AI on Intel Architecture

Dr. Séverine Habert, AI Engineering Manager

intel®

Agenda

- oneAPI AI Analytics Toolkit
  - Intel Distribution for Python
  - Classic Machine Learning Libraries
  - Deep Learning Frameworks
  - Intel Neural Compressor
- BigDL
- OpenVINO
- SigOpt

AI Software Ecosystem and Intel Tools

Engineer Data → Create Machine Learning & Deep Learning Models → Deploy

Container Repository	MLOps	Developer Sandbox
oneContainer	Cnvrg.io	DevCloud

Accelerate End to End Data Science and AI — AI Analytics Toolkit

Connect AI to Big Data — AI Domain Tools: RecSys, Time Series, PPML — BigDL (Analytics Zoo)

Data Analytics at Scale

MODIN, NumPy, pandas, SciPy, Apache Spark, omni·sci

Optimized Frameworks and Middleware

TensorFlow, PyTorch, mxnet, ONNX, PaddlePaddle, scikit-learn, dmlc XGBoost

Optimize and Deploy Models

Automate Model Tuning AutoML	Write Once Deploy Anywhere	Automate Low-Precision Optimization
SigOpt	OpenVINO Toolkit	Neural Compressor

oneAPI: oneDAL, oneDNN, oneCCL, oneMKL

oneAPI AI Analytics toolkit

Intel's oneAPI Ecosystem

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

oneAPI

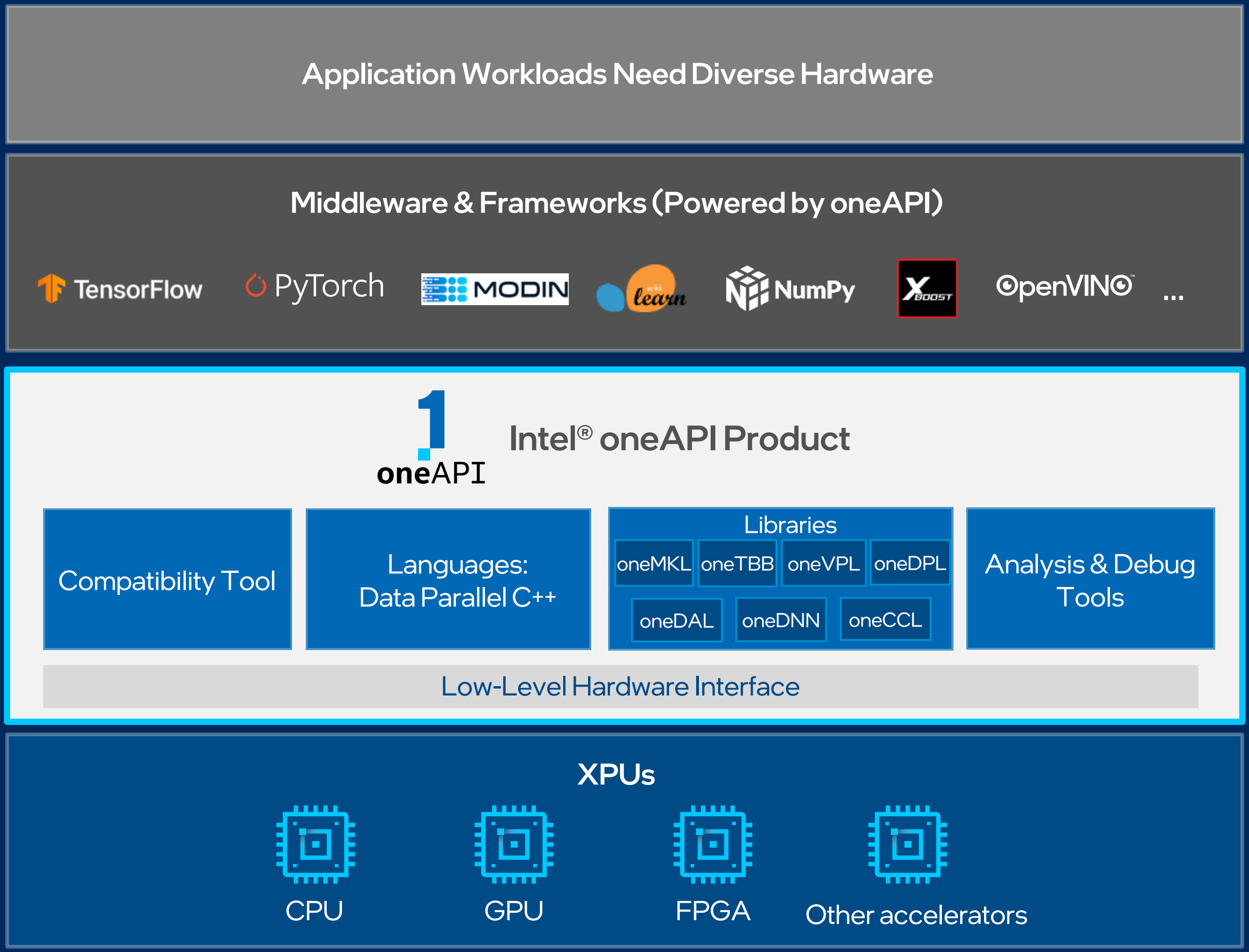
A cross-architecture language based on C++ and SYCL standards

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.

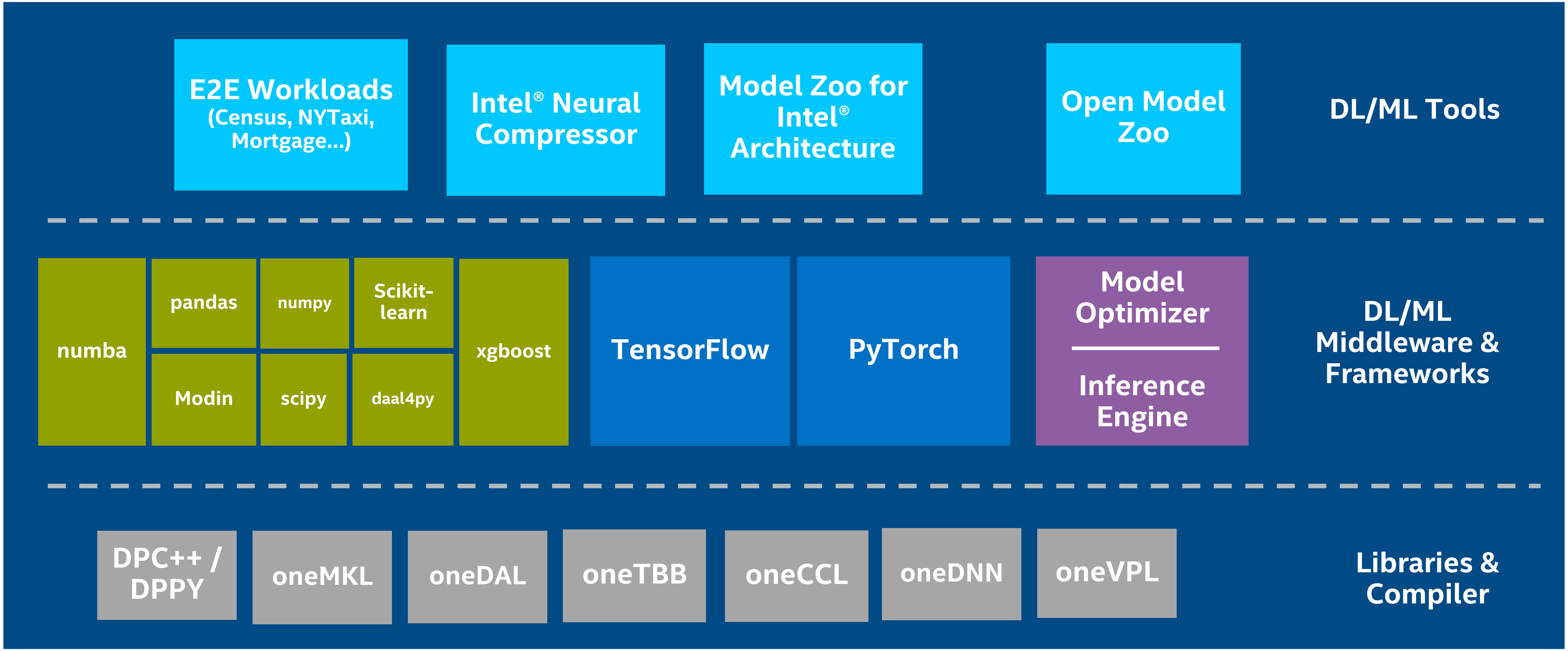


[Available Now](#)

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.

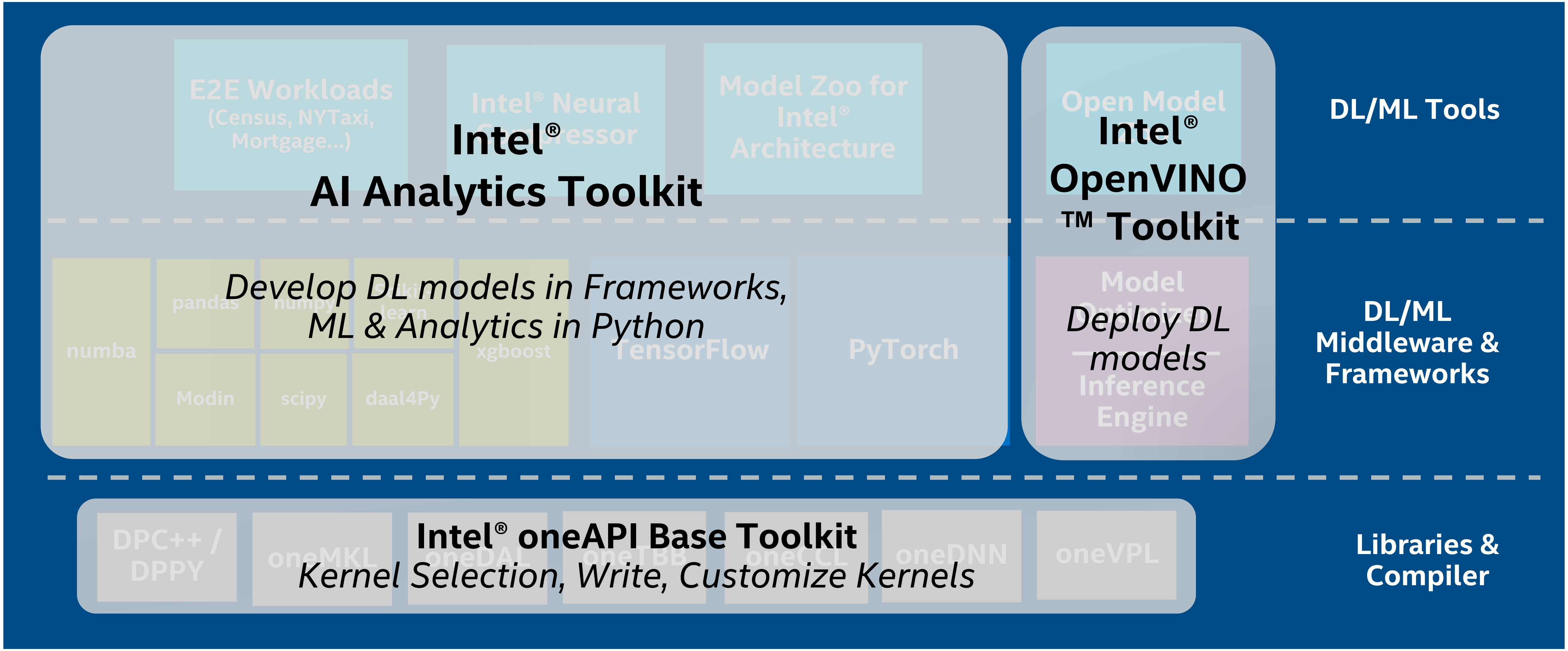
AI Software Stack for Intel® XPUs

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.

Intel® AI Analytics Toolkit

Powered by oneAPI

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

Who Uses It?

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

Top Features/Benefits

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

Deep Learning

- Intel® Optimization for TensorFlow
- Intel® Optimization for PyTorch
- Intel® Neural Compressor
- Model Zoo for Intel® Architecture

Data Analytics & Machine Learning

Accelerated Data Frames

- Intel® Distribution of Modin
- OmniSci Backend

Intel® Distribution for Python

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

Samples and End2End Workloads

CPU GPU

Supported Hardware Architectures[1]

Hardware support varies by individual tool. Architecture support will be expanded over time.
Other names and brands may be claimed as the property of others.

Get the Toolkit HERE or via these locations

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

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

Back to Domain-specific Toolkits for Specialized Workloads

Classical Machine Learning

Intel® Distribution for Python

Intel® Distribution for Python

- Intel® Distribution for Python covers major usages in HPC and Data Science

- Achieve faster Python application performance — right out of the box — with minimal or no changes to a code

- Accelerate NumPy*, SciPy*, and scikit-learn* with integrated Intel® Performance Libraries such as Intel® oneMKL (Math Kernel Library) and Intel® oneDAL (Data Analytics Library)

- By default, already integrated in Anaconda

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

Linux*
Windows*
OS X*

* Also available in the Intel® oneAPI Base Toolkit

Intel Extension for Scikit-learn

THE MOST POPULAR ML PACKAGE FOR PYTHON*

scikit learn | Install | User Guide | API | Examples | More | Go

scikit-learn

Machine Learning in Python

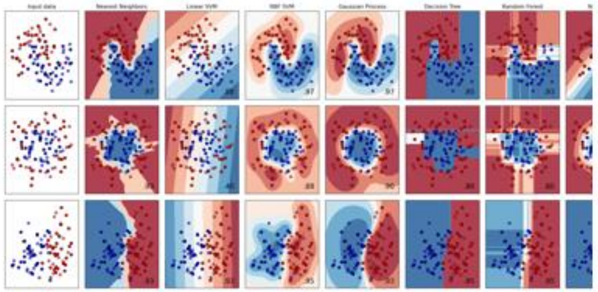
Getting Started | Release Highlights for 0.24 | GitHub

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

Classification

Identifying which category an object belongs to.

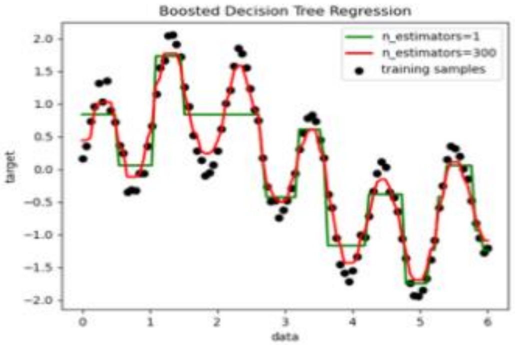
Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...



Regression

Predicting a continuous-valued attribute associated with an object.

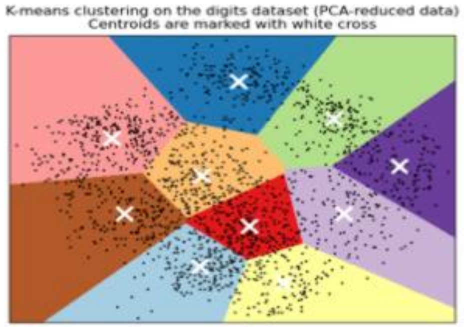
Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...



Clustering

Automatic grouping of similar objects into sets.

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



Intel(R) Extension for Scikit-learn

Common Scikit-learn

```
▪ from sklearn.svm import SVC
▪
  X, Y = get_dataset()

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

Scikit-learn mainline

Scikit-learn with Intel CPU opts

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

X, Y = get_dataset()


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

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

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

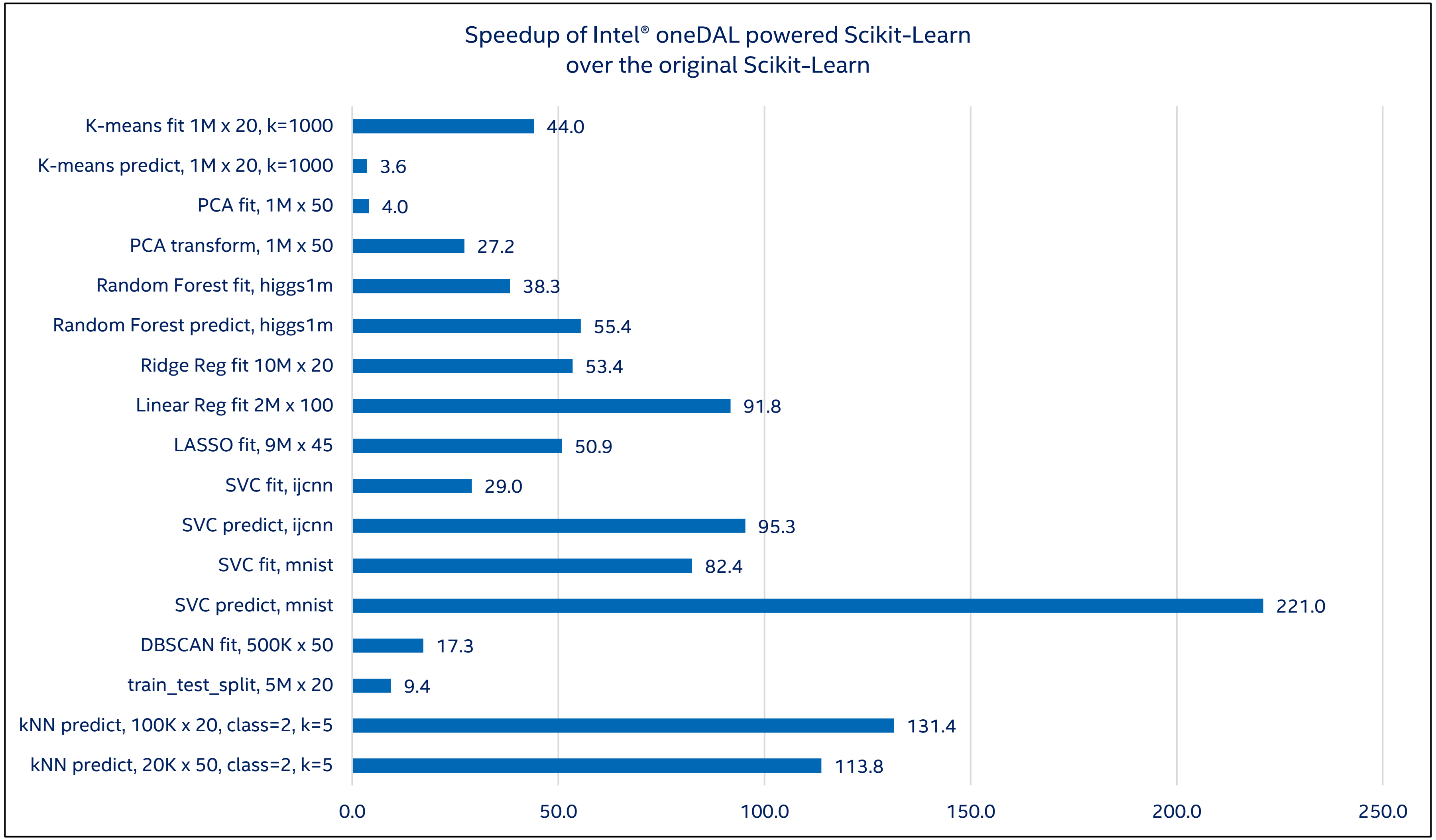
Monkey-patch any scikit-learn*
on the command-line

Same Code,
Same Behavior

PASSED

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

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

Modin

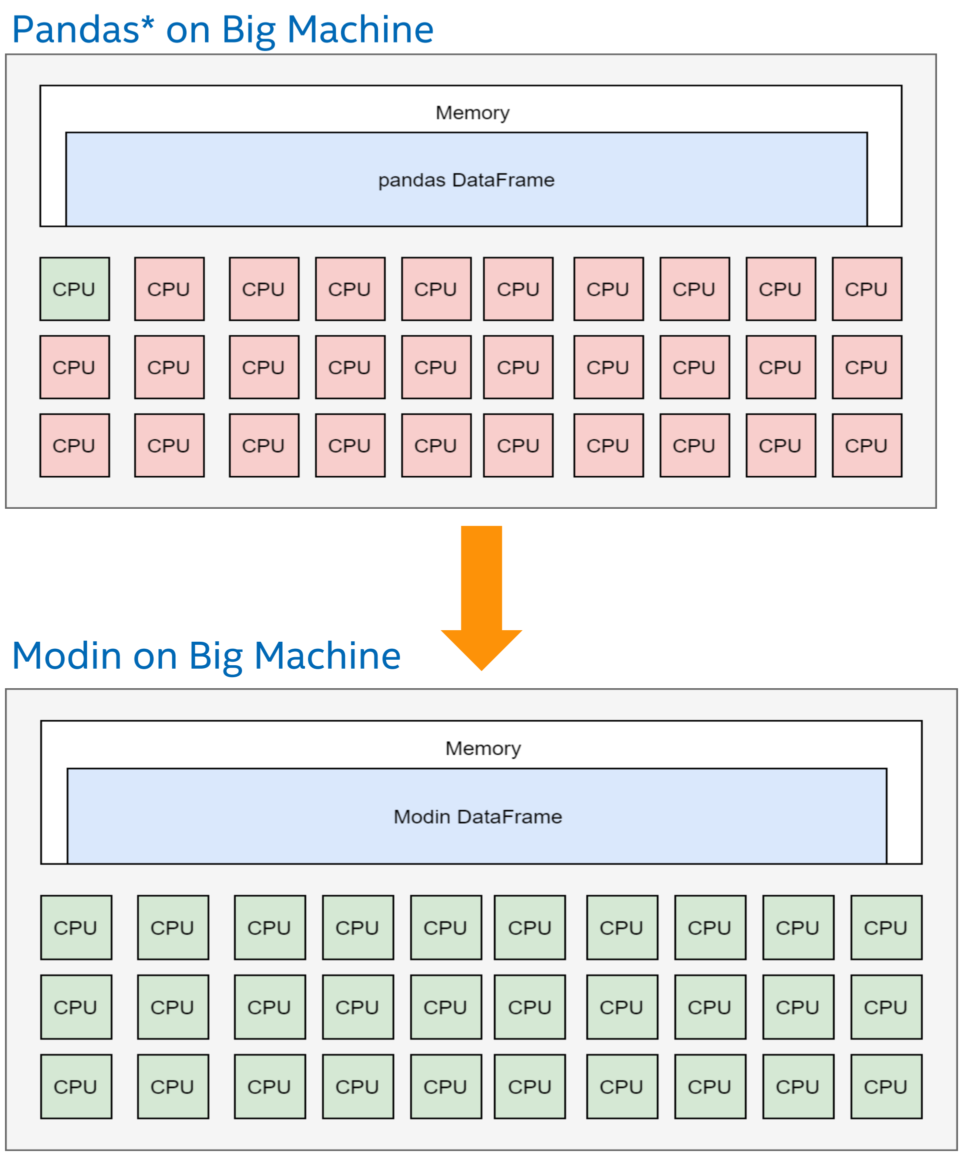
Intel distribution of Modin

- Pandas is a Python package for data manipulation and analysis that offers data structures and operations for manipulating numerical tables and time series

- **Modin =** Pandas + Scalability
- As simple as **import modin.pandas as pd**

```
import pandas as pd
```

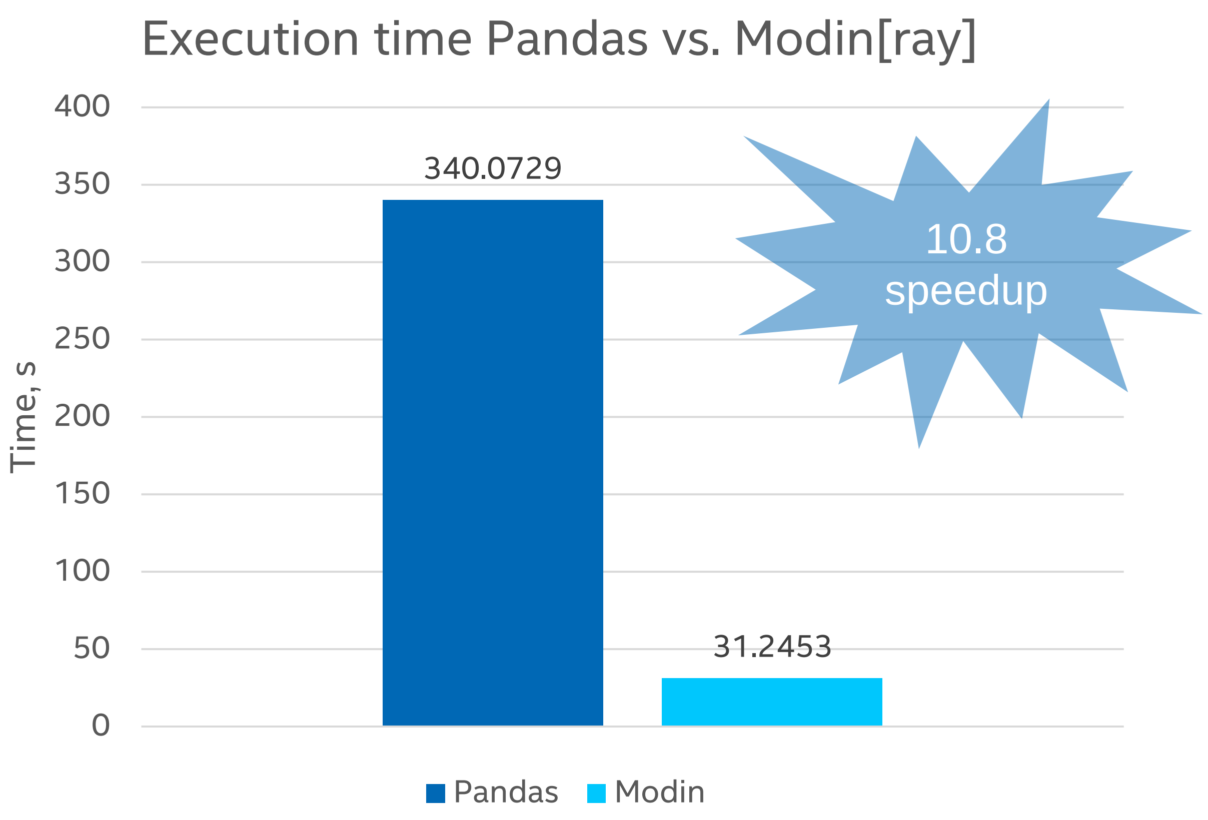
- In opposition to Pandas, Modin will use all available cores on CPU
- No need to know how many cores your system has, and no need to specify how to distribute the data
- You can get speed-up even on a laptop
- As of 0.9 version, Modin supports 100% of Pandas API



Modin

```python
import modin.pandas as pd
import numpy as np


def run_etl():

    def cat_converter(x):
        if x is '':
            return np.int32(0)
        else:
            return np.int32(int(x, 16))

    names = [f"column_{i}" for i in range(40)]
    converter= {names[i]: cat_converter for i in range(14, 40)}

    df = pd.read_csv('data.csv', delimiter='\t', names=names,
                     converters=converter)

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

    return df, count_y


df, count_y = run_etl()
```

- Dataset size: 2.4GB



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

XGBoost

Gradient Boosting - Overview

Gradient Boosting:

- Boosting algorithm (Decision Trees - base learners)
- Solve many types of ML problems (classification, regression, learning to rank)
- Highly-accurate, widely used by Data Scientists
- Compute intensive workload
- Known implementations: XGBoost*, LightGBM*, CatBoost*, Intel® oneDAL, …

Gradient Boosting Acceleration – gain sources

Pseudocode for XGBoost* (0.81) implementation

```
def ComputeHist(node):
  hist = []
  for i in samples:
    for f in features:
      bin = bin_matrix[i][f]
      hist[bin].g += g[i]
      hist[bin].h += h[i]
  return hist

def BuildLvl:
  for node in nodes:
    ComputeHist(node)

  for node in nodes:
    for f in features:
      FindBestSplit(node, f)

  for node in nodes:
    SamplePartition(node)
```

Memory prefetching to mitigate

irregular memory access

Usage uint8 instead of uint32

SIMD instructions instead of scalar code

Nested parallelism

Advanced parallelism, reducing seq loops

Usage of AVX-512, vcompress instruction (from Skylake)

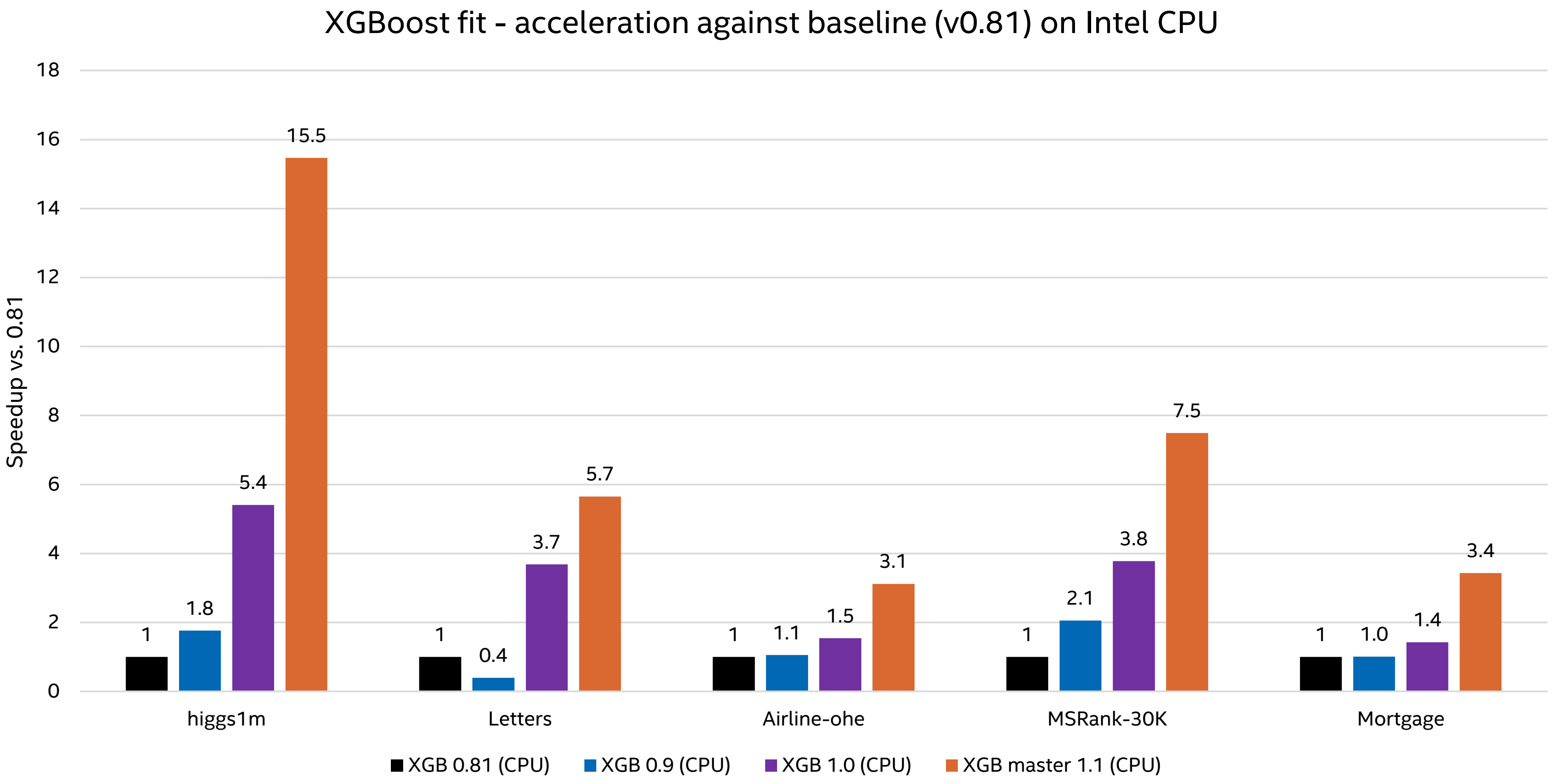
Pseudocode for Intel® oneDAL implementation

```
def ComputeHist(node):
  hist = []
  for i in samples:
    prefetch(bin_matrix[i + 10])
    for f in features:
      bin = bin_matrix[i][f]
      bin_value = load(hist[2*bin])
      bin_value = add(bin_value, gh[i])
      store(hist[2*bin], bin_value)
  return hist


def BuildLvl:
  parallel_for node in nodes:
    ComputeHist(node)

  parallel_for node in nodes:
    for f in features:
      FindBestSplit(node, f)

  parallel_for node in nodes:
    SamplePartition(node)
```

Training stage

Legend: Moved from Intel® oneDAL to XGBoost (v1.3) | Already available in Intel® DAAL, potential optimizations for XGBoost*

XGBoost* fit CPU acceleration ("hist" method)



+ Reducing memory consumption

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

CPU configuration: c5.24xlarge AWS Instance, CLX 8275 @ 3.0GHz, 2 sockets, 24 cores per socket, HT:on, DRAM (12 slots / 32GB / 2933 MHz)

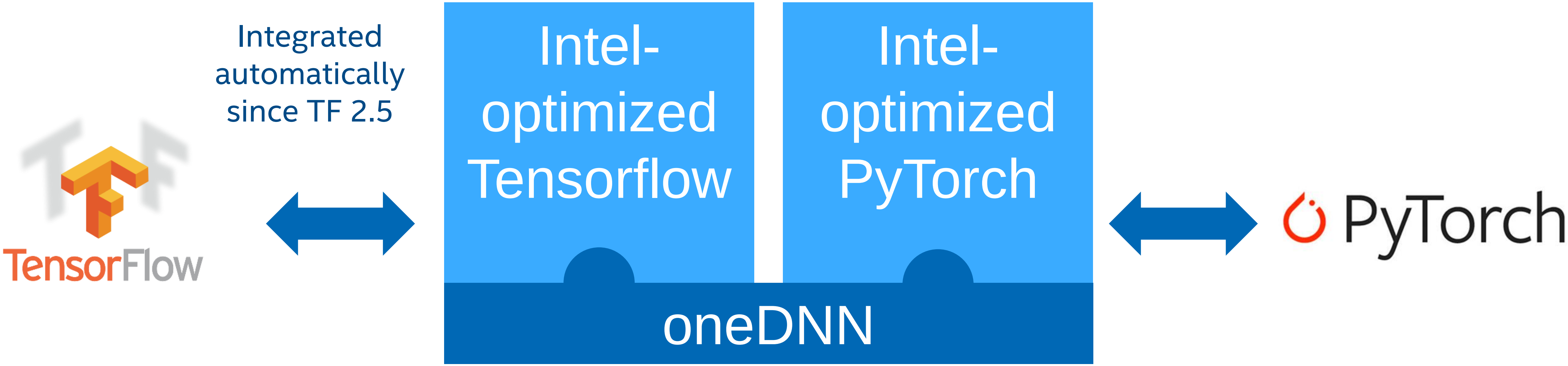
Deep Learning

intel®

Intel-optimized Deep Learning frameworks

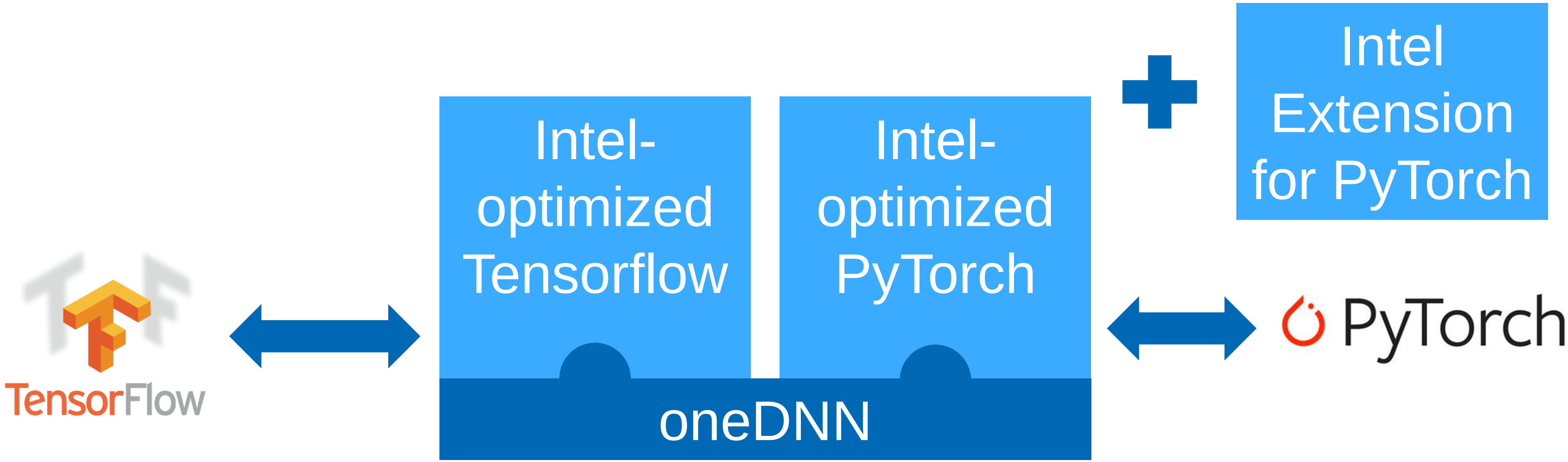
Intel-optimized Deep Learning Frameworks

- Intel-optimized DL frameworks are drop-in replacement,
  - No front code change for the user
- Optimizations are upstreamed automatically (TF) or on a regular basis (PyTorch) to stock frameworks
  - TF: Optimizations are integrated automatically since TF 2.5 and are activated after setting up TF_ENABLE_ONEDNN_OPTS=1



Intel-optimized Deep Learning Frameworks

- Intel Extension for PyTorch is an additional module for functions not supported in standard PyTorch (such as mixed precision and dGPU support)
- As they offer more aggressive optimizations, they offer bigger speed-up for training and inference



Intel® oneAPI Deep Neural Network Library (oneDNN)

Basic Information

- Features
  - **Training**: float32, bfloat16[(1)]
  - **Inference**: float32, bfloat16[(1)], float16[(1)], and int8[(1)]
  - Runs on Intel CPU and GPU

	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

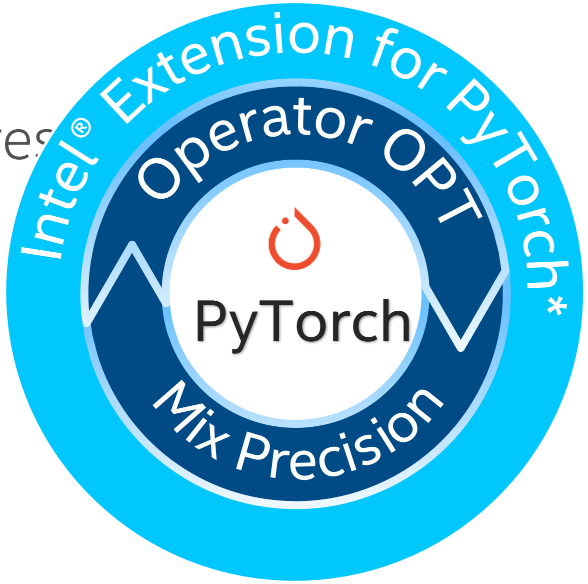
(1) Low precision data types are supported only for platforms where hardware acceleration is available

Optimizations

1. Operator optimizations: Replace default kernels by highly-optimized kernels (using Intel® oneDNN)
2. Memory layout optimizations: set optimal layout for each kernel, while minimizing memory changes in between kernels
3. Graph optimizations: Fusion, Layout Propagation

Intel® Extension for PyTorch* (IPEX)

- Buffer the PRs for stock Pytorch
- Provide users with the up-to-date Intel software/hardware features
- 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

Ease-of-Use User-Facing API (v1.10.x~)

For Float32

```
import torch
import torchvision.models as models

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

model = model.to(memory_format=torch.channels_last)
data = data.to(memory_format=torch.channels_last)

#################### code changes ####################
import intel_extension_for_pytorch as ipex
model = ipex.optimize(model)
######################################################

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

Ease-of-Use User-Facing API (v1.10.x~)

For BFloat16

```
import torch
import torchvision.models as models

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

model = model.to(memory_format=torch.channels_last)
data = data.to(memory_format=torch.channels_last)

#################### code changes ####################
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)
```

Usage

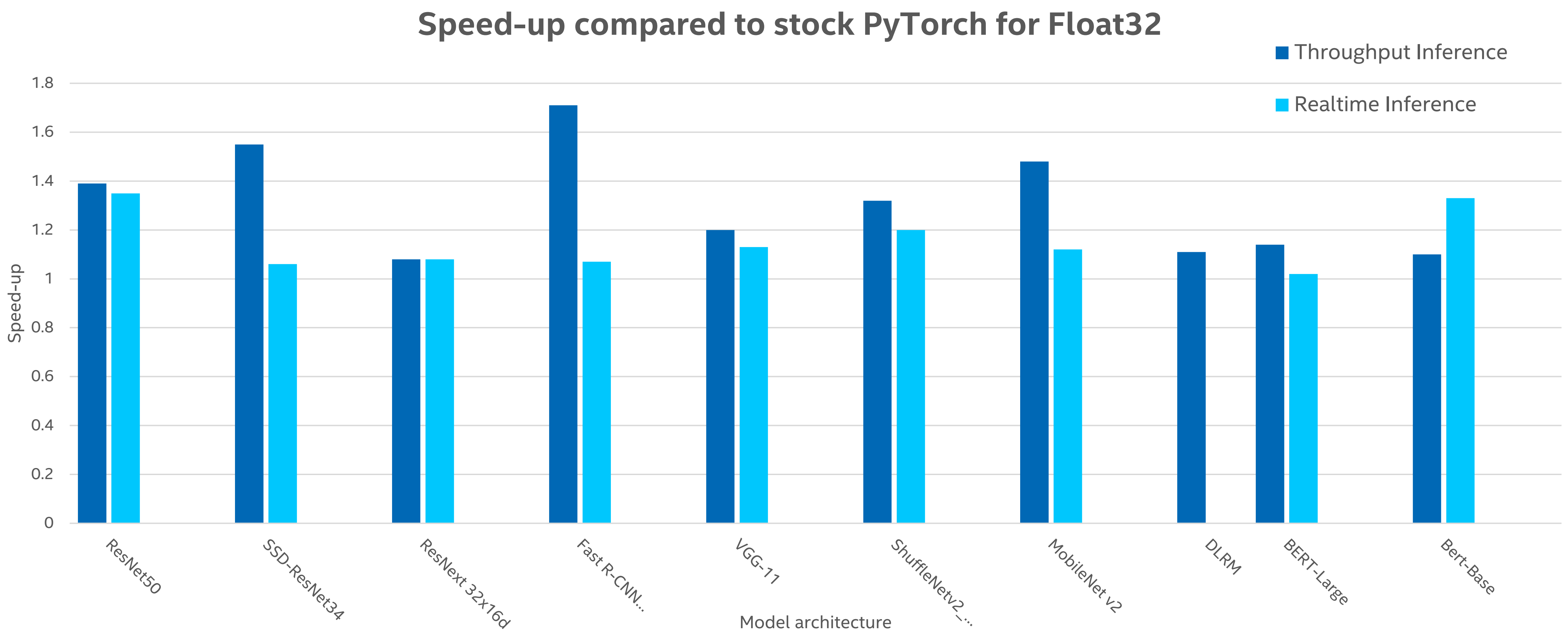
```
import torch
import intel_pytorch_extension

class Model(torch.nn.Module):
  def __init__(self):
    super(Model, self).__init__()
    self.conv2d = torch.nn.Conv2d(3, 5, 5)

  def forward(self, input):
    res = self.conv2d(input)
    return res

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

Prior to v1.10

```
import torch
import intel_extension_for_pytorch as ipex

class Model(torch.nn.Module):
  def __init__(self):
    super(Model, self).__init__()
    self.conv2d = torch.nn.Conv2d(3, 5, 5)

  def forward(self, input):
    res = self.conv2d(input)
    return res

input = torch.randn(5, 3, 9, 9)
model = Model()
model = ipex.optimize(model, dtype=torch.float32, level="O1")
input = input.to(memory_format=torch.channels_last)
res = model(input)
```

v1.10

Intel Extension for PyTorch benchmark

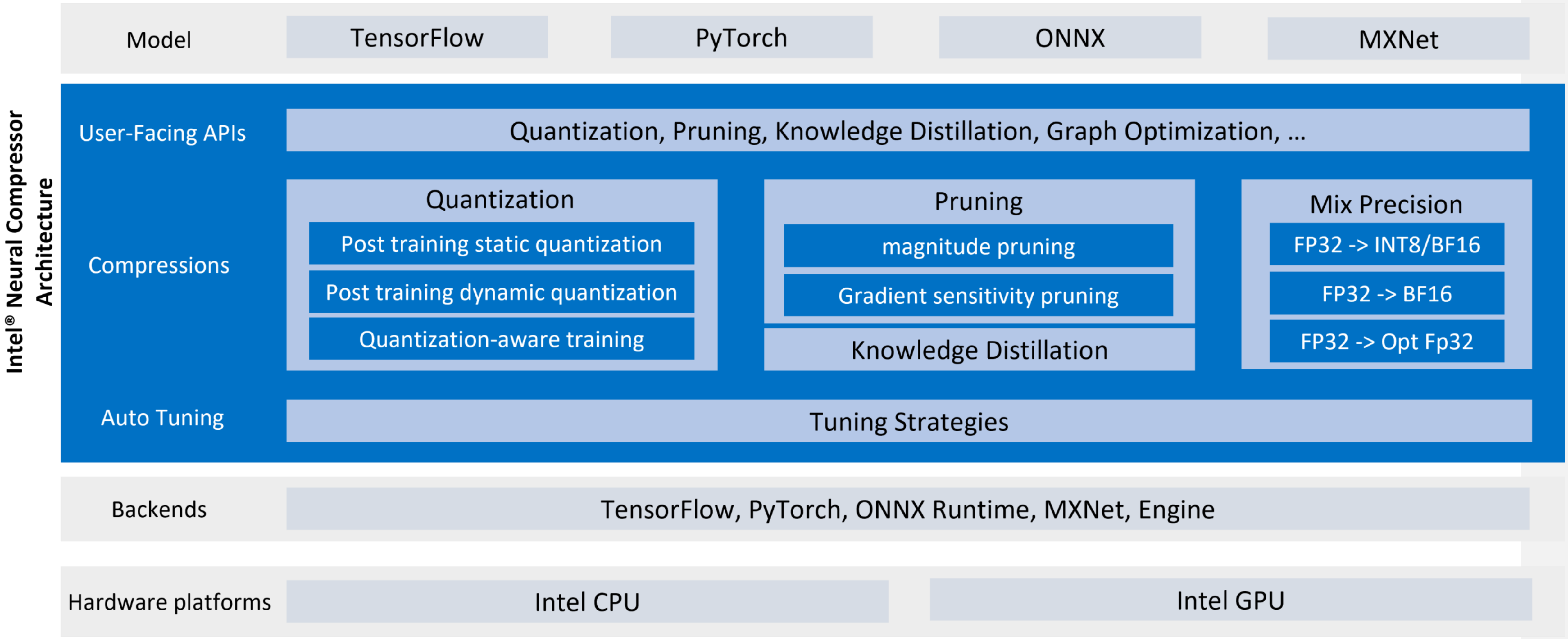


Refer to: https://intel.github.io/intel-extension-for-pytorch/tutorials/performance.html for configuration details

Intel Neural Compressor (INC)

INC: Intel Neural Compressor

- Intel Neural Compressor is an open-source Python library to create low-precision inference solutions on popular deep-learning frameworks
- It supports quantization to BF16/INT8, pruning, knowledge distillation and graph optimizations


Intel® Neural Compressor Architecture
Model
TensorFlow
PyTorch
ONNX
MXNet
User-Facing APIs
Quantization, Pruning, Knowledge Distillation, Graph Optimization, …
Compressions
Quantization
Post training static quantization
Post training dynamic quantization
Quantization-aware training
Pruning
magnitude pruning
Gradient sensitivity pruning
Knowledge Distillation
Mix Precision
FP32 -> INT8/BF16
FP32 -> BF16
FP32 -> Opt Fp32
Auto Tuning
Tuning Strategies
Backends
TensorFlow, PyTorch, ONNX Runtime, MXNet, Engine
Hardware platforms
Intel CPU
Intel GPU

BigDL

BigDL for Big Data AI

Technology Stack

- Bringing AI to Big Data software ecosystem
- Leading with “IA differentiated” domain-specific solutions

Domain Specific *(Vertical)* Solutions

- Friesian *(Recommendation System)*
- Chronos *(Time Series)*
- PPML *(Privacy Preserving ML)*

End-to-End *(Horizontal)* Big Data AI Pipelines

- Orca: Distributed DL *(TF/PyTorch/OpenVINO)* on Spark & Ray
- DLlib: Distributed DL library for Spark

Laptop | K8s Cluster | Hadoop Cluster | Cloud

Value of Big Data AI Toolkit

Value	Example Users
Rich *software ecosystem* for Big Data processing on IA	*Mastercard, BBVA, Alibaba Cloud, Inspur, etc.*
Better *E2E productivity and performance* for AI pipelines	*Burger King, SK Telecom, JD.com, Midea, etc.*
Domain-specific AI solutions for Big Data	*Ant Financial, Capgemini, Mavenir, UnionPay, etc.*

OpenVINO

Intel® Distribution of OpenVINO™ Toolkit

- Tool Suite for High-Performance, Deep Learning Inference
- Fast, accurate real-world results using high-performance, AI and computer vision inference deployed into production across Intel® architecture from edge to cloud

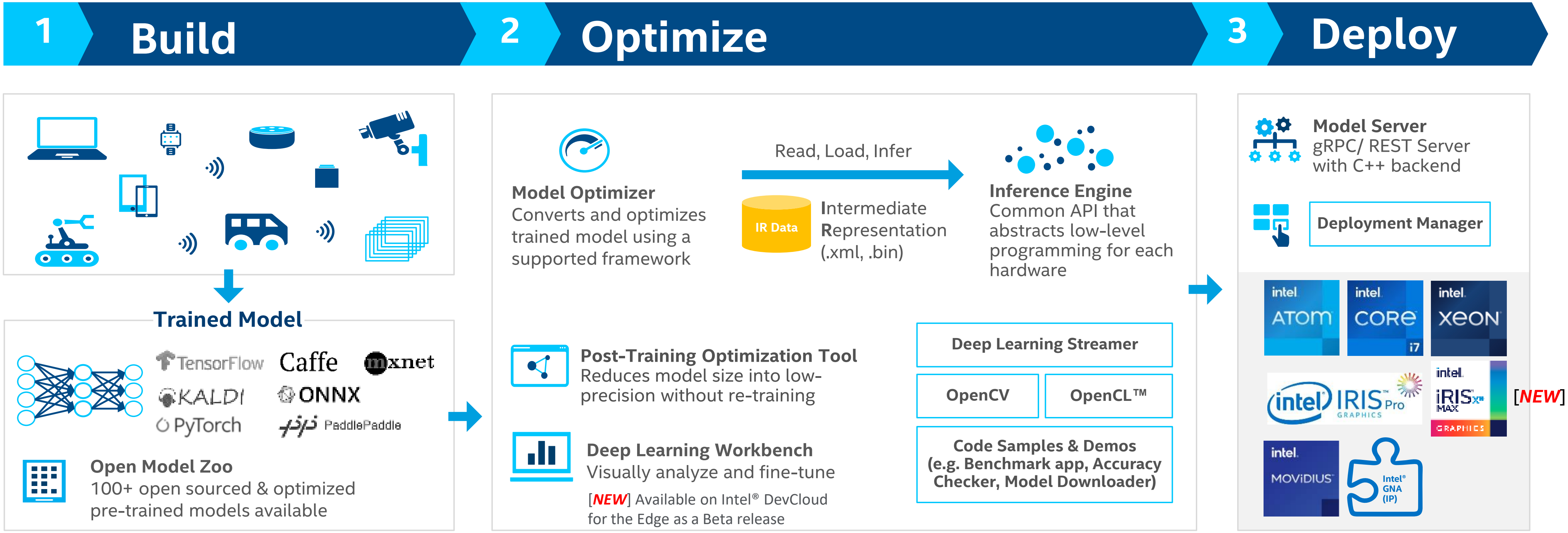
High-Performance,
Deep Learning Inference

Streamlined Development,
Ease of Use

Write Once,
Deploy Anywhere

- Enables deep learning inference from the edge to cloud.
- Supports heterogeneous execution across Intel accelerators, using a common API for the Intel® CPU, Intel® Integrated Graphics, Intel® Gaussian & Neural Accelerator, Intel® Neural Compute Stick 2, Intel® Vision Accelerator Design with Intel® Movidius™ VPUs.
- Speeds time-to-market through an easy-to-use library of CV functions and pre-optimized kernels.
- Includes optimized calls for CV standards, including OpenCV* and OpenCL™.

Three steps for the Intel® Distribution of OpenVINO™ toolkit



Supported Frameworks

Breadth of supported frameworks to enable developers with flexibility

Supported Frameworks and Formats ▸ https://docs.openvinotoolkit.org/latest/_docs_IE_DG_Introduction.html#SupportedFW
Configure the Model Optimizer for your Framework ▸ https://docs.openvinotoolkit.org/latest/_docs_MO_DG_prepare_model_Config_Model_Optimizer.html

Model Optimization

Breadth of supported frameworks to enable developers with flexibility

Model Optimizer loads a model into memory, reads it, builds the internal representation of the model, optimizes it, and produces the **Intermediate Representation**.

Optimization techniques available are:

- Node merging
- Horizontal fusion
- Batch normalization to scale shift
- Fold scale shift with convolution
- Drop unused layers (dropout)

Note: Except for ONNX (.onnx model formats), all models have to be converted to an IR format to use as input to the Inference Engine

.xml – describes the network topology
.bin – describes the weights and biases binary data

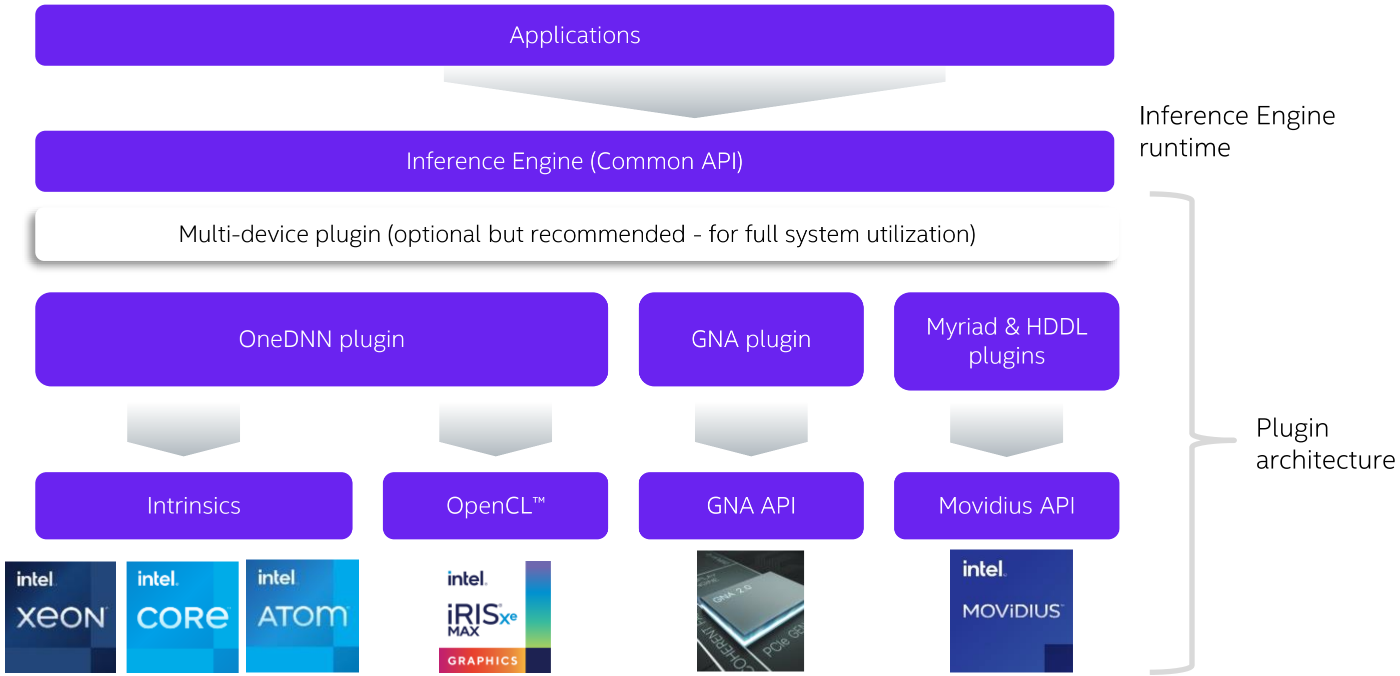
Optimal Model Performance Using the Inference Engine

Core Inference Engine Libraries

- Create Inference Engine Core object to work with devices
- Read the network
- Manipulate network information
- Execute and pass inputs and outputs

Device-specific Plugin Libraries

- For each supported target device, Inference Engine provides a plugin — a DLL/shared library that contains complete implementation for inference on this device.



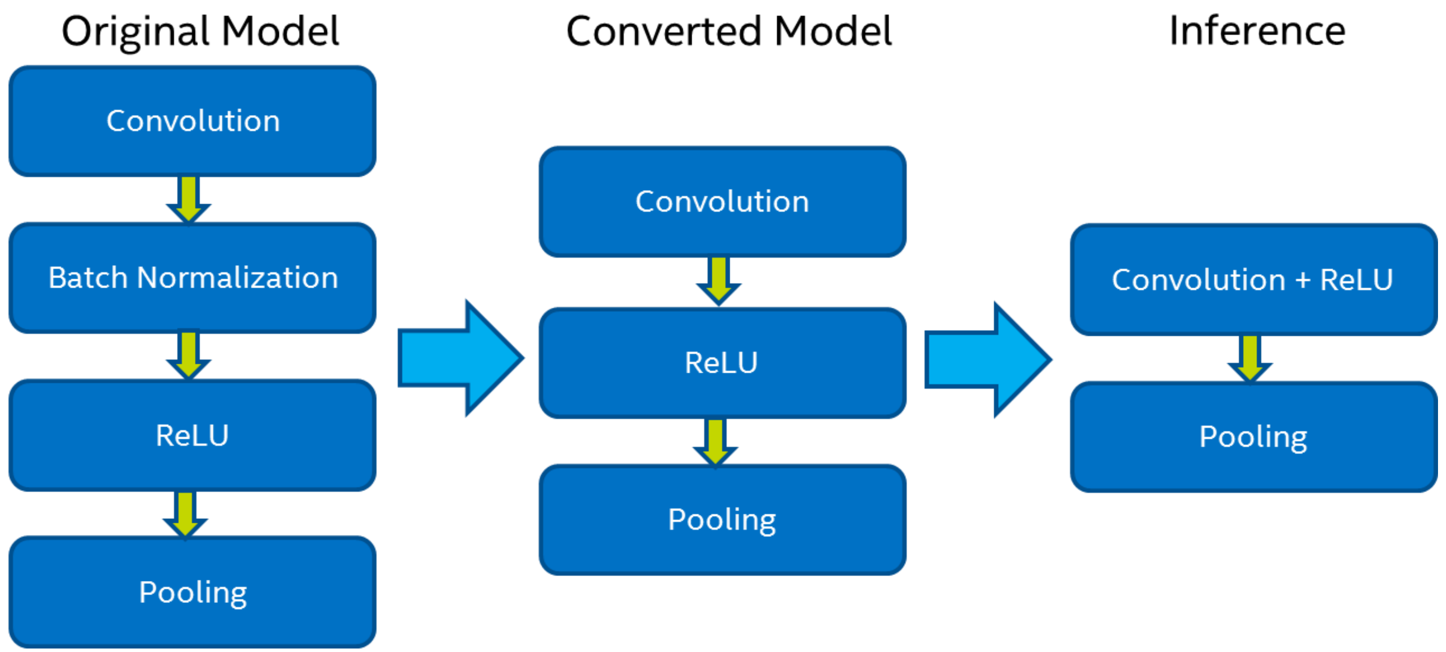
GPU = Intel CPU with integrated graphics/Intel® Processor Graphics/GEN

GNA = Gaussian mixture model and Neural Network Accelerator

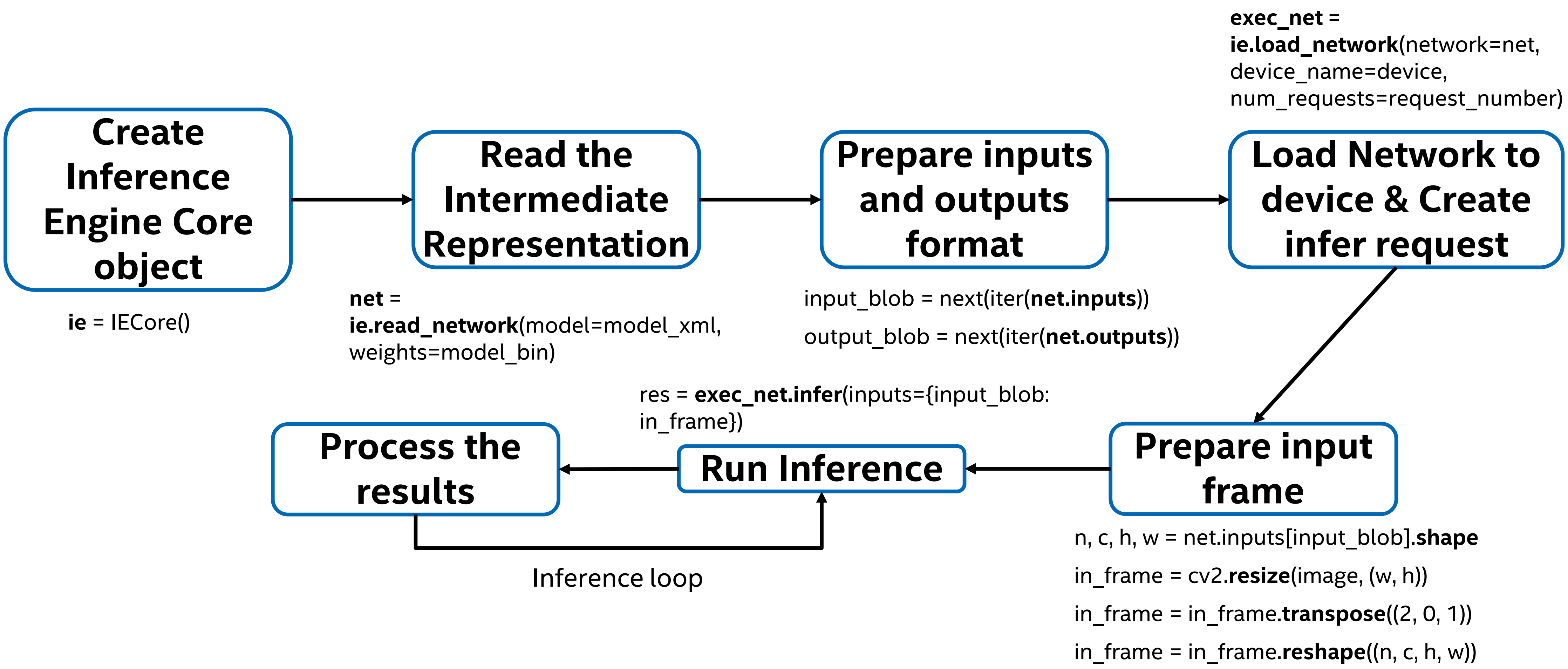
Model Optimizer: Linear Operation Fusing

- Example

1. Remove Batch normalization stage.
2. Recalculate the weights to ‘include’ the operation.
3. Merge Convolution and ReLU into one optimized kernel.



Common Workflow for Using the Inference Engine API



http://docs.openvinotoolkit.org/latest/_docs_IE_DG_Integrate_with_customer_application_new_API.html

Pre-Trained Models and Public Models

Open-sourced repository of pre-trained models and support for public models

Use free **Pre-trained Models** to speed up development and deployment

Take advantage of the **Model Downloader** and other automation tools to quickly get started

Iterate with the **Accuracy Checker** to validate the accuracy of your models

100+ Pre-trained Models
Common AI tasks

Object Detection
Object Recognition
Reidentification
Semantic Segmentation
Instance Segmentation
Human Pose Estimation
Image Processing
Text Detection
Text Recognition
Text Spotting
Action Recognition
Image Retrieval
Compressed Models
Question Answering

100+ Public Models
Pre-optimized external models

Classification
Segmentation
Object Detection
Human Pose Estimation
Monocular Depth Estimation
Image Inpainting
Style Transfer
Action Recognition
Colorization

OpenVINO as execution provider

- You can use OpenVINO Inference Engine as backend of other DL Inference Frameworks such as Tensorflow or ONNX Runtime

- Benefit: the advantages of OpenVINO (multiple HW support and acceleration) in your favorite framework

OpenVINO™ Integration with TensorFlow*

Only 2 lines to be added to regular Tensorflow

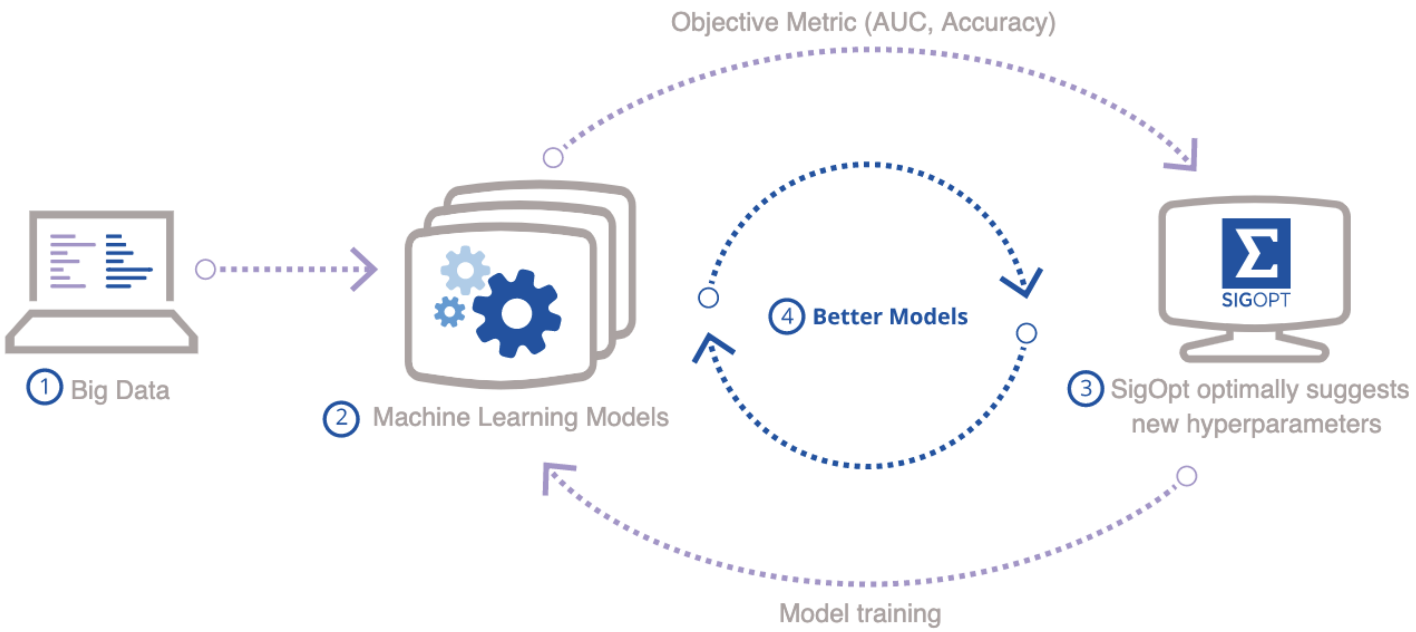
```python
# Installation steps
# more details : https://github.com/openvinotoolkit/openvino_tensorflow
#pip3 install -U pip==21.0.1
#pip3 install -U tensorflow==2.4.1
#pip3 install openvino-tensorflow

# Import package and set backend
import openvino_tensorflow
openvino_tensorflow.set_backend('GPU'))

# Load a TF Saved Model
model = tf.keras.models.load_model('resnet50_saved_model')

# Get the input size of the model
network_input_size = saved_model_loaded.input.shape()

# Resize the input image
resized_image = resize(input_image, network_input_size)

# Run inference
model.predict(resized_image)
```

CPU
GPU
MYRIAD
VAD-M

SigOpt

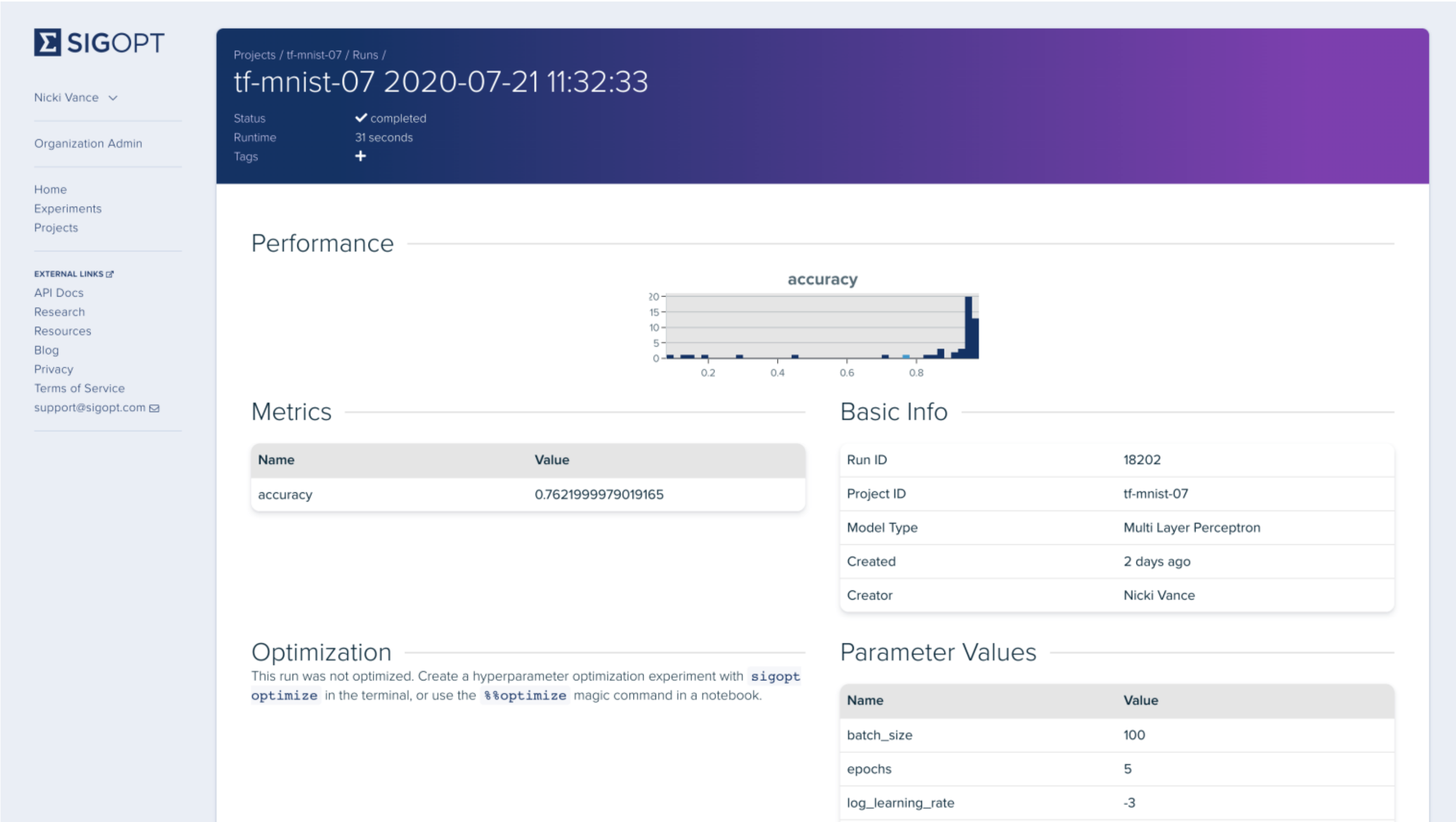
SigOpt

- SigOpt is the only experimentation platform that brings together:
  - Bayesian-based hyperparameter optimization tuning (including multi-metric optimization)



SigOpt

- And experiment management



Example code

Questions?