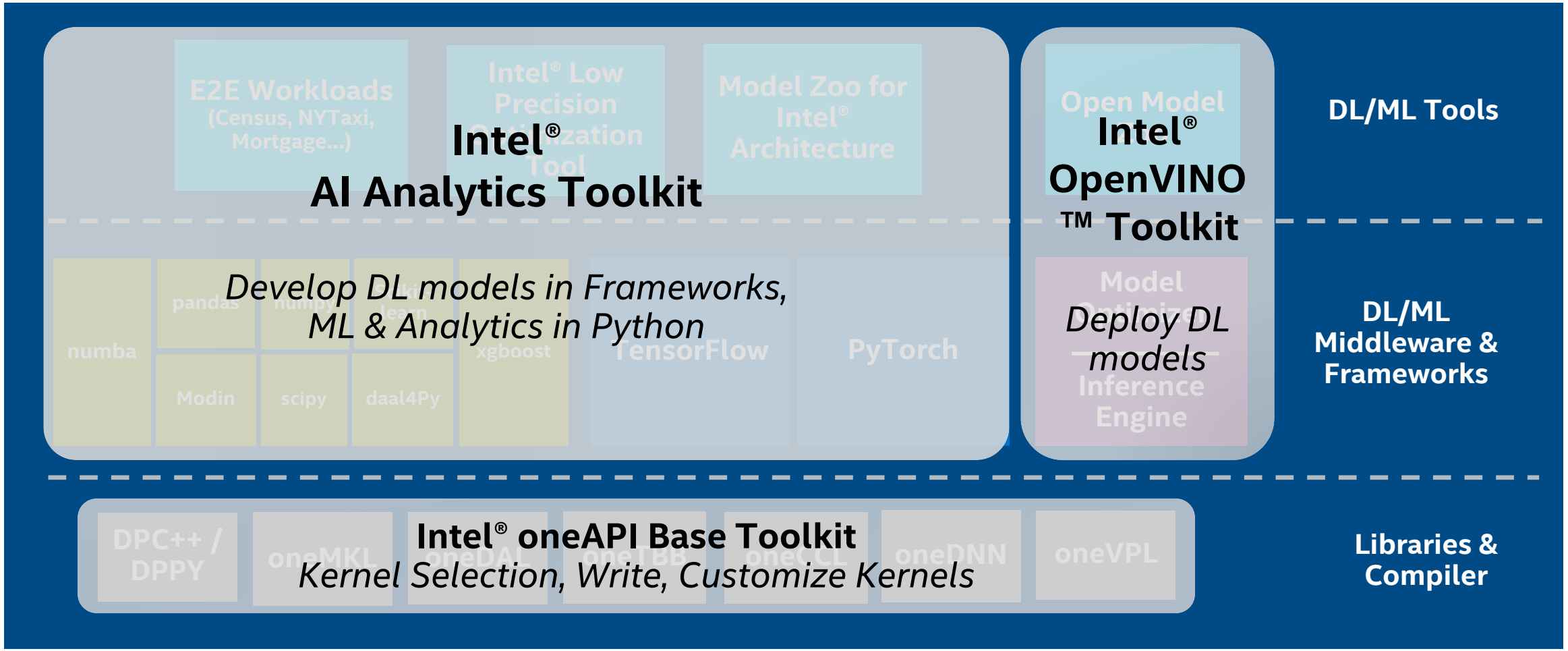
AI Software Stack for Intel® XPUs

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



AI Software Stack for Intel® XPUs

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



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

Intel® AI Analytics Toolkit

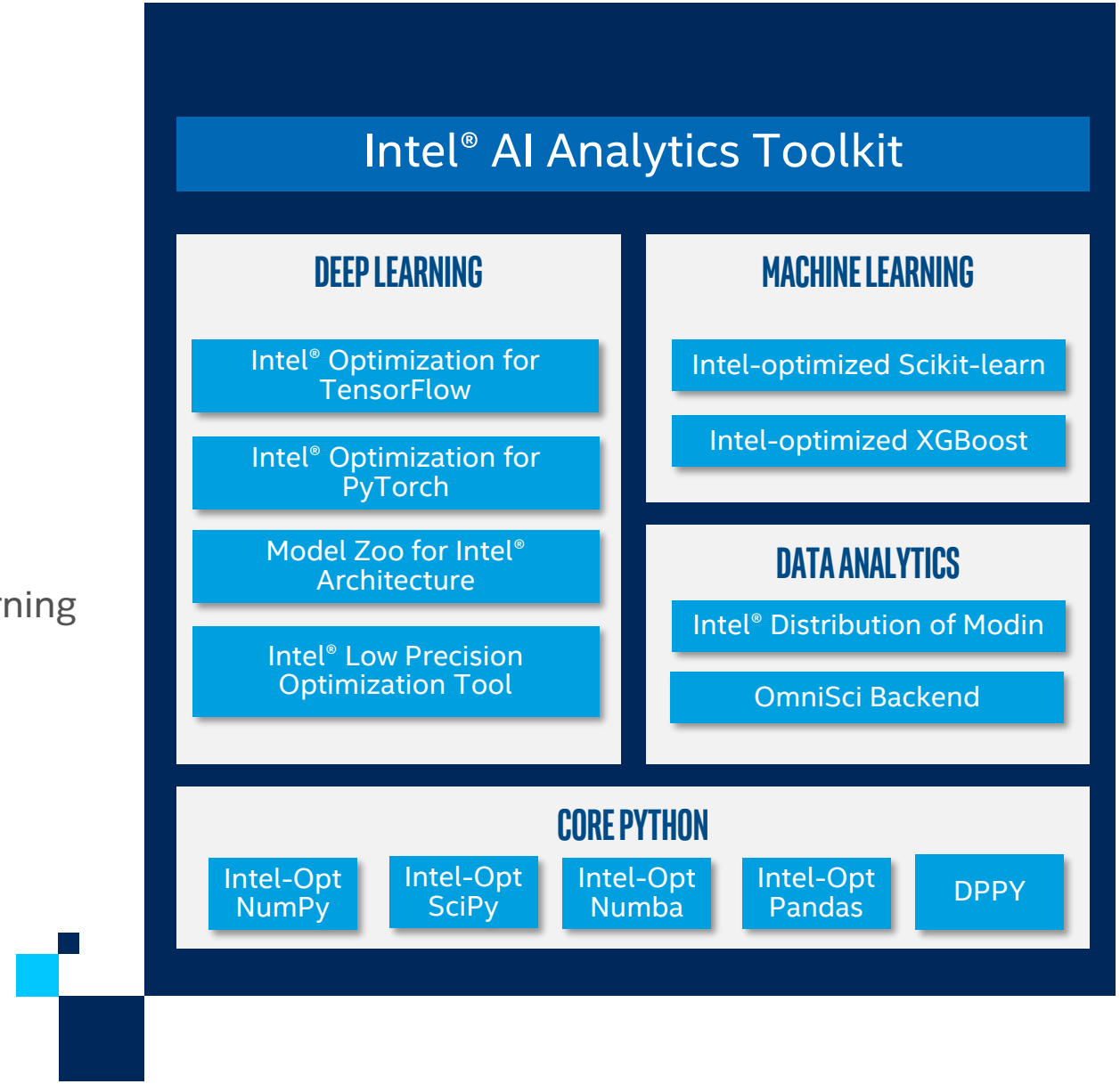
Powered by oneAPI

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

Choose the best accelerated technology – the software doesn't decide for you



Learn More: intel.com/oneAPI-AIKit

Key Features & Benefits

- Accelerate end-to-end AI and Data Science pipelines and achieve drop-in acceleration with optimized Python tools built using oneAPI libraries (i.e. oneMKL, oneDNN, oneCCL, oneDAL, and more)
- Achieve high-performance for deep learning training and inference with Intel-optimized versions of TensorFlow and PyTorch, and low-precision optimization with support for fp16, int8 and bfloat16
- Expedite development by using the open-source pre-trained deep learning models optimized by Intel for best performance via Model Zoo for Intel® Architecture
- Enable distributed training through Torch-CCL and support of the standards based Horovod library
- Seamlessly scale Pandas workflows across multi-node dataframes with Intel® Distribution of Modin, accelerate analytics with performant backends such as OmniSci
- Increase machine learning model accuracy and performance with algorithms in Scikit-learn and XGBoost optimized for Intel architectures
- Supports cross-architecture development (Intel® CPUs/GPUs) and compute

Model Zoo for Intel® Architecture (IA)

Intel model zoo provides recommendations to achieve best parallelism for different models

- **For many popular open-source machine learning models, Model Zoo has**
  - links to pre-trained models (*benchmarks* folder)
  - sample scripts (*models* and *benchmark* folders)
  - best practices (*docs* folder)
  - step-by-step tutorials (*docs* folder)
- **Purpose of the Model Zoo**
  - Learn which AI topologies and workloads Intel has optimized to run on its hardware
  - Benchmark the performance of optimized models on Intel hardware
  - Get started efficiently running optimized models in containers or on bare metal

Run-time options	**Recommendations**			
	ResNet50	InceptionV3	ResNet101	Wide and deep
Batch Size	128. Regardless of the hardware			512
Hyperthreading	Enabled. Turn on in BIOS. Requires a restart.			
intra_op_parallelism_threads	#physical cores per socket	#physical cores per socket	# all physical cores	1 to physical cores
inter_op_parallelism_threads	1	1	2	1
Data Layout	NCHW			NC
NUMA Controls	numactl --cpunodebind=0 --membind=0			all
KMP_AFFINITY	KMP_AFFINITY=granularity=fine,verbose,compact,1,0			
KMP_BLOCKTIME	1			
OMP_NUM_THREADS	# intra_op_parallelism_threads		#physical cores per socket	1 to physical cores

Getting Started with Intel® AI Analytics Toolkit

Overview	Installation	Hands on	Learning	Support
▪ Visit Intel® 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

Choose the Best Accelerated Technology

Intel® Acceleration for Classical Machine Learning

All information provided in this deck is subject to change without notice.
Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Intel® AI Analytics Toolkit

Powered by oneAPI

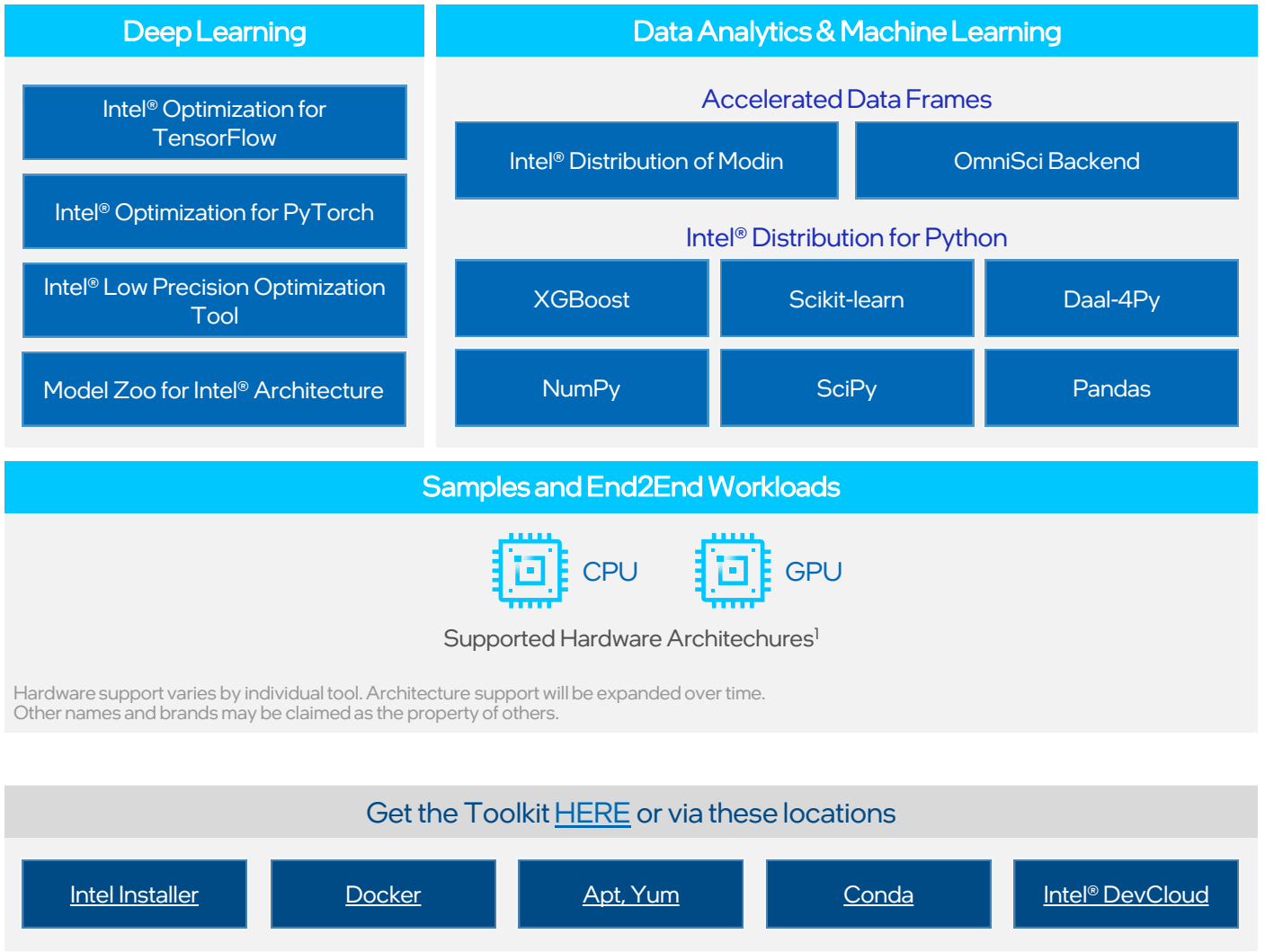
Accelerate end-to-end AI and data analytics pipelines with libraries optimized for Intel® architectures

Who Uses It?

Data scientists, AI researchers, ML and DL developers, AI application developers

Top Features/Benefits

- Deep learning performance for training and inference with Intel optimized DL frameworks and tools
- Drop-in acceleration for data analytics and machine learning workflows with compute-intensive Python packages



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

Back to Domain-specific Toolkits for Specialized Workloads

Intel® Distribution for Python

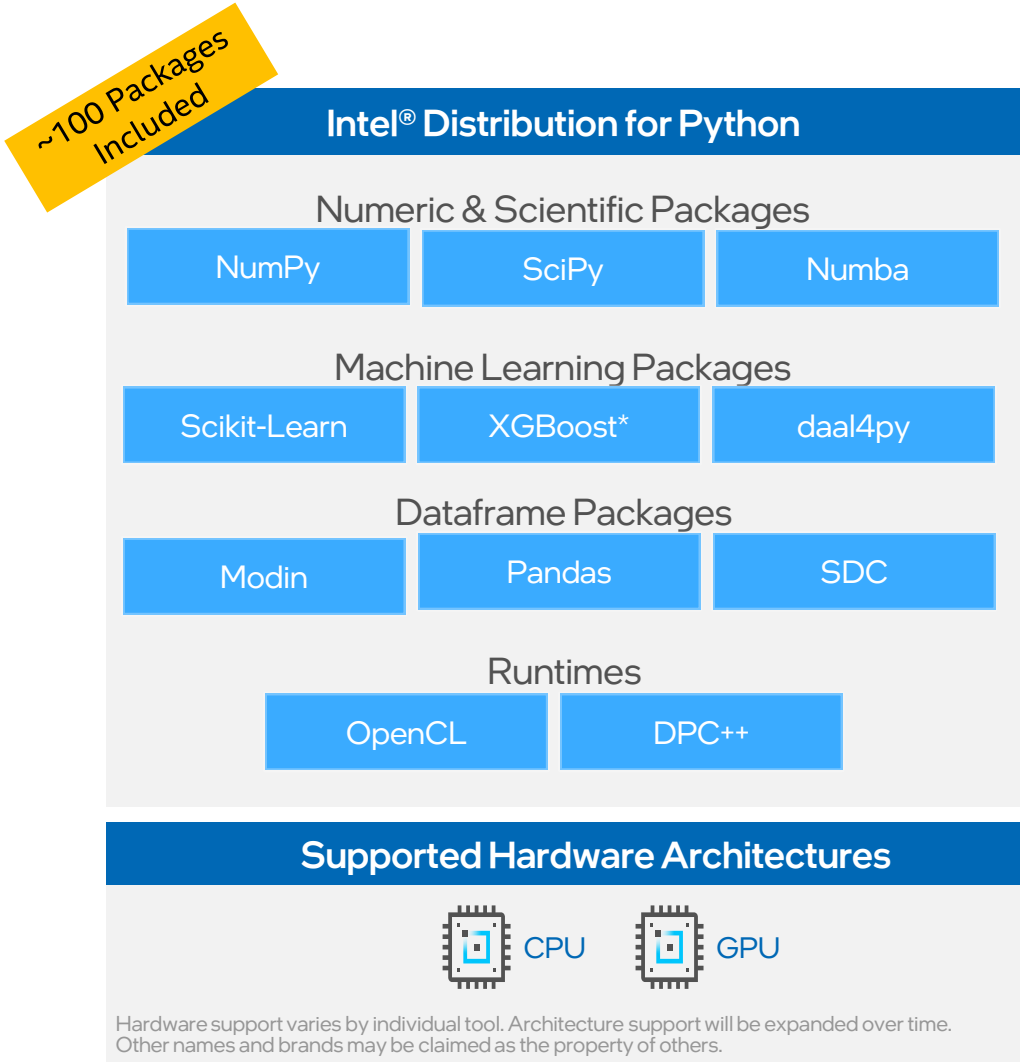
oneAPI Powered

Develop fast, performant Python code with this set of essential computational packages

Who Uses It?

- Machine Learning Developers, Data Scientists, and Analysts can implement performance-packed, production-ready scikit-learn algorithms
- Numerical and Scientific Computing Developers can accelerate and scale the compute-intensive Python packages NumPy, SciPy, and mpi4py
- High-Performance Computing (HPC) Developers can unlock the power of modern hardware to speed up your Python applications

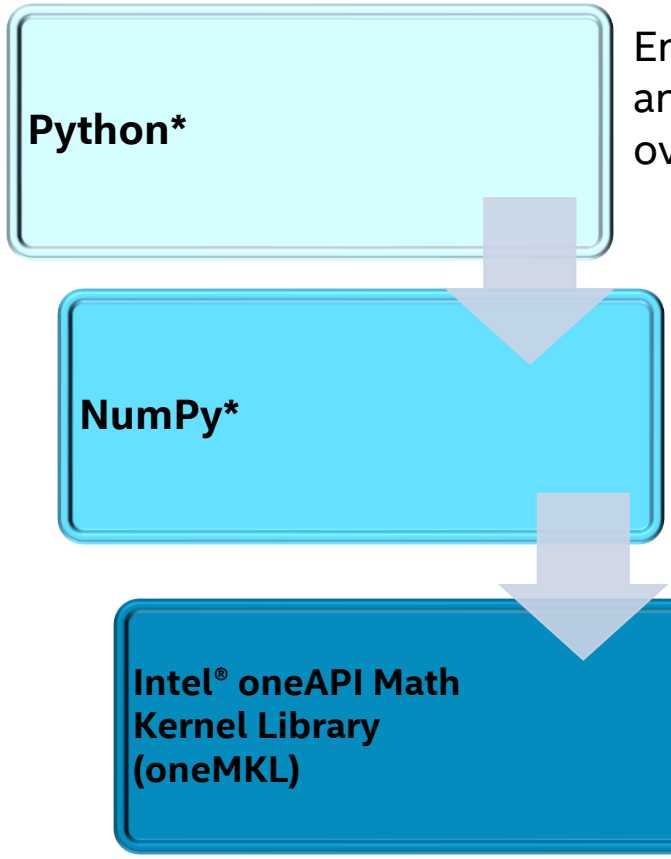
Initial GPU support enabled with Data Parallel Python



Introduction to Python Performance, cont.*

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



Enforces *Global Interpreter Lock* (GIL) and is single-threaded, abstraction overhead. No advanced types.

Gets around the GIL (multi-thread and multi-core) *BLAS API* can be the bottleneck

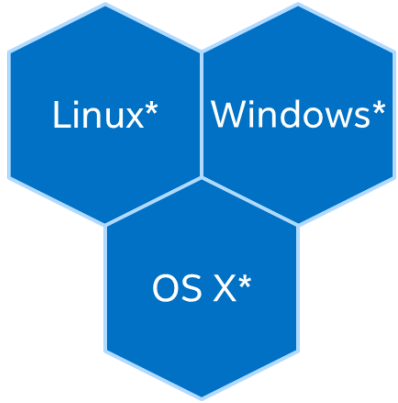
*Basic Linear Algebra Subprograms (BLAS) [CBLAS]

Gets around BLAS API bottleneck
Much stricter typing
Fastest performance level
Dispatches to hardware vectorization

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

Choose Your Download Option

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



* Also available in the Intel® oneAPI Base Toolkit

Intel Extension for Scikit Learn

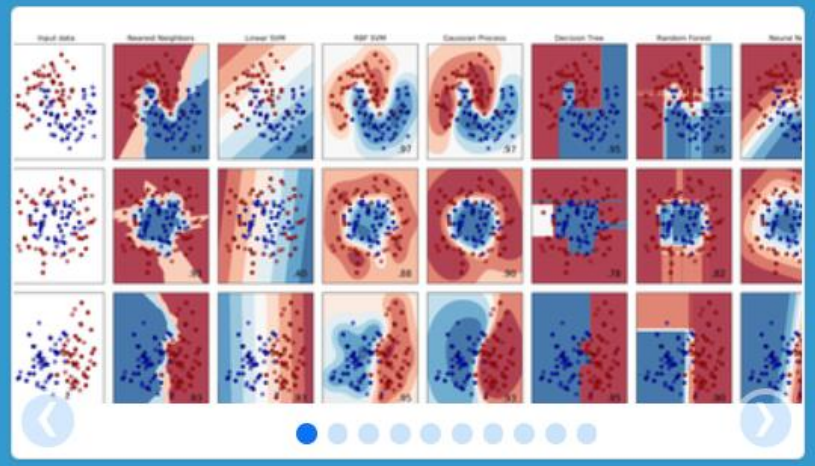
All information provided in this deck is subject to change without notice.
Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

THE MOST POPULAR ML PACKAGE FOR PYTHON*

scikit learn

Home Installation Documentation Examples

Google Custom Search Search



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, … — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, … — Examples

Clustering

Automatic grouping of similar objects into sets.

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

Intel(R) Extension for Scikit-learn

Common Scikit-learn

```
from sklearn.svm import SVC

X, Y = get_dataset()


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

Scikit-learn 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)
```

Available through PIP or conda
(pip install scikit-learn-intelex)

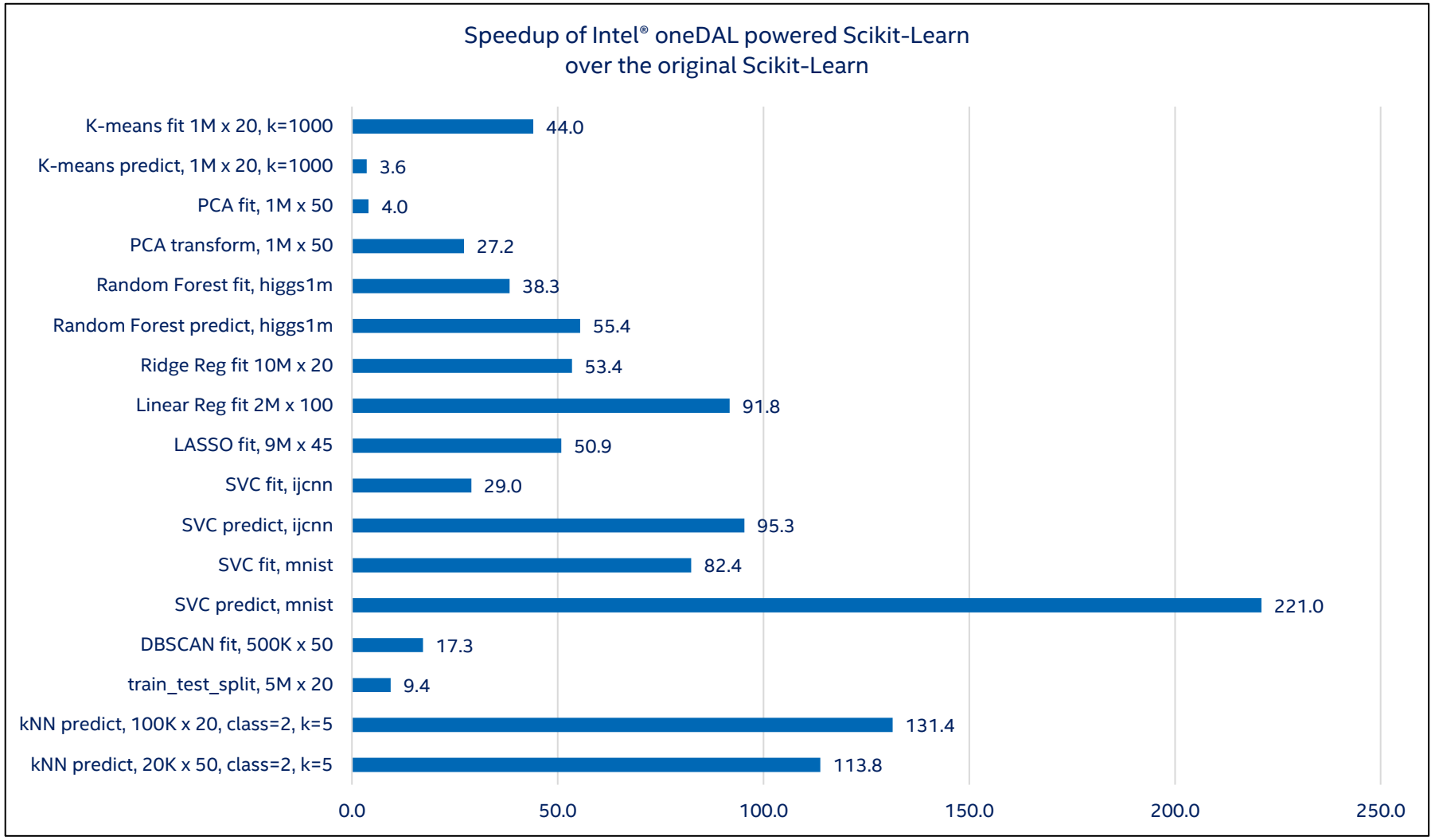
Same Code, Same Behavior

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

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
  - kNN (kd-tree and brute force)

Intel optimized Scikit-Learn



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

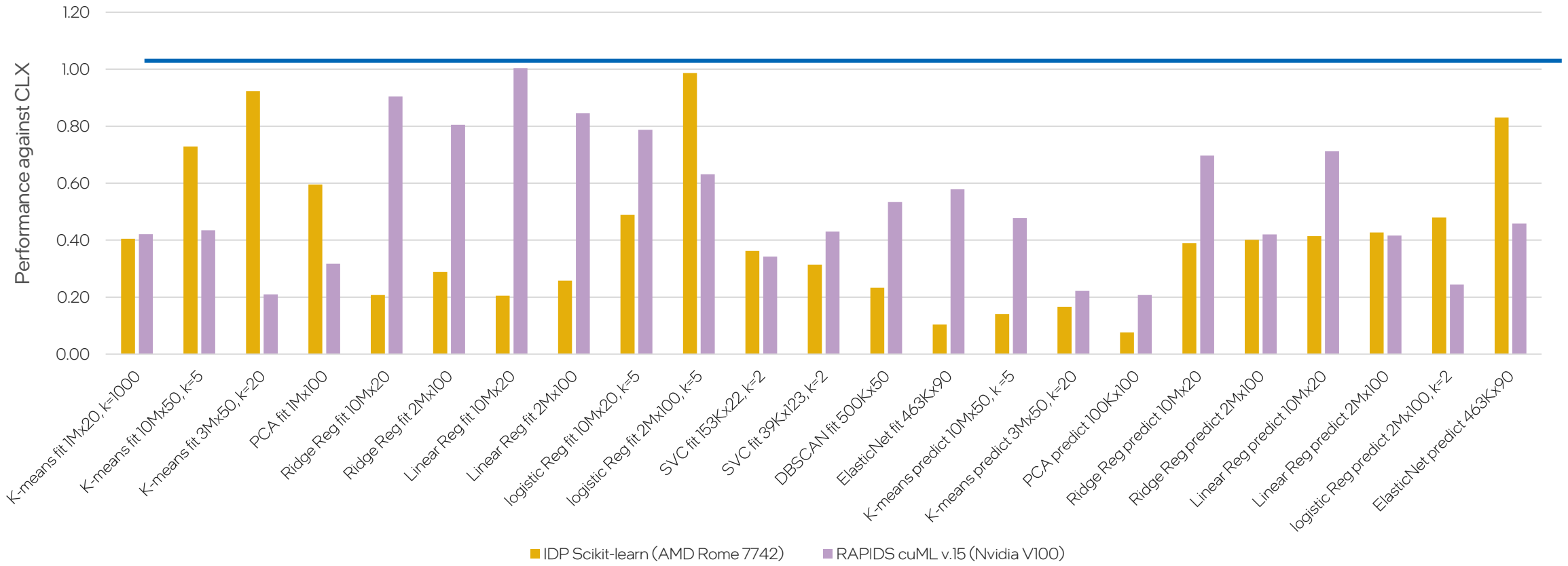
Same Code, Same Behavior

PASSED

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

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

(Intel = 1)



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

Configuration Details and Workload Setup: Intel® oneDAL beta10, Scikit-learn 0.23.1, Intel® Distribution for Python 3.7, Intel® AI Analytics Toolkit 2021.1, Intel(R) Xeon(R) Platinum 8280 CPU @ 2.70GHz, 2 sockets, 28 cores per socket, microcode: 0x4003003, total available memory 376 GB, 12X32GB modules, DDR4. AMD Configuration: AMD Rome 7742 @2.25 GHz, 2 sockets, 64 cores per socket, microcode: 0x8301038, total available memory 512 GB, 16X32GB modules, DDR4, Intel® oneDAL beta10, Scikit-learn 0.23.1, Intel® Distribution for Python 3.7. NVIDIA Configuration: Nvidia Tesla V100-16Gb, total available memory 376 GB, 12X32GB modules, DDR4, Intel(R) Xeon(R) Platinum 8280 CPU @ 2.70GHz, 2 sockets, 28 cores per socket, microcode: 0x5003003, cuDF 0.15, cuML 0.15, CUDA 10.2.89, driver 440.33.01, Operation System: CentOS Linux 7 (Core), Linux 4.19.36 kernel.

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.

Choose the Best Accelerated Technology

Intel Performance optimizations for Deep Learning

All information provided in this deck is subject to change without notice.
Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Intel® oneAPI AI Analytics Toolkit

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

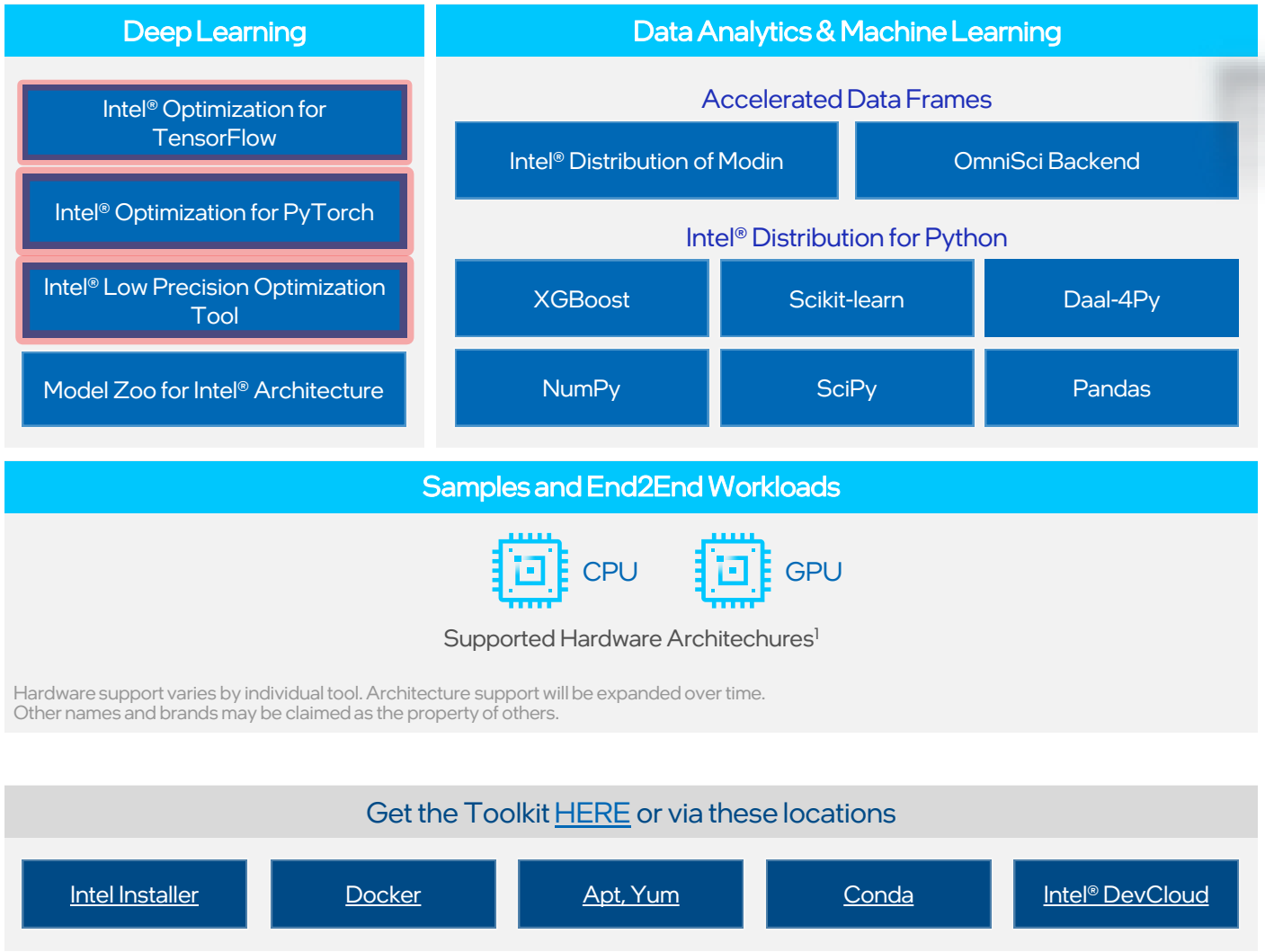
Who Uses It?

Data scientists, AI researchers, ML and DL developers, AI application developers

Top Features/Benefits

- Deep learning performance for training and inference with Intel optimized DL frameworks and tools
- Drop-in acceleration for data analytics and machine learning workflows with compute-intensive Python packages

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



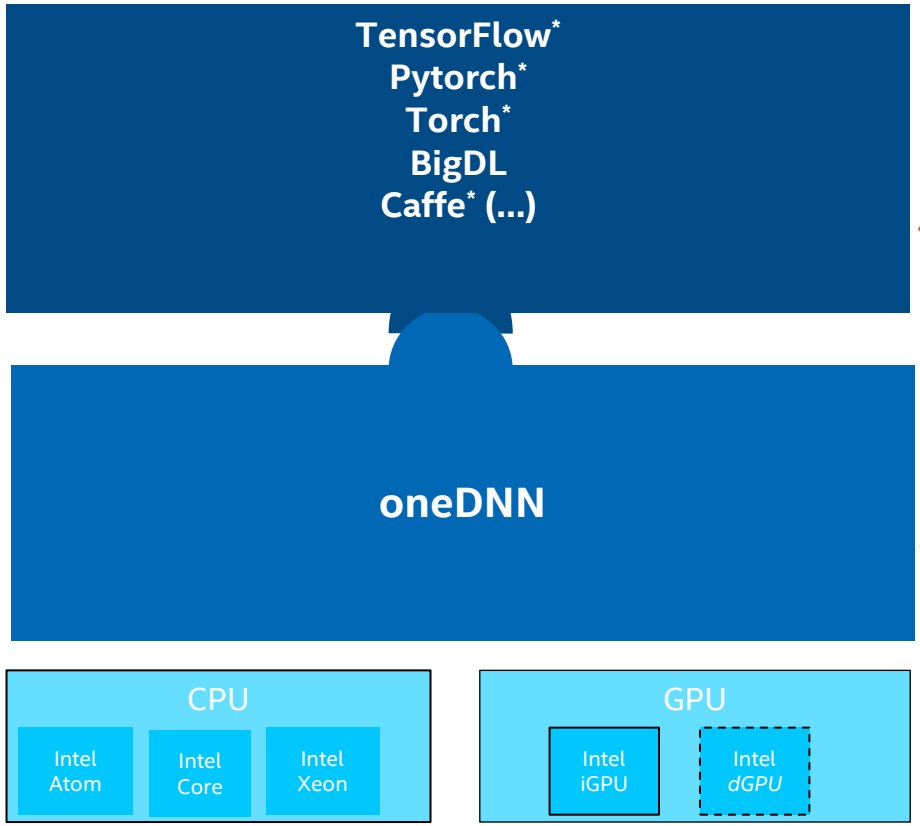
Develop Fast Neural Networks on Intel® CPUs & GPUs

with Performance-optimized Building Blocks

Intel® oneAPI Deep Neural Network Library (oneDNN)

ONEAPI DEEP NEURAL NETWORK LIBRARY (ONEDNN)

WHAT'S INSIDE



Deep learning and AI ecosystem includes edge and datacenter applications.

- **Open source frameworks** (Tensorflow*, Pytorch*, ONNX Runtime*)
- OEM applications (Matlab*, DL4J*)
- Cloud service providers internal workloads
- **Intel deep learning products** (OpenVINO™, BigDL)

oneDNN is an open source performance library for deep learning applications

- Includes **optimized** versions of **key deep learning functions**
- **Abstracts out** instruction set and other complexities of performance optimizations
- Same API for both Intel CPU's and GPU's, use the best technology for the job
- **Open** for community contributions

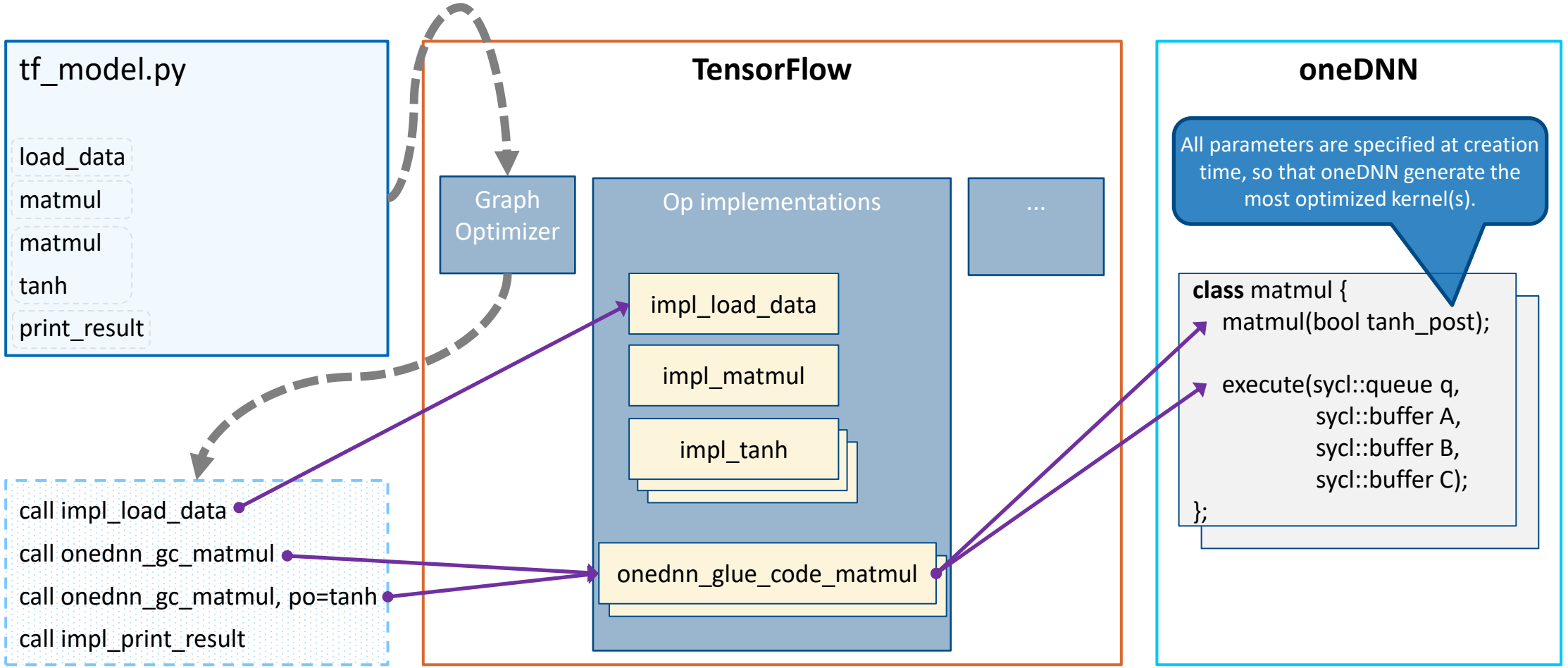
Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, 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.
Notice Revision #20110804

Intel® oneAPI Deep Neural Network Library (oneDNN)

- Cross-platform performance library of basic building blocks for DL
- Optimized for Intel® Architecture Processors, Intel® Processor Graphics and Xe architecture-based Graphics
- Supports the computation of a variety of data types including FP32, BF16 and INT8

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

oneDNN <-> Frameworks interaction



Deep Learning Framework (Optimizations by Intel)

SCALING
- Improve load balancing
- Reduce synchronization events, all-to-all comms

UTILIZE ALL THE CORES
- OpenMP, MPI
- Reduce synchronization events, serial code
- Improve load balancing

VECTORIZE / SIMD
- Unit strided access per SIMD lane
- High vector efficiency
- Data alignment

EFFICIENT MEMORY / CACHE USE
- Blocking
- Data reuse
- Prefetching
- Memory allocation

Graph optimizations
- fuse
- Batch normalization
- Memory allocation

See installation guides at ai.intel.com/framework-optimizations/

More framework optimizations underway (e.g., PaddlePaddle*, CNTK* & more)

SEE ALSO: Machine Learning Libraries for Python (Scikit-learn, Pandas, NumPy), R (Cart, randomForest, e1071), Distributed (MlLib on Spark, Mahout)

* Limited availability today

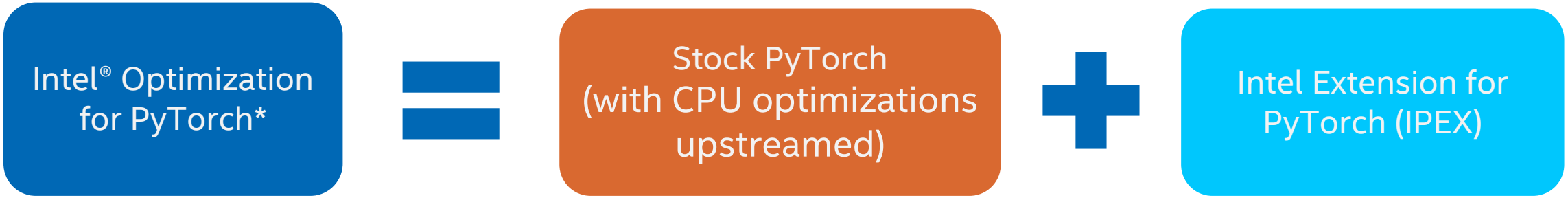
Other names and brands may be claimed as the property of others.
Optimization Notice

Intel Optimizations for Pytorch

All information provided in this deck is subject to change without notice.
Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

What is Intel Optimization for PyTorch ?

Intel regularly upstreams most of the CPU optimizations to stock PyTorch, and releases additional features and performance improvements through extensions.



Both stock and extension libraries are combined into one package called Intel Optimization for Pytorch and made available via multiple distribution channels through Intel oneAPI AI Analytics Toolkit.

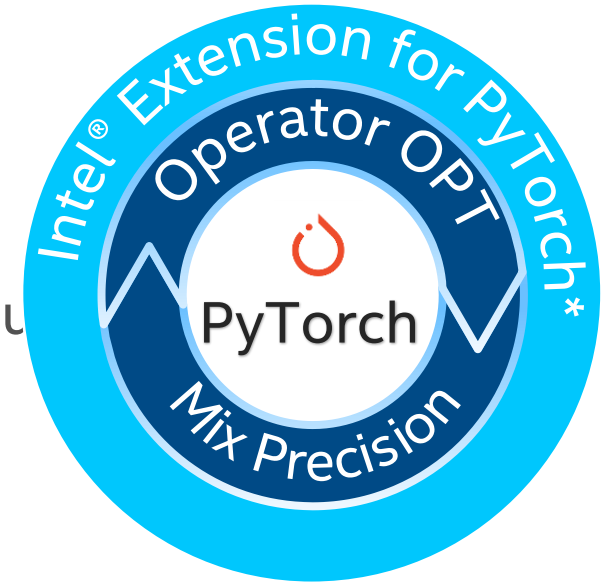
CPU Optimizations for Stock PyTorch

- CPU Optimizations are up streamed to stock Pytorch
  - Vectorize kernel by Intel® AVX2/AVX-512
  - Integrate oneDNN by default
  - Support NHWC/BF16/INT8
  - ……

```
import torch
import torch.nn as nn


class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self._conv2d = nn.Conv2d(3, 5, 5)
        self._softmax = nn.Softmax(dim=1)
    def forward(self, input):
        conv_res = self._conv2d(input)
        res = self._softmax(conv_res_dense)
        return res

input = torch.randn(5, 3, 9, 9)
model = Model()
res = model(input)
```

Intel® Extension for PyTorch (IPEX)

- Buffer the PRs for stock Pytorch
- Provide users with the up-to-date Intel software/hardware featu
- Streamline the work to integrate oneDNN
- Unify user experiences on Intel CPU and GPU



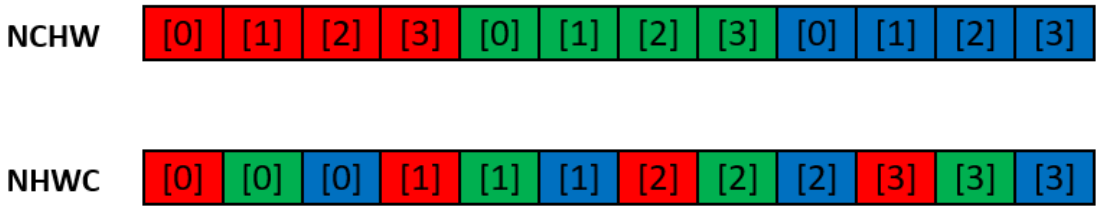
Operator Optimization
- Customized operators
- Auto graph optimization

Mix Precision
- Accelerate PyTorch operator by LP
- Simplify the data type conversion

Optimal Optimizer
- Split Optimizer (e.g., split-sgd)
- Fused Optimizer

Data Layouts in PyTorch

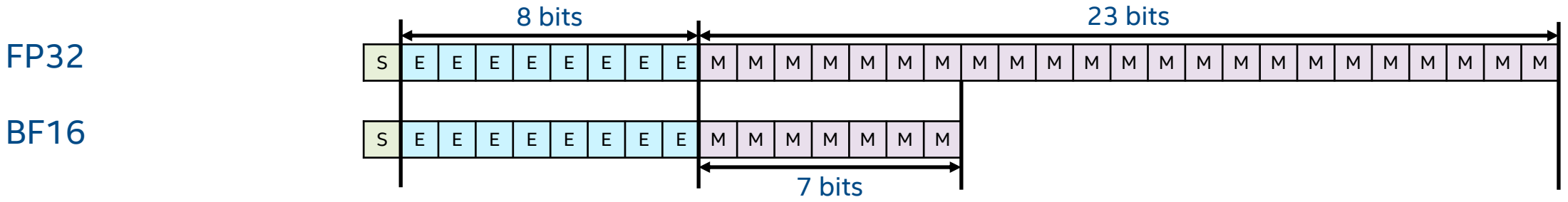
- Used in Vision workloads
- NCHW
  - Default format
  - *torch.contiguous_format*
- **NHWC**
  - A working-in-progress feature of PyTorch
  - *torch.channels_last*
  - NHWC format yields higher performance




```
## NB: internally blocked format will still be used.
##   aka. we do 'reorder' for 'input', 'weight' and 'output',
##   and believe me this is expensive, roughly 50% perf loss...
input = torch.randn(1, 10, 32, 32)
model = torch.nn.Conv2d(10, 20, 1, 1)
output = model(input)
```

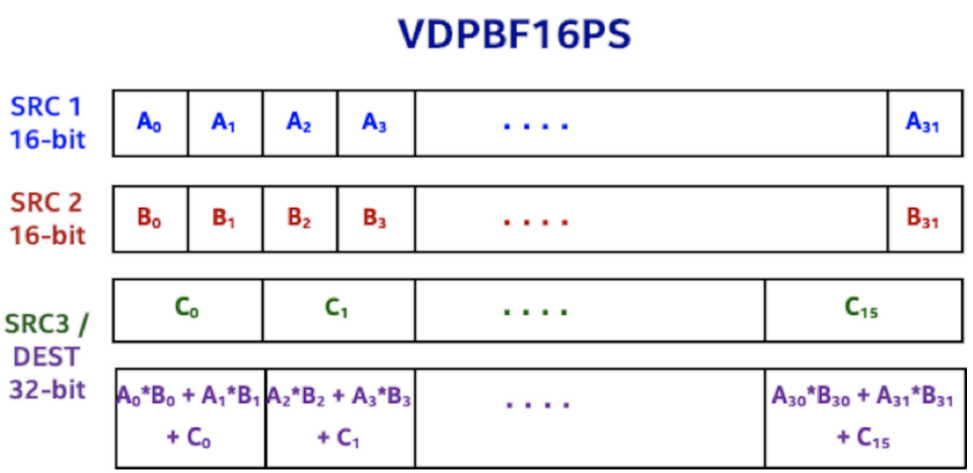
```
input = torch.randn(1, 10, 32, 32)
model = torch.nn.Conv2d(10, 20, 1, 1)
## NB: convert to Channels Last memory format.
##   oneDNN supports NHWC for feature maps (input, output),
##   but weight still needs to be of blocked format.
##   Still we can save reorders for feature maps.
input = input.to(memory_format=torch.channels_last)
model = model.to(memory_format=torch.channels_last)
output = model(input)
```

BFloat16 Data Type



BFloat16 data type specific instructions:

VCVTNE2PS2BF16	Convert Two Packed Single Data to One Packed BF16 Data
VCVTNEPS2BF16	Convert Packed Single Data to Packed BF16 Data
VDPBF16PS	Dot Product of BF16 Pairs Accumulated into Packed Single Precision

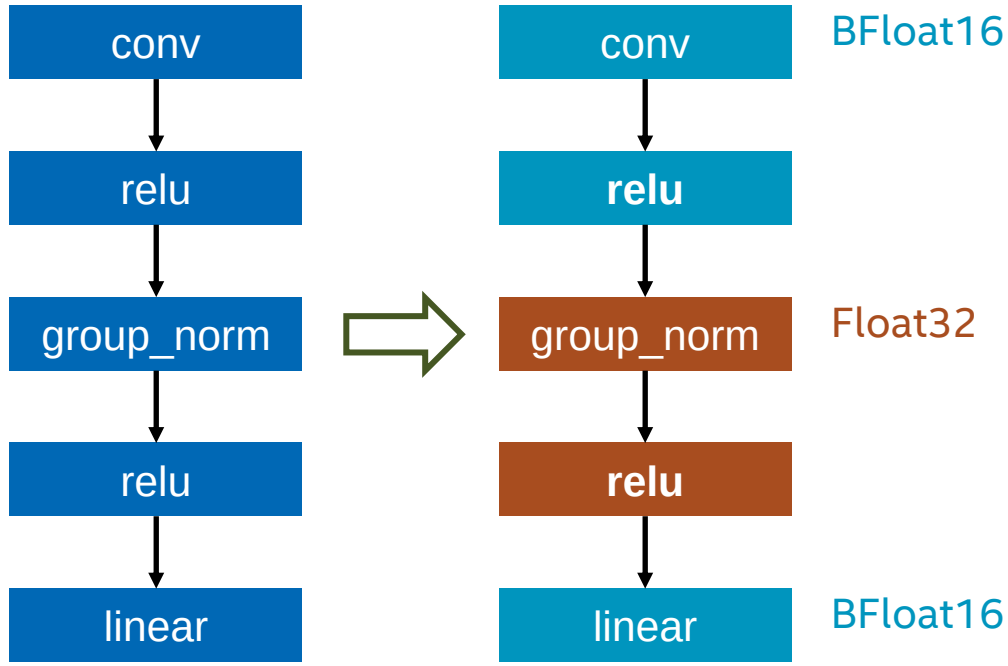


https://www.intel.com/content/dam/develop/external/us/en/documents/bf16-hardware-numerics-definition-white-paper.pdf

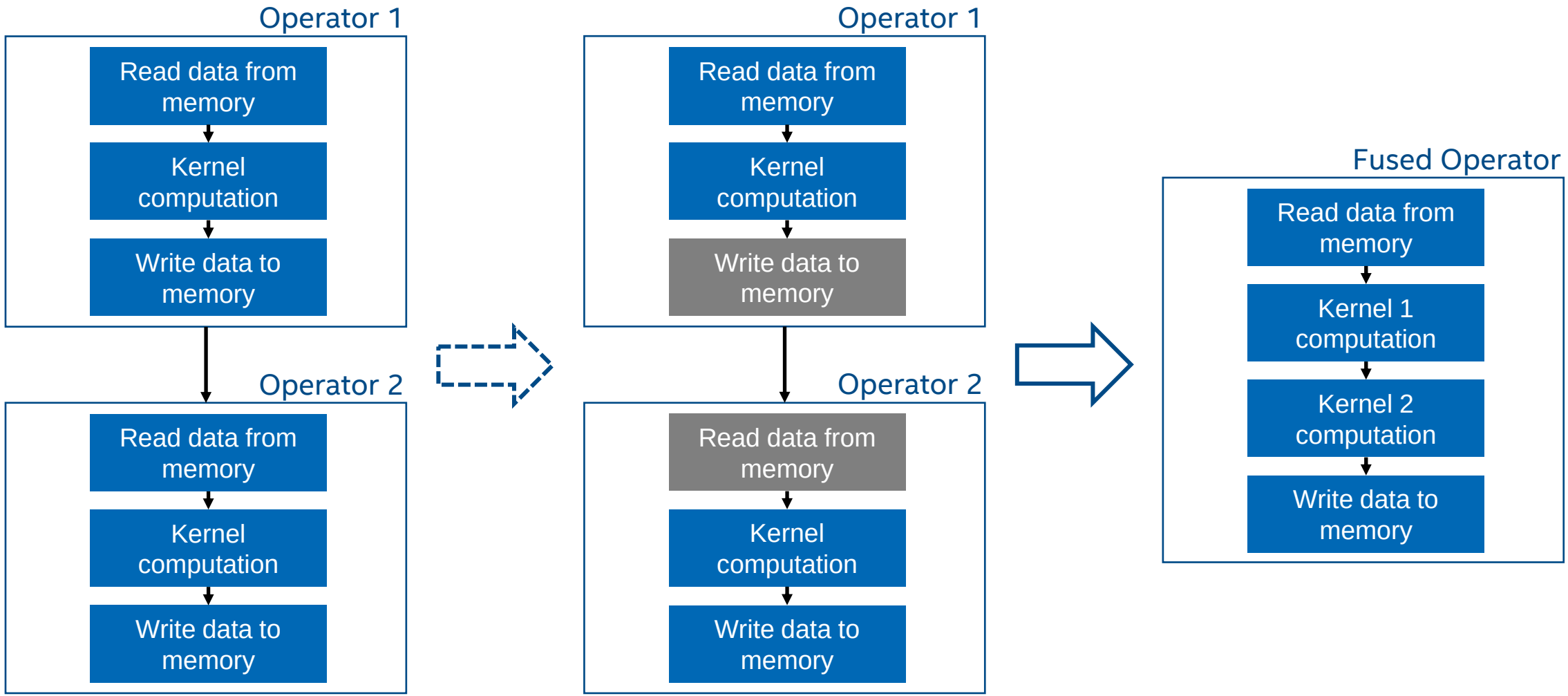
Auto Mixed Precision (AMP)

```
model = SimpleNet().eval()
x = torch.rand(64, 64, 224, 224)
with torch.cpu.amp.autocast():
    y = model(x)
```

- 3 Categories of operators
  - **lower_precision_fp**
    - Computation bound operators that could get performance boost with **BFloat16**.
    - E.g.: conv, linear
  - **Fallthrough**
    - Operators that runs with both Float32 and BFloat16 but might not get performance boost with BFloat16.
    - E.g.: relu, max_pool2d
  - **FP32**
    - Operators that are not enabled with BFloat16 support yet. Inputs of them are casted into float32 before execution.
    - E.g.: max_pool3d, group_norm



Operator Fusion



FP32 & BF16 fusion patterns

- Conv2D + ReLU
- Conv2D + SUM
- Conv2D + SUM + ReLU
- Conv2D + Sigmoid
- Conv2D + Sigmoid + MUL
- Conv2D + HardTanh
- Conv2D + SiLU
- Conv2D + ELU
- Conv3D + ReLU
- Conv3D + SUM
- Conv3D + SUM + ReLU
- Conv3D + SiLU
- Linear + ReLU
- Linear + GELU
- Add + LayerNorm
- Div + Add + Softmax
- Linear + Linear + Linear
- View + Transpose + Contiguous + View

Python – Imperative Mode

- FP32

```
import torch
import torchvision.models as models

model = models.resnet50(pretrained=True)
model.eval()
data = torch.rand(1, 3, 224, 224)

import intel_extension_for_pytorch as ipex
model = model.to(memory_format=torch.channels_last)
model = ipex.optimize(model)
data = data.to(memory_format=torch.channels_last)

with torch.no_grad():
  model(data)
```

- BFloat16

```
import torch
from transformers import BertModel

model = BertModel.from_pretrained(args.model_name)
model.eval()

vocab_size = model.config.vocab_size
batch_size = 1
seq_length = 512
data = torch.randint(vocab_size, size=[batch_size, seq_length])

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)
```

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

Python – TorchScript Mode

- FP32

```
import torch
from transformers import BertModel

model = BertModel.from_pretrained(args.model_name)
model.eval()

vocab_size = model.config.vocab_size
batch_size = 1
seq_length = 512
data = torch.randint(vocab_size, size=[batch_size, seq_length])

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

with torch.no_grad():
  d = torch.randint(vocab_size, size=[batch_size, seq_length])
  model = torch.jit.trace(model, (d,), check_trace=False, strict=False)
  model = torch.jit.freeze(model)

  model(data)
```

- BFloat16

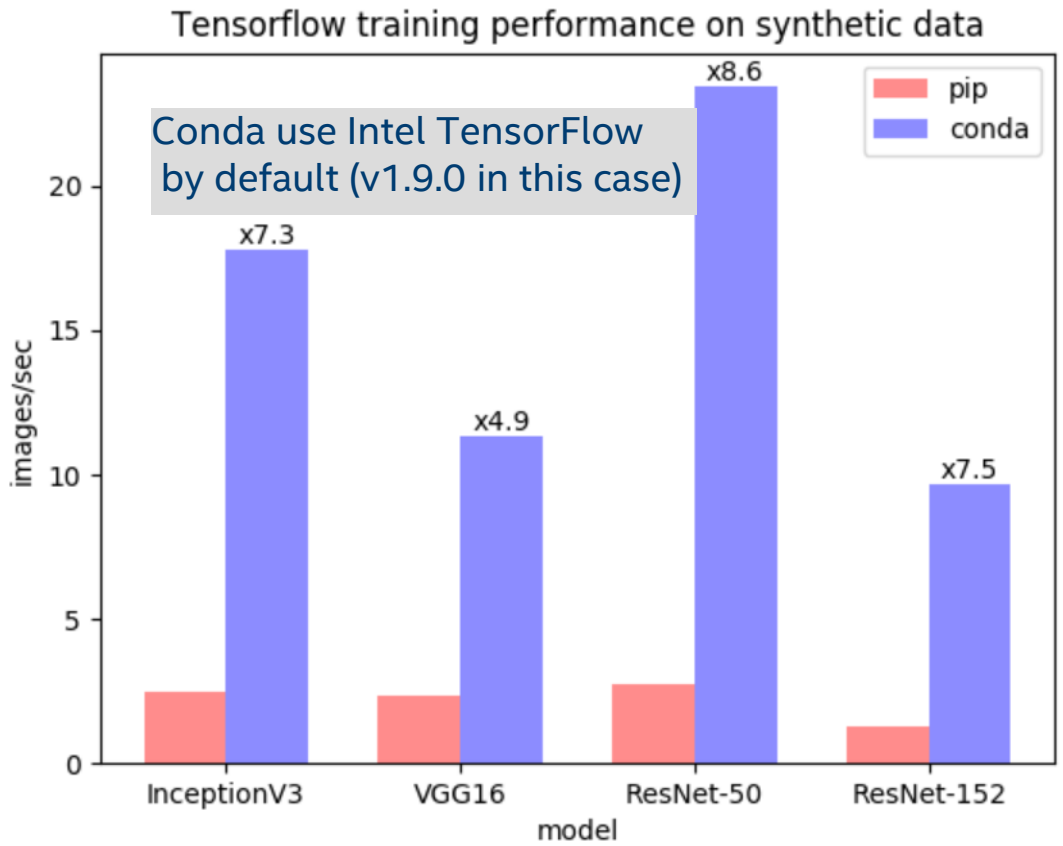
```
import torch
import torchvision.models as models

model = models.resnet50(pretrained=True)
model.eval()
data = torch.rand(1, 3, 224, 224)

import intel_extension_for_pytorch as ipex
model = model.to(memory_format=torch.channels_last)
model = ipex.optimize(model, dtype=torch.bfloat16)
data = data.to(memory_format=torch.channels_last)

with torch.no_grad():
  with torch.cpu.amp.autocast():
    model = torch.jit.trace(model, torch.rand(1, 3, 224, 224))
    model = torch.jit.freeze(model)

    model(data)
```

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

Intel Optimizations for Tensorflow

All information provided in this deck is subject to change without notice.
Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

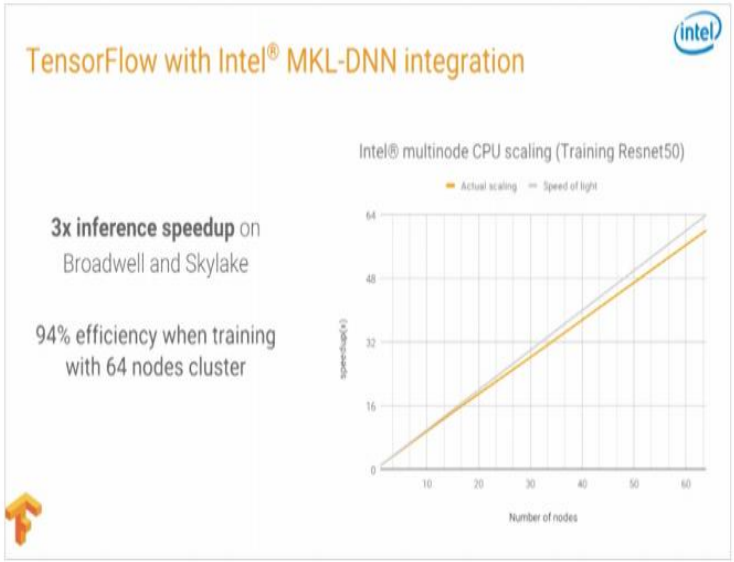
Examples of speedups on Intel® Xeon® Scalable Processors



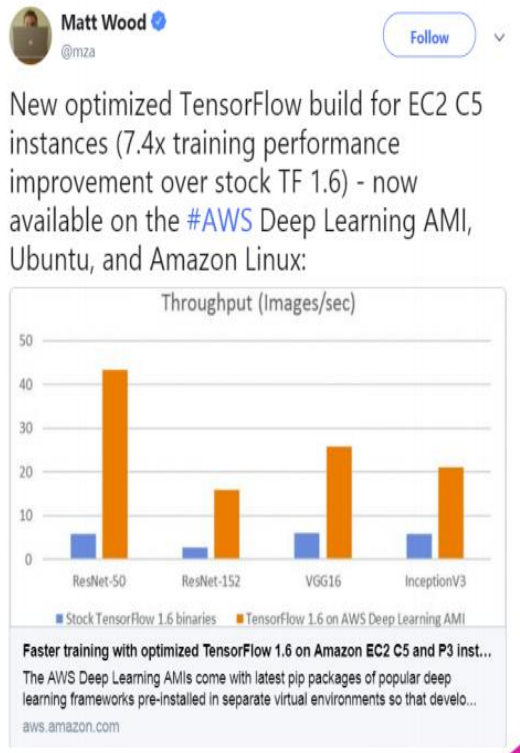
https://www.anaconda.com/blog/tensorflow-in-anaconda

PERFORMANCE GAINS REPORTED BY OTHERS

Intel TensorFlow Scalability Results Presented by Google @TF Summit March 30, '18



"By making use of [Intel's] open source library [MKL-DNN], we were able to achieve a 3x performance benefit and great scaling efficiency on training. This is an example of how important it is to have strong collaborations with companies like Intel."



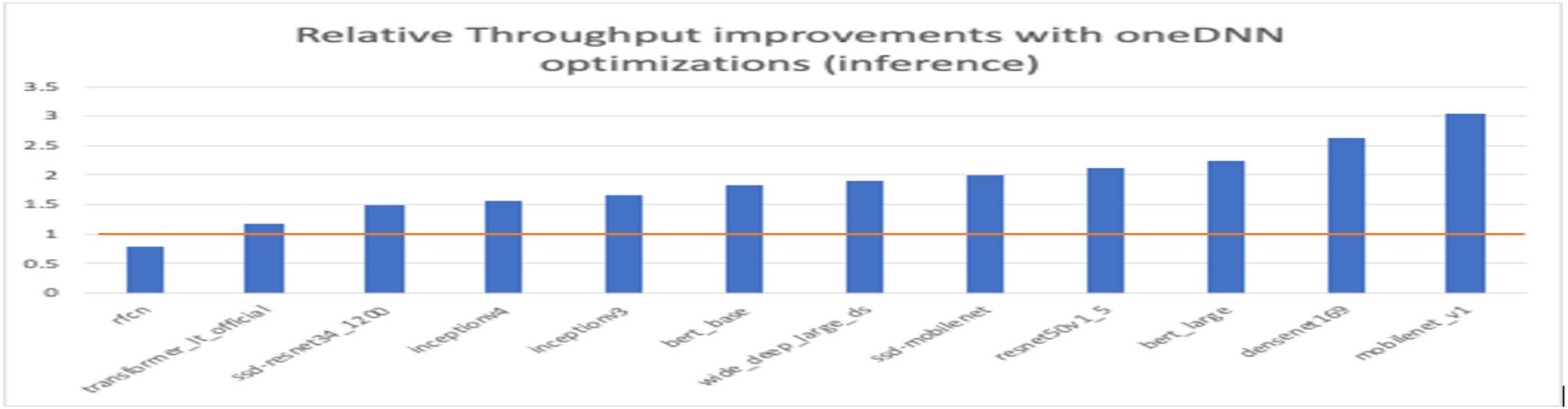
*Other names and brands may be claimed as the property of others

What's New for TensorFlow Optimization
- oneDNN is in official TensorFlow release!

- https://github.com/tensorflow/

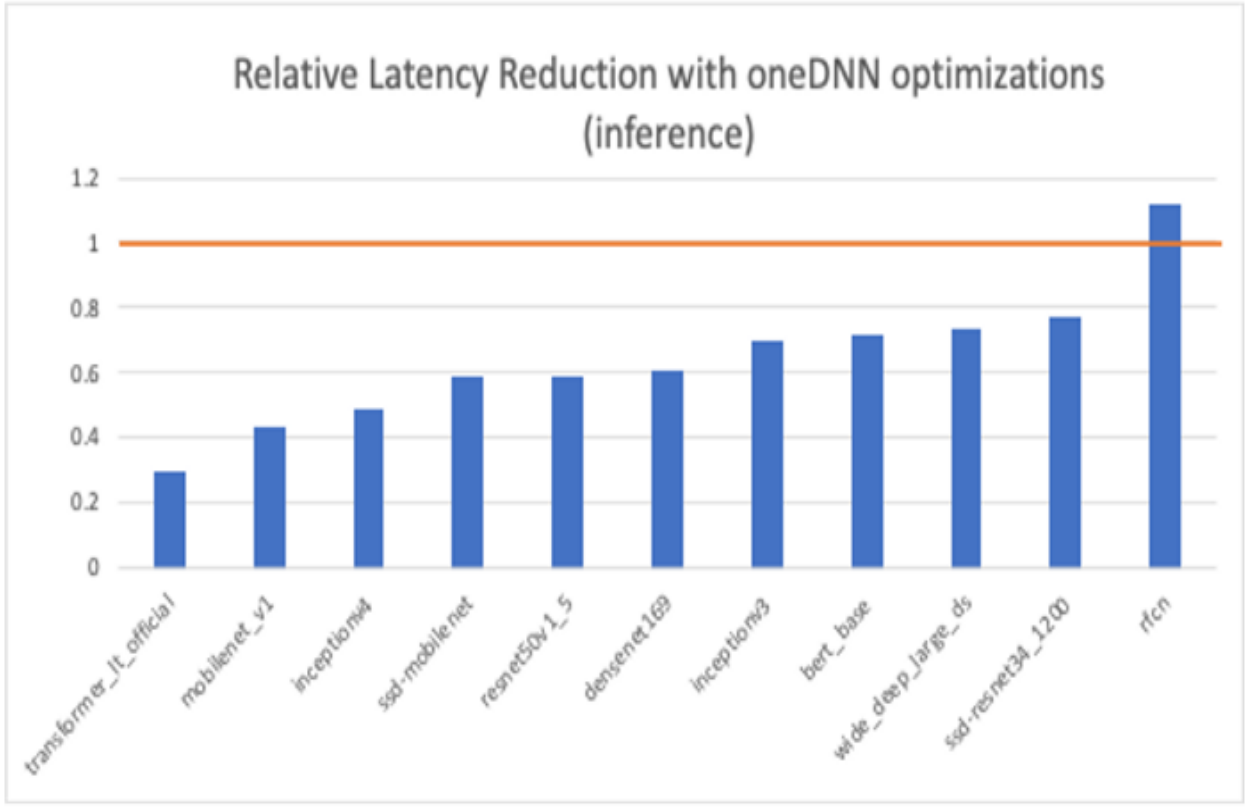
  - Register custom graph optimization passes with Graph Optimization C API.

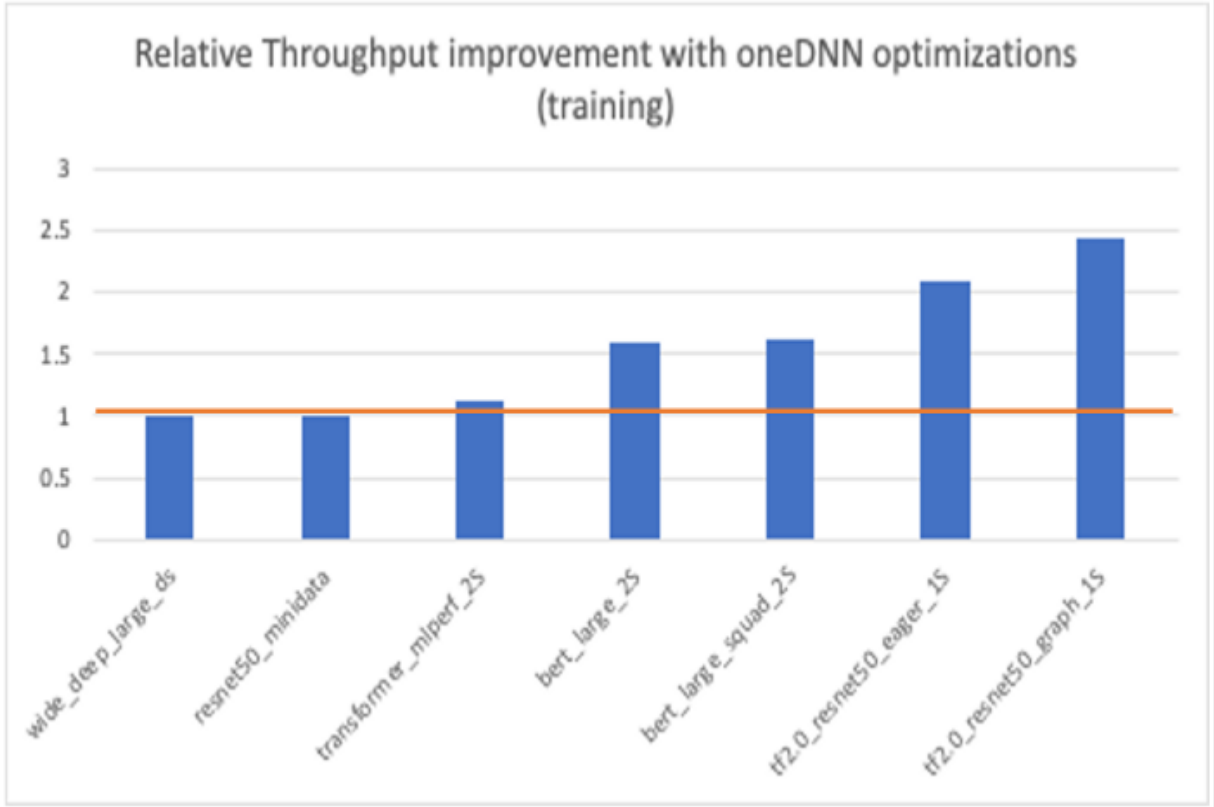
- oneAPI Deep Neural Network Library (oneDNN) CPU performance optimizations from Intel-optimized TensorFlow are now available in the official x86-64 Linux and Windows builds.
  - They are off by default. Enable them by setting the environment variable `TF_ENABLE_ONEDNN_OPTS=1`.
  - We do not recommend using them in GPU systems, as they have not been sufficiently tested with GPUs yet.



What's New for TensorFlow Optimization

- more performance number from official TF release





INFERENCE THROUGHPUT PERFORMANCE

CPU OPTIMIZED TENSORFLOW COMPARED WITH UNOPTIMIZED (STOCK) TENSORFLOW

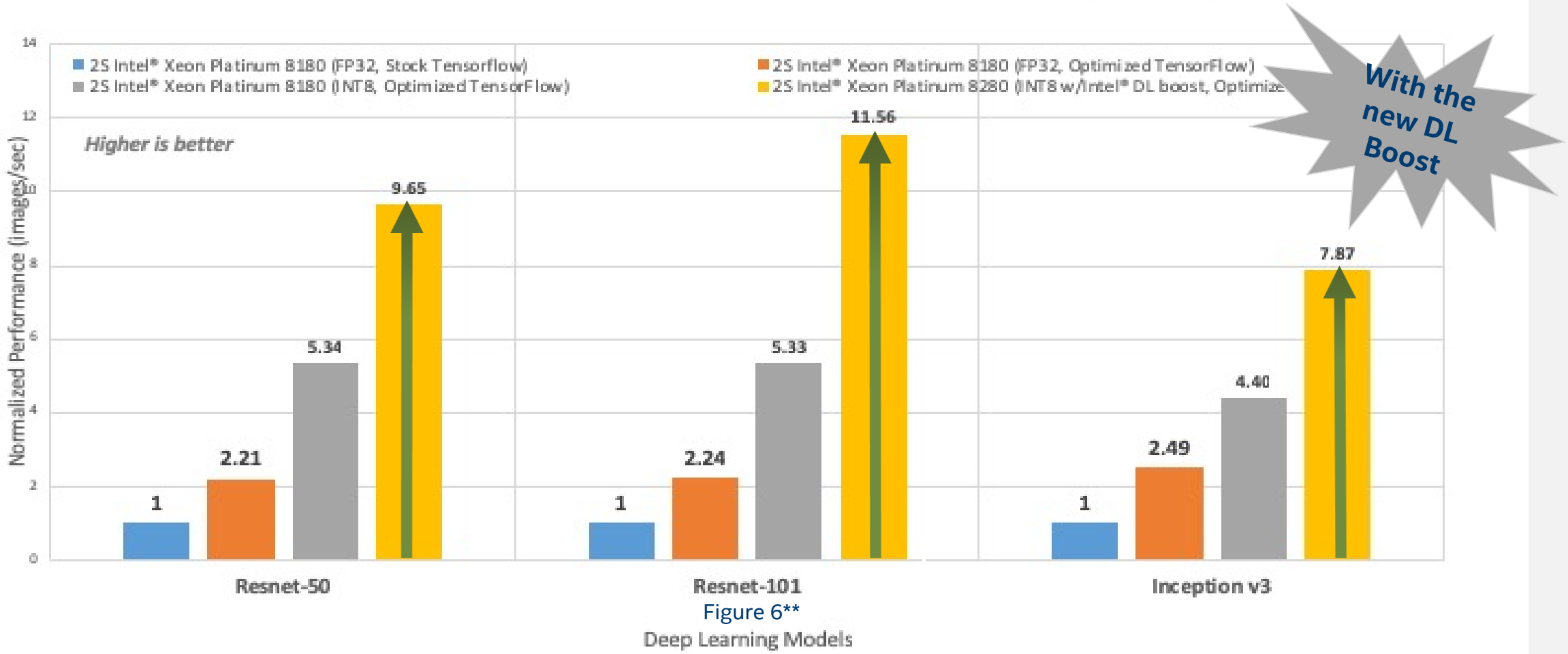


Figure 6**

Deep Learning Models

For More Details: https://www.anaconda.com/tensorflow-cpu-optimizations-in-anaconda/

What's New for TensorFlow Optimization

- comparison between official and intel TF release

https://software.intel.com/content/www/us/en/develop/articles/intel-optimization-for-tensorflow-installation-guide.html

	Intel Optimization for Tensorflow	official TensorFlow (Running on Intel CPUs)
oneDNN optimiziations	Enabled by default	Enable by setting environment variable TF_ENABLE_ONEDNN_OPTS=1 at runtime
OpenMP Optimizations	Enabled by default	N/A. use eigen thread pool instead.
Native Layout Format	Enabled by default. could disable the feature by setting the env-variable TF_ENABLE_MKL_NATIVE_FORMAT=0	Enabled by default.
int8 support from oneDNN	Only work when setting the env-variable TF_ENABLE_MKL_NATIVE_FORMAT=0	Coming soon.

Anaconda

- **Linux:** Main Channel **(v2.5)** | Intel Channel **(v2.5)** | Intel AI Analytics Toolkit **(v2.5)**
- **Windows**: Main Channel **(v2.3)**
- **MacOS:** Main Channel **(v2.0)**

PIP Wheels

- **Linux:** Py36 | Py37 | Py38 | Py39 **(v2.5)**

Docker Containers

- **Linux:** Intel containers **(v2.5)** | Google DL containers **(v2.5)**

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