Choose the Best Accelerated Technology

Intel Performance optimizations for Deep Learning

Shailen Sobhee – Senior AI Software Solutions Engineer
shailen.sobhee@intel.com
14 October 2021

Agenda

- Quick recap of oneAPI
- Overview of oneDNN
- Training:
  - Overview of performance-optimized DL frameworks
    - Tensorflow
    - PyTorch
  - Distributed Training with Intel® Xeon and oneAPI
- Inferencing:
  - Intel® Neural Compressor (old name: Low Precision Optimization Tool)
  - Intro to Intel® Distribution of OpenVINO

Intel's oneAPI Ecosystem

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

oneAPI

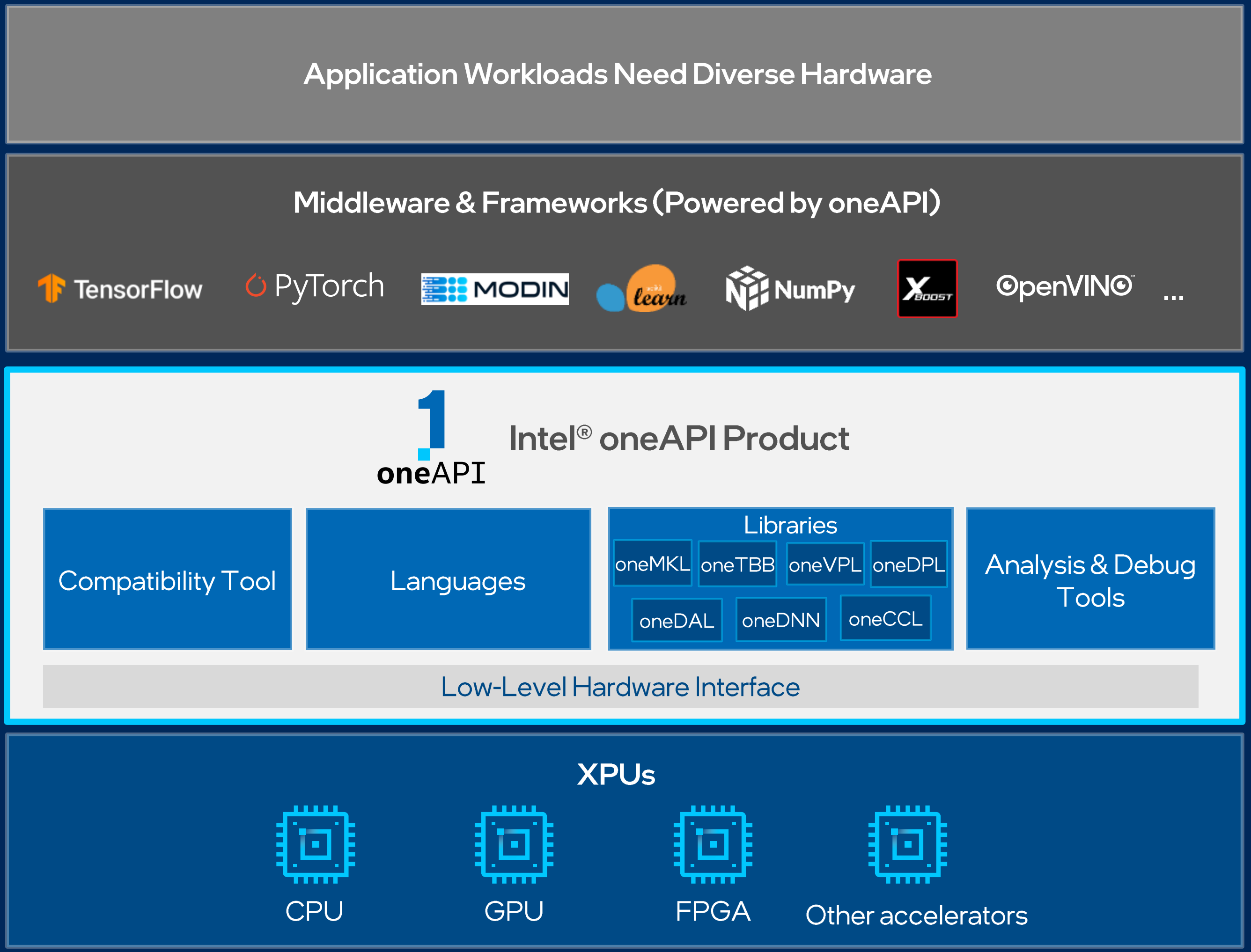
A cross-architecture language based on C++ and SYCL standards

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](#)

Intel® oneAPI Toolkits

A complete set of proven developer tools expanded from CPU to XPU

oneAPI

Intel® oneAPI Base Toolkit

Native Code Developers

A core set of high-performance tools for building C++, Data Parallel C++ applications & oneAPI library-based applications

Add-on Domain-specific Toolkits

Specialized Workloads

Intel® oneAPI Tools for HPC

Deliver fast Fortran, OpenMP & MPI applications that scale

Intel® oneAPI Tools for IoT

Build efficient, reliable solutions that run at network's edge

Intel® oneAPI Rendering Toolkit

Create performant, high-fidelity visualization applications

Toolkits powered by oneAPI

Data Scientists & AI Developers

Intel® AI Analytics Toolkit

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

Intel® Distribution of OpenVINO™ Toolkit

Deploy high performance inference & applications from edge to cloud

Latest version is 2021.1

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

Deep Learning

- Intel® Optimization for TensorFlow
- Intel® Optimization for PyTorch
- Intel® Low Precision Optimization Tool
- 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

Develop Fast Neural Networks on Intel® CPUs & GPUs

with Performance-optimized Building Blocks

Intel® oneAPI Deep Neural Network Library (oneDNN)

intel®

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*

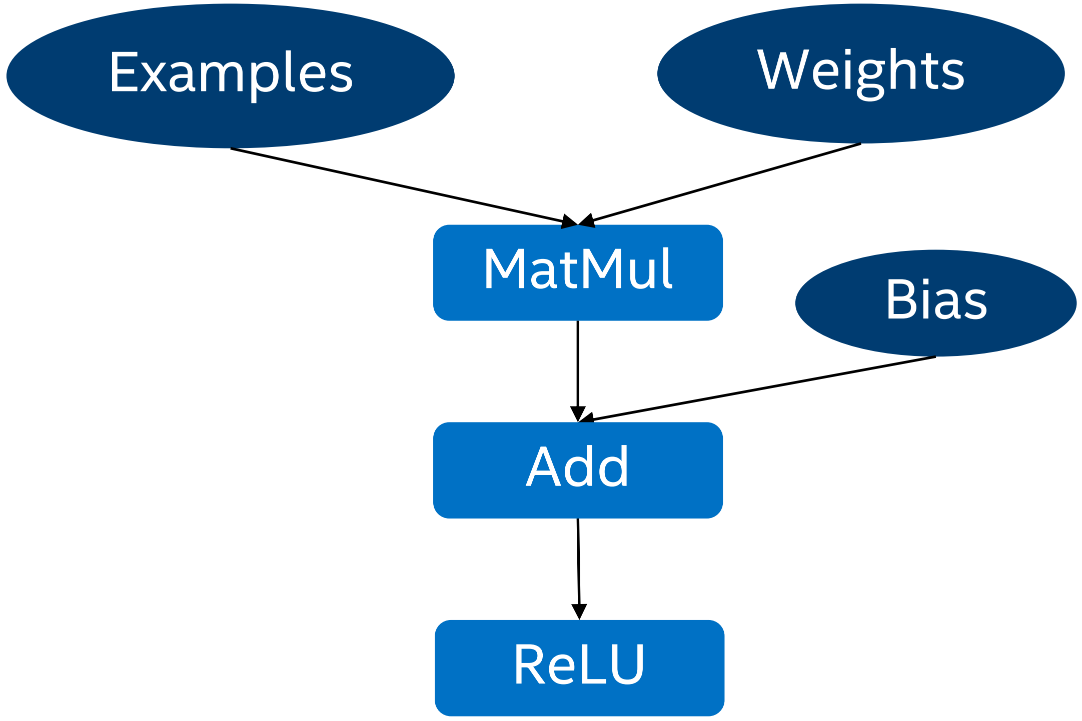
Intel® TensorFlow* optimizations

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.

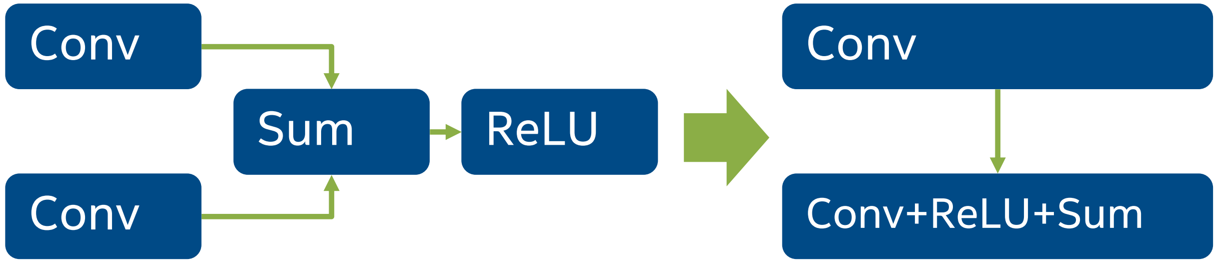


Operator optimizations

- 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.

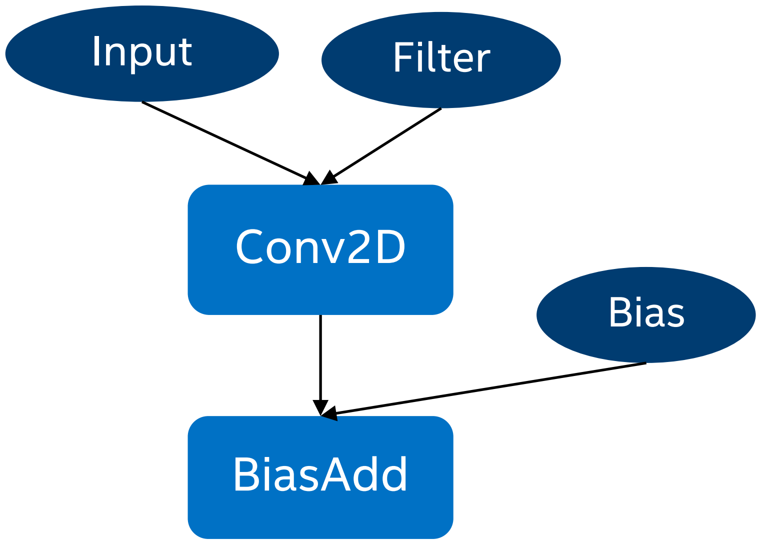
Forward	Backward
Conv2D	Conv2DGrad
Relu, TanH, ELU	ReLUGrad, TanHGrad, ELUGrad
MaxPooling	MaxPoolingGrad
AvgPooling	AvgPoolingGrad
BatchNorm	BatchNormGrad
LRN	LRNGrad
MatMul, Concat	

Fusing computations

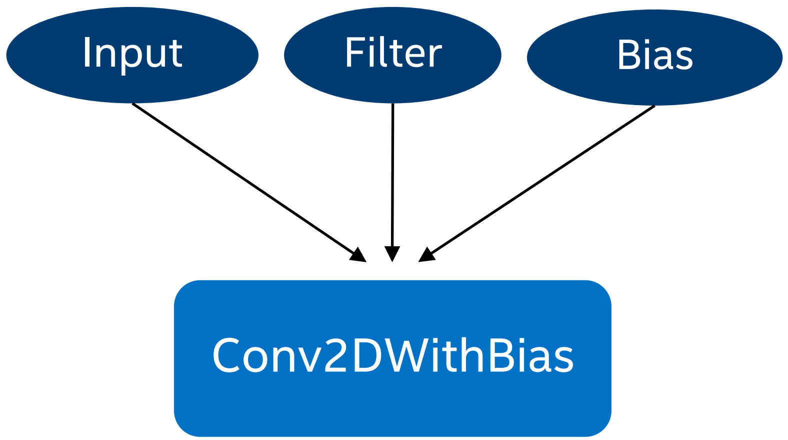


- On Intel processors a high percentation of time is typically spent in BW-limited ops
  - ~40% of ResNet-50, even higher for inference
- The solution is to fuse BW-limited ops with convolutions or one with another to reduce the # of memory accesses
  - Conv+ReLU+Sum, BatchNorm+ReLU, etc
- The frameworks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

Graph optimizations: fusion

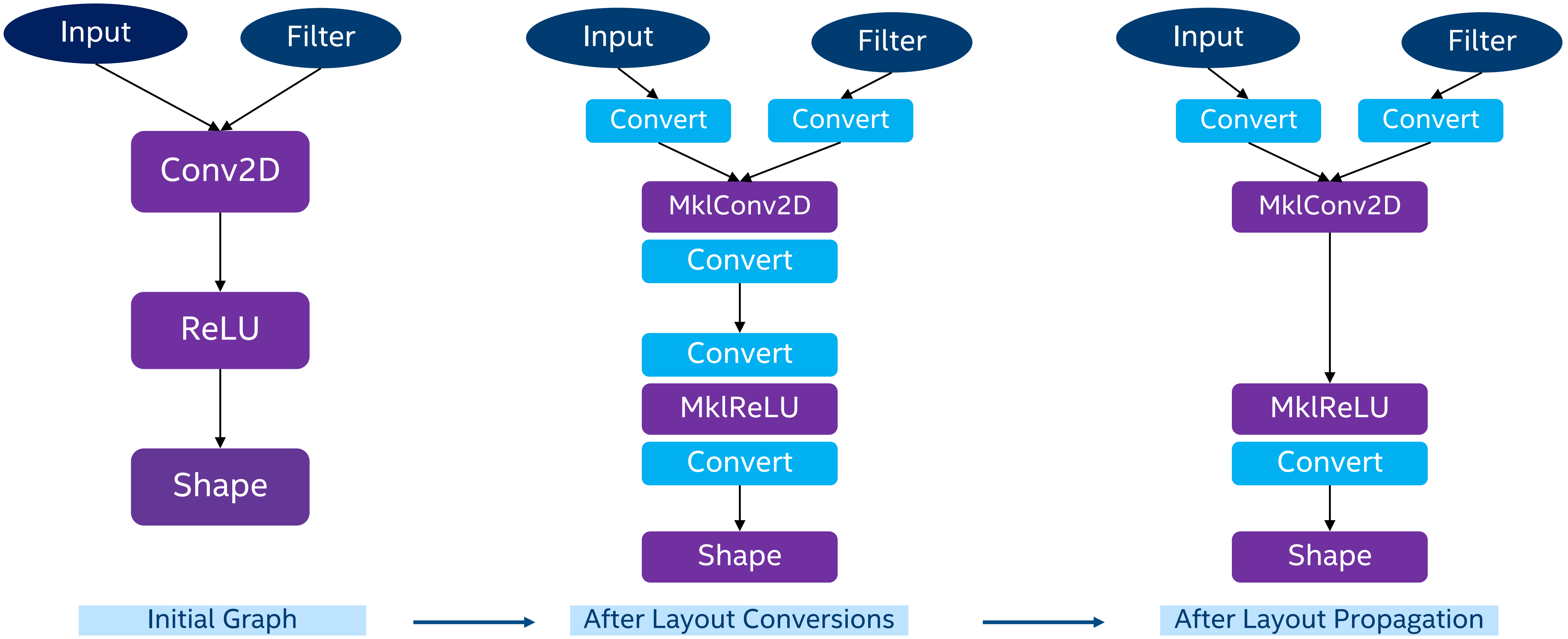


Before Merge



After Merge

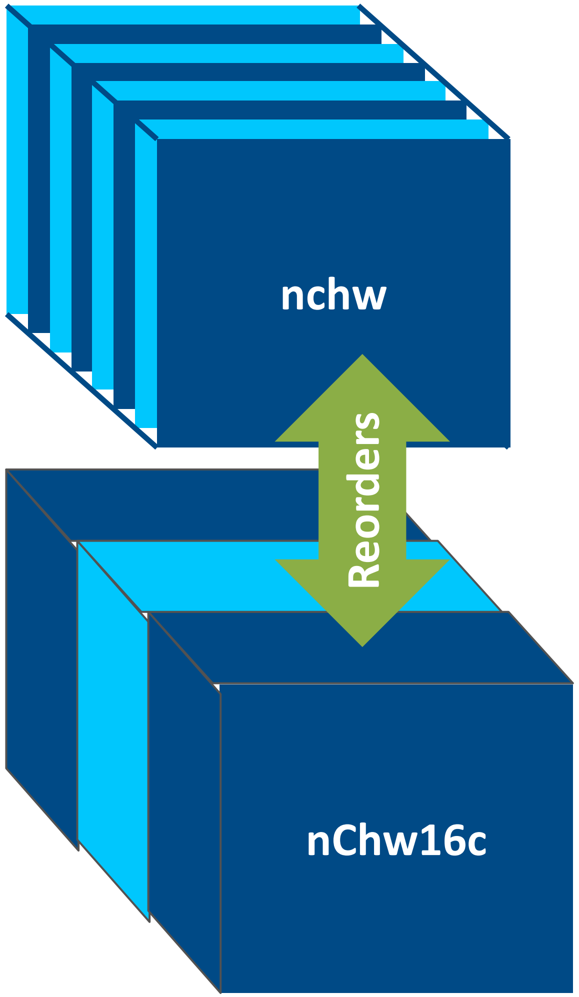
Graph optimizations: layout propagation



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

More on memory channels: Memory layouts

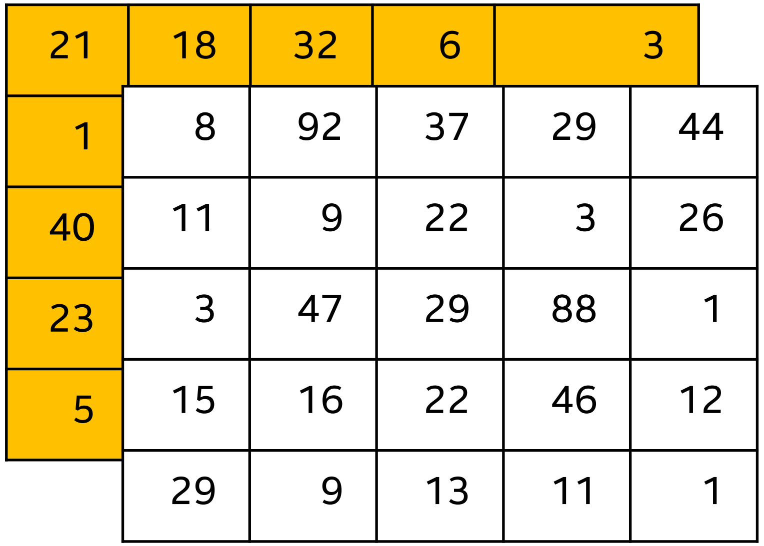
- Most popular memory layouts for image recognition are **nhwc** and **nchw**
  - Challenging for Intel processors either for vectorization or for memory accesses (cache thrashing)
- Intel oneDNN convolutions use blocked layouts
  - Example: **nhwc** with channels blocked by 16 – **nChw16c**
  - Convolutions 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

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	…	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( …)
```

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`**

Performance Guide

- Maximize TensorFlow* Performance on CPU: Considerations and Recommendations for Inference Workloads: https://software.intel.com/en-us/articles/maximize-tensorflow-performance-on-cpu-considerations-and-recommendations-for-inference

Example setting system environment variables with python **`os.environ`** :

```
os.environ["KMP_AFFINITY"] = "granularity=fine,compact,1,0"

os.environ["KMP_SETTINGS"] = "0"
```

Tuning MKL for the best performance

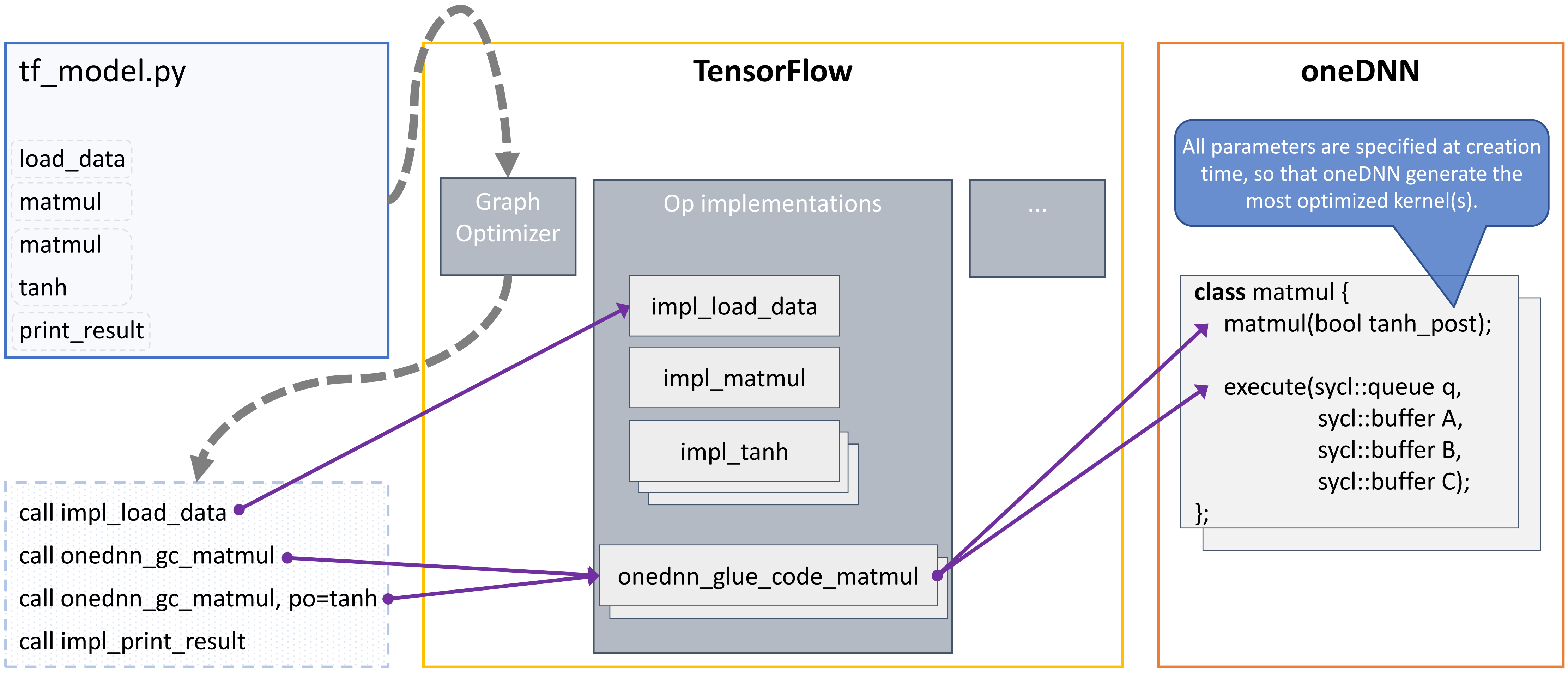
This section details the different configurations and environment variables that can be used to tune the MKL to get optimal performance. Before tweaking various environment variables make sure the model is using the `NCHW` (`channels_first`) data format. The MKL is optimized for `NCHW` and Intel is working to get near performance parity when using `NHWC`.

MKL uses the following environment variables to tune performance:

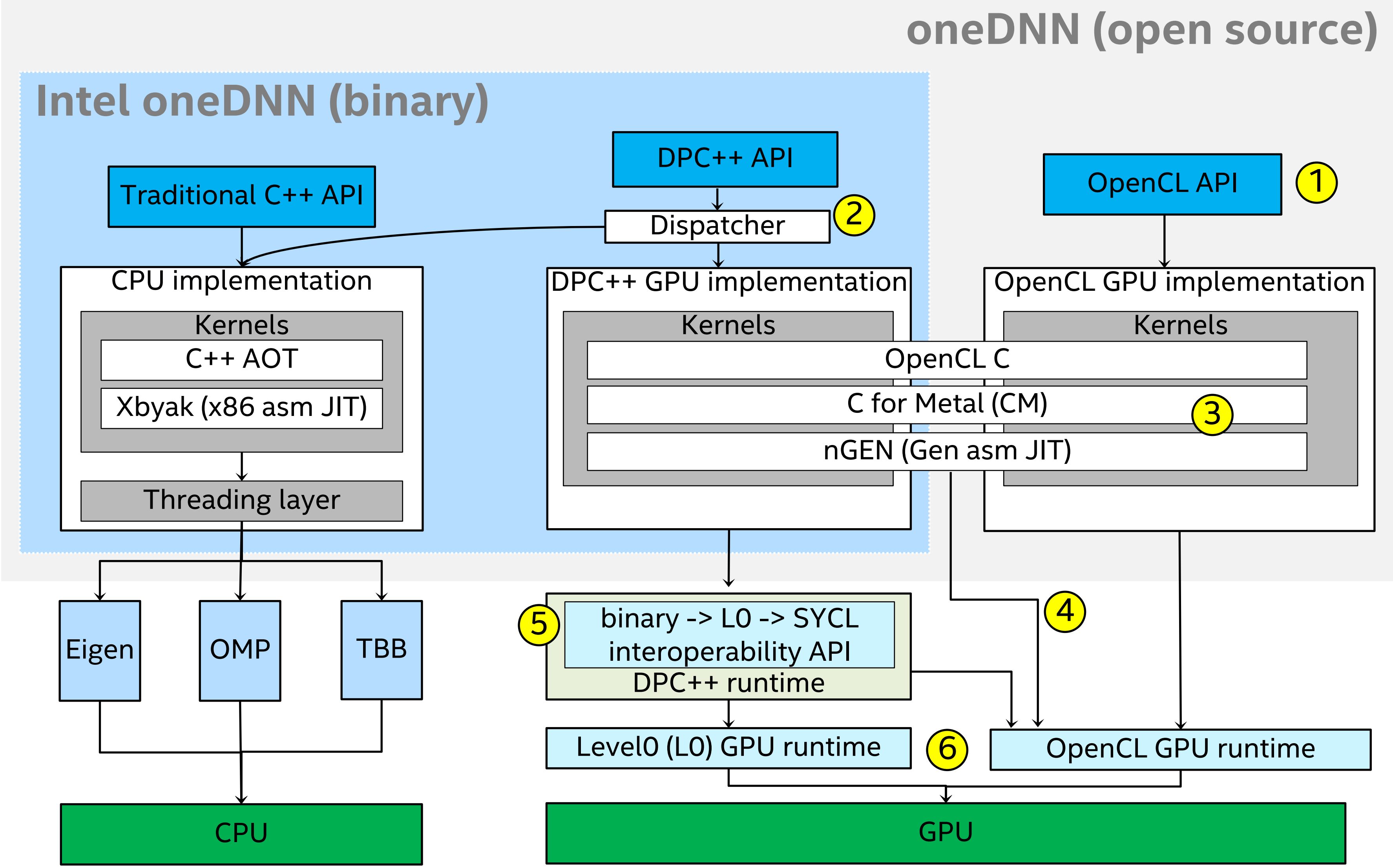
- KMP_BLOCKTIME - Sets the time, in milliseconds, that a thread should wait, after completing the execution of a parallel region, before sleeping.
- KMP_AFFINITY - Enables the run-time library to bind threads to physical processing units.
- KMP_SETTINGS - Enables (true) or disables (false) the printing of OpenMP* run-time library environment variables during program execution.
- OMP_NUM_THREADS - Specifies the number of threads to use.

Intel Tensorflow* install guide is available → https://software.intel.com/en-us/articles/intel-optimization-for-tensorflow-installation-guide

oneDNN <-> Frameworks interaction



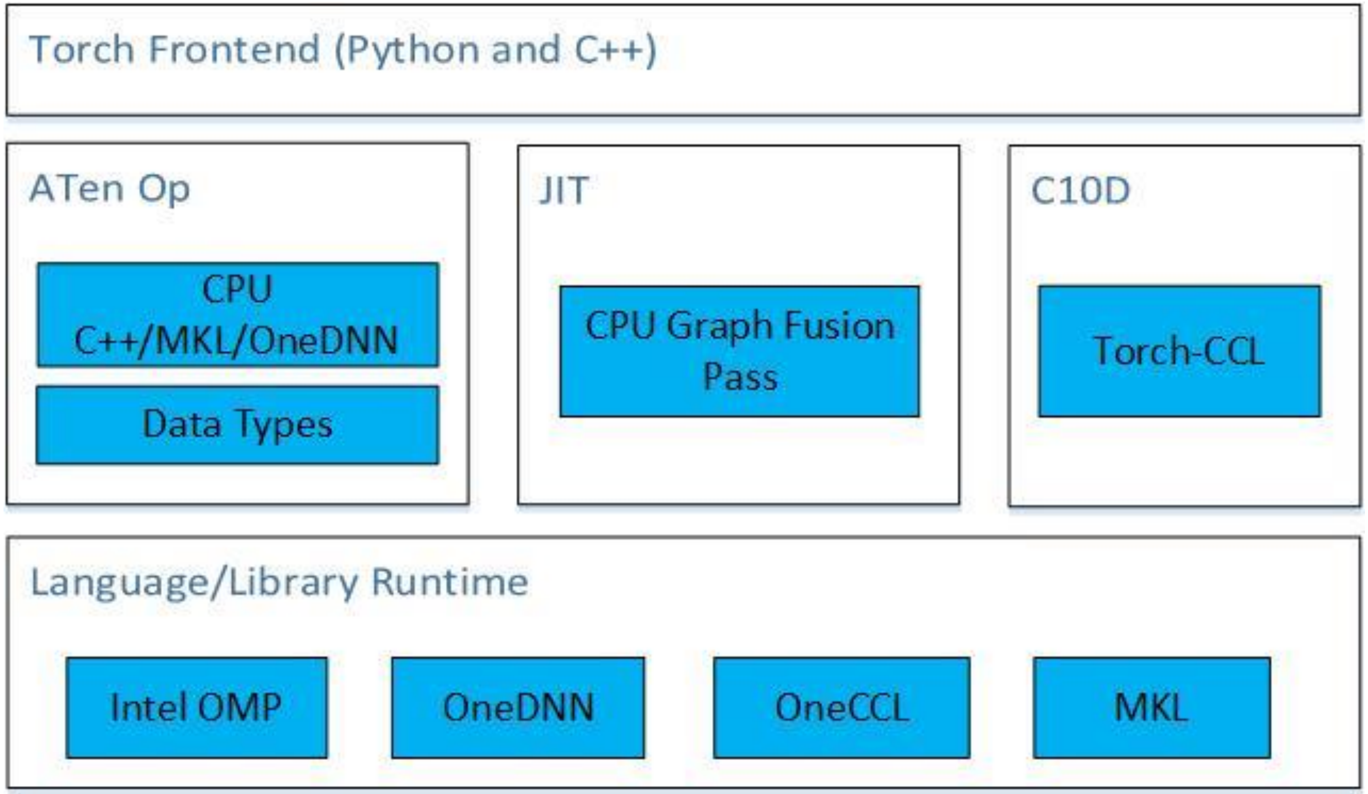
oneDNN architecture overview



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 L0, binary kernels need to be wrapped to L0 modules to create SYCL kernels eventually
6. Under DPC++ API/runtime, users can run on GPU via either OpenCL or L0 GPU runtime: it should be specified in compile time, but can be checked during execution time

Intel Optimizations for PyTorch

- Accelerated operators
- Graph optimization
- Accelerated communications



Motivation for Intel Extension for PyTorch (IPEX)

- Provide customers with the up-to-date Intel software/hardware features
- Streamline the work to enable Intel accelerated library

Operator Optimization

➢Auto dispatch the operators optimized by the extension backend

➢Auto operator fusion via PyTorch graph mode

Mix Precision

➢Accelerate PyTorch operator by bfloat16

➢Automatic mixed precision

PyTorch-IPEX Demo

How to get IPEX

1. oneAPI AI Analytics Toolkit
2. Install from source

IPEX from the oneAPI AI Analytics Toolkit

Intel Optimizations for PyTorch

Intel-Optimized PyTorch

- PyTorch back-end optimizations
- Up-streamed to regular PyTorch
- Same front-end code as regular PyTorch

Intel Extension for PyTorch (IPEX)

- Additional optimizations and Mixed Precision support
- Different front-end

Torch-CCL

- For distributed learning
- PyTorch bindings for oneCCL

Installing IPEX from source

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

Build and install

1. Install PyTorch from source
2. Download and install Intel PyTorch Extension source
3. Add new backend for Intel Extension for PyTorch
4. Install Intel Extension for PyTorch

Automatic Mixed Precision Feature (FP32 + BF16)

```
import torch
import intel_pytorch_extension as ipex
ipex.enable_auto_optimization(mixed_dtype = torch.bfloat16, train = True)

EPOCH = 20
BATCH_SIZE = 128
LR = 0.001

def main():
    train_loader = ...
    test_loader = ...
    net = topology()
    net = net.to(ipex.DEVICE)
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(net.parameters(), lr = LR, momentum=0.9)
    for epoch in range(EPOCH):
        net.train()
        for batch_idx, (data, target) in enumerate(train_loader):
            data = data.to(ipex.DEVICE)
            target = target.to(ipex.DEVICE)
            optimizer.zero_grad()
            output = net(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()

        net.eval()
        test_loss = 0
        correct = 0
        with torch.no_grad():
            for data, target in test_loader:
                data = data.to(ipex.DEVICE)
                target = target.to(ipex.DEVICE)
                output = net(data)
                test_loss += criterion(output, target, reduction='sum').item()
                pred = output.argmax(dim=1, keepdim=True)
                correct += pred.eq(target.view_as(pred)).sum().item()
        test_loss /= len(test_loader.dataset)

if __name__ == '__main__':
    main()
```

1. import ipex

2. Enable Auto-Mix-Precision by API

* Subject to change

3. Convert the input tensors to the extension device

4. Convert the model to the extension device

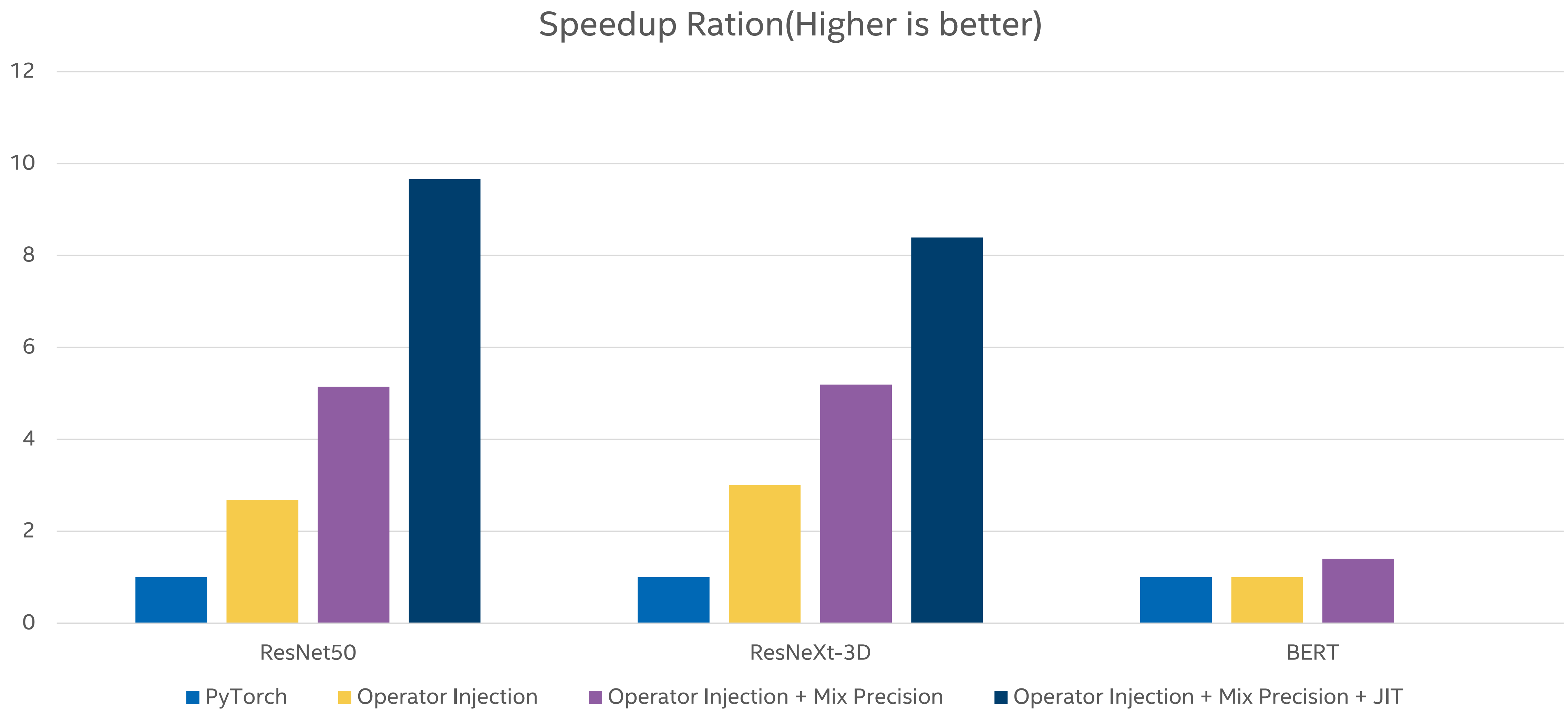
Data types

https://software.intel.com/sites/default/files/managed/40/8b/bf16-hardware-numerics-definition-white-paper.pdf?source=techstories.org

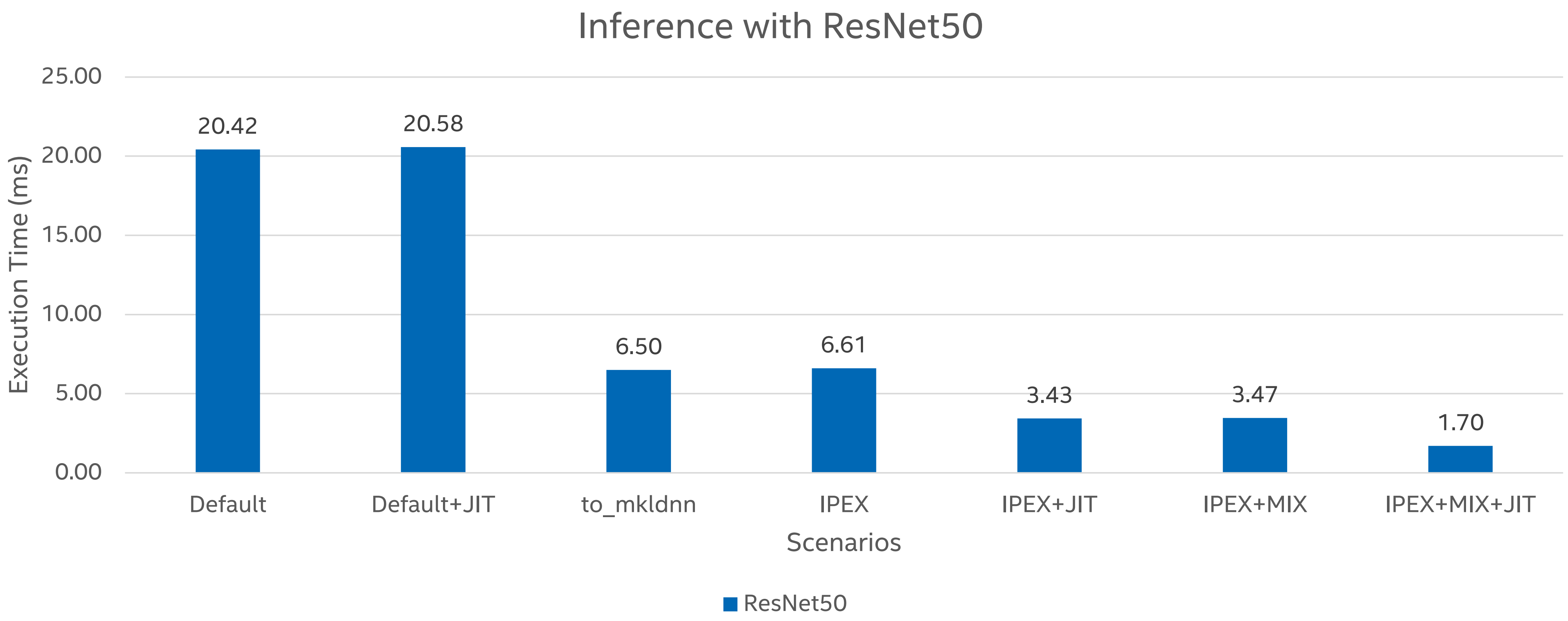
- Benefit of bfloat16
  - Performance 2x up
  - Comparable accuracy loss against fp32
  - No loss scaling, compared to fp16

*** bfloat16 intrinsic support starts from 3rd Generation Intel® Xeon® Scalable Processors**

Extension Performance comparison



Inference with IPEX for ResNet50



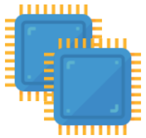
Worker11 (CPX)

LD_PRELOAD=/root/anaconda3/lib/libiomp5.so OMP_NUM_THREADS=26 KMP_AFFINITY=granularity=fine,compact,1,0 numactl -N 0 -m 0 python resnet50.py

Intel Low Precision Optimization Tool Tutorial

The motivation for low precision

Lower Power



Lower memory bandwidth

Lower storage

Higher performance

Important:

Acceptable accuracy loss

The key term:

- Quantization

Quantization in a nutshell

Floating Point		**Integer**
96.1924	→	96
32 -bit		8 bit

10110110	10110110	→	10110110
10110110	10110110		

Challenge & Solution of Low Precision Optimization Tool (for Inferencing in Deep Learning)

- Low Precision Inference can speed up the performance by reducing the computing, memory and storage of AI model.
- Intel provides solution to cover the challenge of it:

Challenge	Intel Solution	How
Hardware support	Intel® Deep Learning Boost supported by the Second-Generation Intel® Xeon® Scalable Processors and later.	VNNI intrinsic. Support INT8 MulAdd.
Complex to convert the FP32 model to INT8/BF16 model	Intel® Low Precision Optimization Tool (LPOT)	Unified quantization API
Accuracy loss in converting to INT8 model	Intel® Low Precision Optimization Tool (LPOT)	Auto tuning

Product Definition

- Convert the FP32 model to INT8/BF16 model. Optimize the model in same time.
- Support multiple Intel optimized DL frameworks (TensorFlow, PyTorch, MXNet) on both CPU and GPU.
- Support automatic accuracy-driven tuning, along with additional custom objectives like performance, model size, or memory footprint
- Provide the easy extension capability for new backends (e.g., PDPD, ONNX RT) and new tuning strategies/metrics (e.g., HAWQ from UCB)

Tuning Zoo

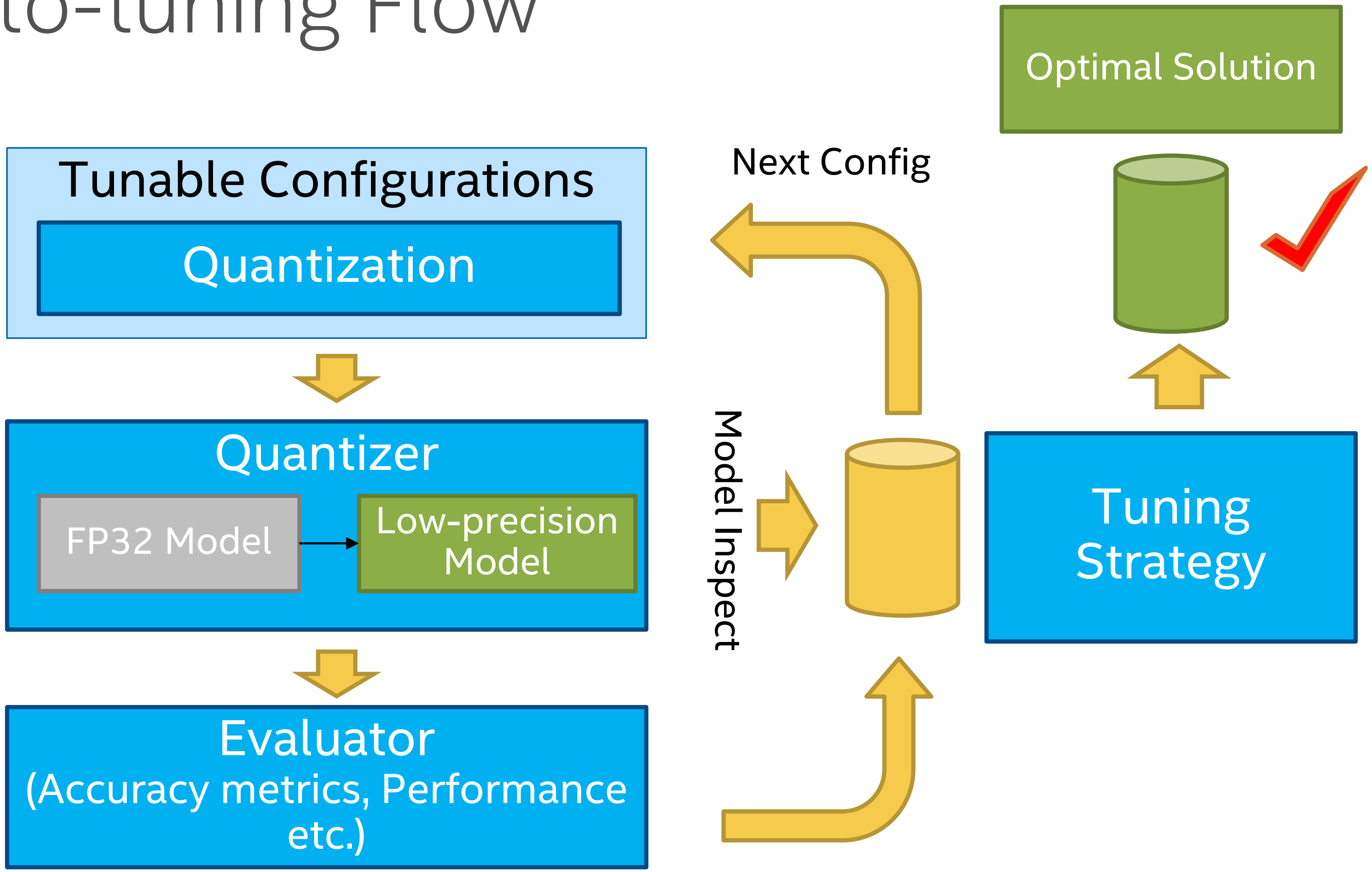
The followings are the models supported by Intel® Low Precision Optimization Tool for auto tuning.

TensorFlow Model	Category
ResNet50 V1	Image Recognition
ResNet50 V1.5	Image Recognition
ResNet101	Image Recognition
Inception V1	Image Recognition
Inception V2	Image Recognition
Inception V3	Image Recognition
Inception V4	Image Recognition
ResNetV2_50	Image Recognition
ResNetV2_101	Image Recognition
ResNetV2_152	Image Recognition
Inception ResNet V2	Image Recognition
SSD ResNet50 V1	Object Detection
Wide & Deep	Recommendation
VGG16	Image Recognition
VGG19	Image Recognition
Style_transfer	Style Transfer

PyTorch Model	Category
BERT-Large RTE	Language Translation
BERT-Large QNLI	Language Translation
BERT-Large CoLA	Language Translation
BERT-Base SST-2	Language Translation
BERT-Base RTE	Language Translation
BERT-Base STS-B	Language Translation
BERT-Base CoLA	Language Translation
BERT-Base MRPC	Language Translation
DLRM	Recommendation
BERT-Large MRPC	Language Translation
ResNext101_32x8d	Image Recognition
BERT-Large SQUAD	Language Translation
ResNet50 V1.5	Image Recognition
ResNet18	Image Recognition
Inception V3	Image Recognition
YOLO V3	Object Detection
Peleenet	Image Recognition
ResNest50	Image Recognition
SE_ResNext50_32x4d	Image Recognition
ResNet50 V1.5 QAT	Image Recognition

MxNet Model	Category
ResNet50 V1	Image Recognition
MobileNet V1	Image Recognition
MobileNet V2	Image Recognition
SSD-ResNet50	Object Detection
SqueezeNet V1	Image Recognition
ResNet18	Image Recognition
Inception V3	Image Recognition

Auto-tuning Flow



System Requirements

▪ Hardware

Intel® Low Precision Optimization Tool supports systems based on Intel 64 architecture or compatible processors.

The quantization model could get acceleration by Intel® Deep Learning Boost if running on the Second-Generation Intel® Xeon® Scalable Processors and later:

Verified:

- Cascade Lake & Cooper Lake, with Intel DL Boost VNNI
- Skylake, with AVX-512 INT8

▪ OS: Linux

Verified: CentOS 7.3 & Ubuntu 18.04

▪ Software

Intel® Low Precision Optimization Tool requires to install Intel optimized framework version for TensorFlow, PyTorch, and MXNet.

Verified Release	Installation Example
Intel Optimization for TensorFlow: v1.15 (up1), v2.1, v2.2, v2.3	pip install intel-tensorflow==2.3.0
PyTorch: v1.5	pip install torch==1.5.0+cpu******
MXNet: v1.6, v1.7	pip install mxnet-mkl==1.6.0

Installation

- Install from Intel AI Analytics Toolkit (Recommended)

```
source /opt/intel/oneapi/setvars.sh
conda activate tensorflow
cd /opt/intel/oneapi/iLiT/latest
sudo ./install_iLiT.sh
```

- Install from source

```
git clone https://github.com/intel/lpot.git
cd lpot
python setup.py install
```

- Install from binary

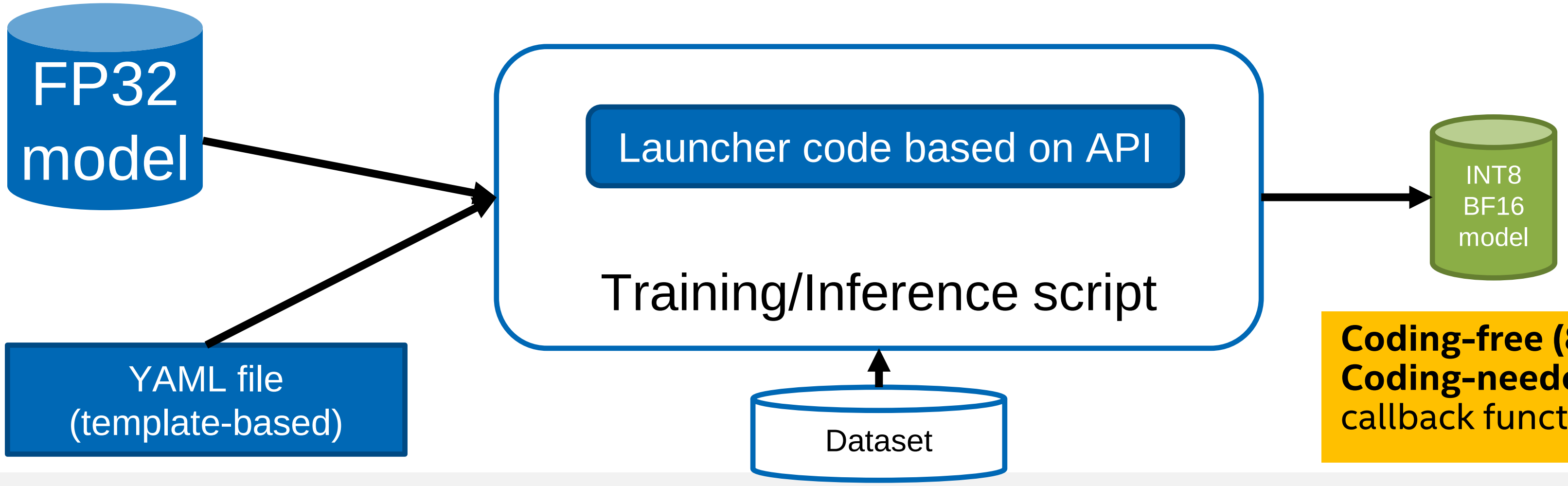
```
# install from pip
pip install lpot
# install from conda
conda install lpot -c intel -c conda-forge
```

For more detailed installation info, please refer to https://github.com/intel/lpot

Usage: Simple Python API + YAML config

LPOT is designed to reduce the workload of the user and keep the flexibility.

Python API	YAML
• Simple API is easy to integrated in original training/inference script.	• Common functions are integrated and controlled by parameters; • Templates are easy to refer; • Lots of advance parameters provide powerful tuning capability.



Coding-free (80%): template-based configs
Coding-needed (20%): user providing callback functions

Python API

- Core User-facing API:

❑ Quantization()

- Follow a specified tuning strategy to tune a low precision model through QAT or PTQ which can meet pre-defined accuracy goal and objective.

```
class Quantization(object):
    def __init__(self, conf_fname):
        ...

    def __call__(self, model, q_dataloader=None, q_func=None,
                 eval_dataloader=None, eval_func=None):
        ...
```

Intel LPOT YAML Configure

Intel LPOT YAML config consists of 6 building blocks:

- ❑ model
- ❑ device
- ❑ quantization
- ❑ evaluation
- ❑ tuning

```
# ilit yaml building block
model:          # model specific info, such as model name, framework,
input/output node name required for tensorflow.
  ...

device: ...     # the device ilit runs at, cpu or gpu. default is cpu.

quantization:   # the setting of calibration/quantization behavior. only
required for PTQ and QAT.
  ...

evaluation:     # the setting of how to evaluate a model.
  ...

tuning:         # the tuning behavior, such as strategy, objective, accuracy
criterion.
  ...
```

Easy: TensorFlow ResNet50

YAML config

```yaml
model:
  name: resnet50_v1_5
  framework: tensorflow
  inputs: input_tensor
  outputs: softmax_tensor

quantization:
  calibration:
    sampling_size: 50, 100
    dataloader:
      batch_size: 10
      dataset:
        Imagenet:
          root: /path/to/calibration/dataset
      transform:
        ParseDecodeImagenet:
        ResizeCropImagenet:
          height: 224
          width: 224
          mean_value: [123.68, 116.78, 103.94]

evaluation:
  accuracy:
    metric:
      topk: 1
    dataloader:
      batch_size: 32
      dataset:
        Imagenet:
          root: /path/to/evaluation/dataset
      transform:
        ParseDecodeImagenet:
        ResizeCropImagenet:
          height: 224
          width: 224
          mean_value: [123.68, 116.78, 103.94]

tuning:
  accuracy_criterion:
    relative:  0.01
  exit_policy:
    timeout: 0
  random_seed: 9527
```

```python
from lpot import Quantization
quantizer = Quantization("./conf.yaml")
q_model = quantizer(model)
```

Code change

Full example:
https://github.com/intel/lpot/tree/master/examples/tensorflow/image_recognition

DEMO

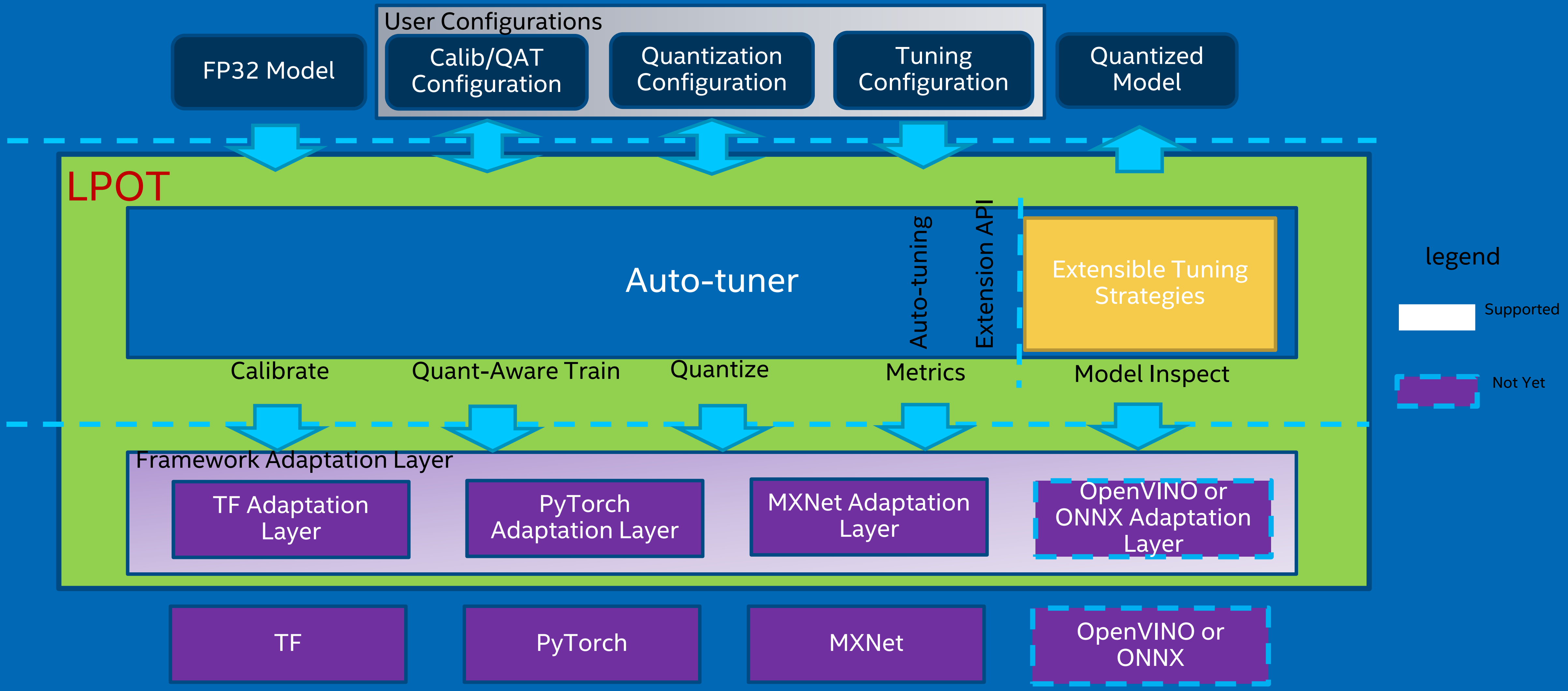
Demo

- **Intel AI Analytics Toolkit Samples:**
- https://github.com/oneapi-src/oneAPI-samples/tree/master/AI-and-Analytics

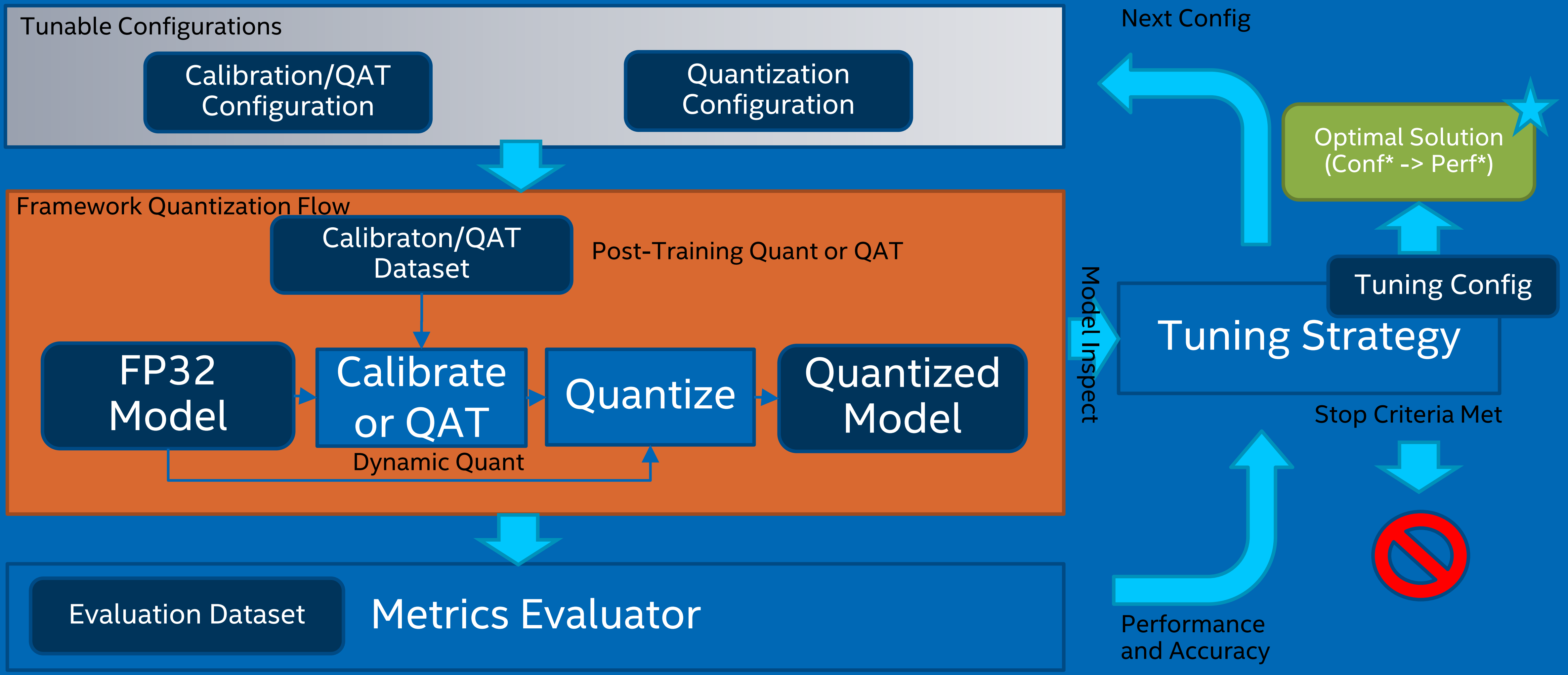
- Intel LPOT Sample for Tensorflow:-samples
- https://github.com/oneapi-src/oneAPI-samples/tree/master/AI-and-Analytics/Getting-Started-Samples/LPOT-Sample-for-Tensorflow

intel.

Infrastructure



Working Flow



OpenVINO™

Product Overview

Intel® Distribution of OpenVINO™ toolkit

VISION PRODUCT OF THE YEAR 2019

2019 Developer Tool of the Year
Awarded by the Edge AI and Vision Alliance

intel®

WRITE once, deploy & scale diversely

TensorFlow

ONNX

mxnet

KALDI

Caffe

Model Optimizer

OpenVINO™

Inference Engine

CPU

intel XEON inside

FPGA

intel STRATIX inside

Edge

intel MOVIDIUS inside

GPU

intel Iris Graphics

From a bird's eye-view

Advanced capabilities to streamline deep learning deployments

1. Build

Trained Model

TensorFlow Caffe
KALDI mxnet
ONNX

Open Model Zoo

100+ open sourced and optimized pre-trained models;
80+ supported public models

2. Optimize

Model Optimizer

Converts and optimizes trained model using a supported framework

Read, Load, Infer

IR Data

Intermediate Representation (.xml, .bin)

Post-Training Optimization Tool

Deep Learning Workbench

Deep Learning Streamer

OpenCV

OpenCL™

Code Samples & Demos
(e.g. Benchmark app, Accuracy Checker, Model Downloader)

3. Deploy

Inference Engine

Common API that abstracts low-level programming for each hardware

Deployment Manager

- CPU Plugin
- GPU Plugin
- GNA Plugin
- Myriad Plugin (For Intel® NCS2 & NCS)
- HDDL Plugin
- FGPA Plugin

intel ATOM · intel CORE i7 · intel XEON · intel IRIS Pro GRAPHICS · intel MOVIDIUS · intel ARRIA 10

Get Started

Typical workflow from development to deployment

Train a model

Find a trained model

Run the Model Optimizer

Intermediate Representation

.bin, .xml

Deploy using the Inference Engine

intel XEON

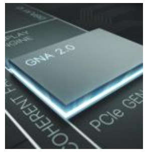
intel CORE i7

intel ATOM

intel IRIS Pro GRAPHICS

intel MOVIDIUS

intel ARRIA 10



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

Core Components

Model optimization to deployment

Model Optimizer

- A Python-based tool to **import** trained models and **convert** them to Intermediate Representation
- **Optimizes for performance** or space with conservative topology transformations
- **Hardware-agnostic** optimizations

Development Guide ▸
https://docs.openvinotoolkit.org/latest/_docs_MO_DG_Deep_Learning_Model_Optimizer_DevGuide.html

Inference Engine

- High-level, C, C++ and Python, inference **runtime API**
- Interface is implemented as **dynamically loaded plugins** for each hardware type
- Delivers best performance for each type **without requiring users to implement and maintain multiple code pathways**

Development Guide ▸
https://docs.openvinotoolkit.org/latest/_docs_IE_DG_Deep_Learning_Inference_Engine_DevGuide.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:

— Linear operation fusing

— Stride optimizations

— Group convolutions fusing

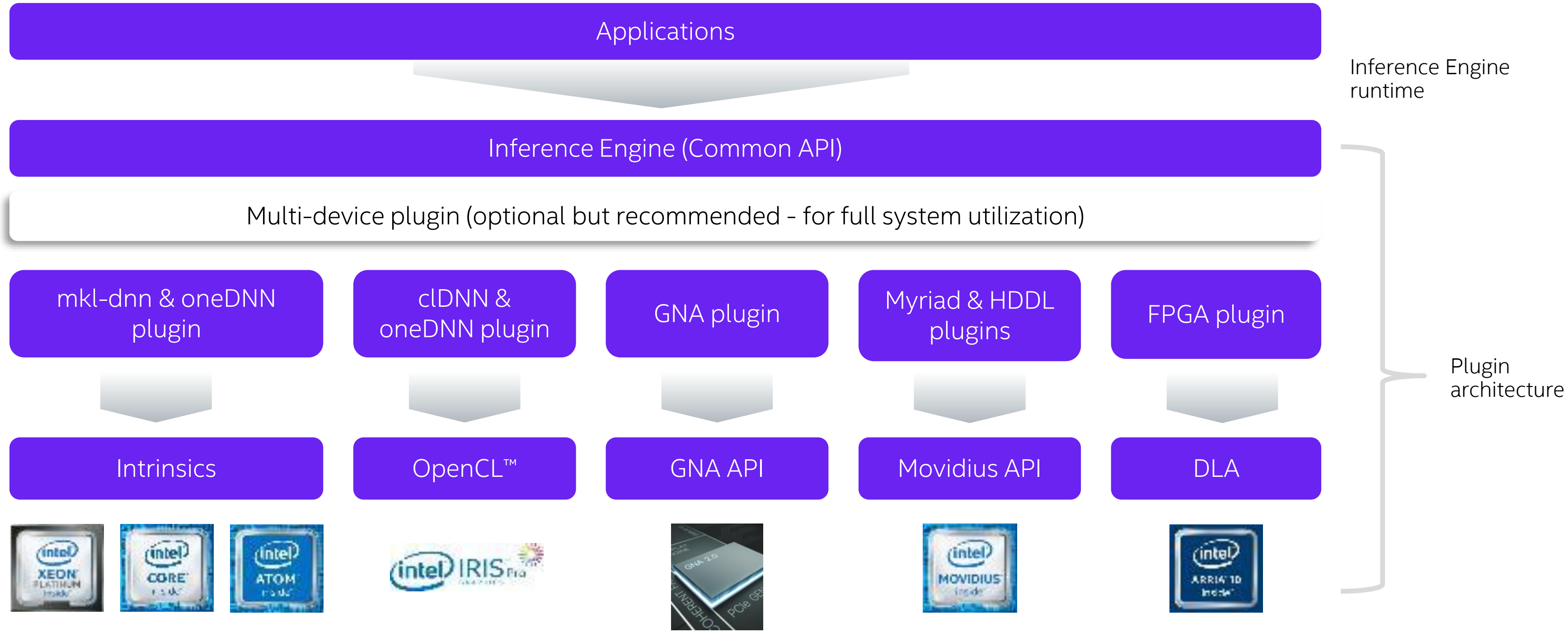
Note: Except for ONNX (.onnx model formats), all models have to be converted to an IR format to use as input to the Inference Engine

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

Inference Engine

Common high-level inference runtime for cross-platform flexibility

OpenVINO

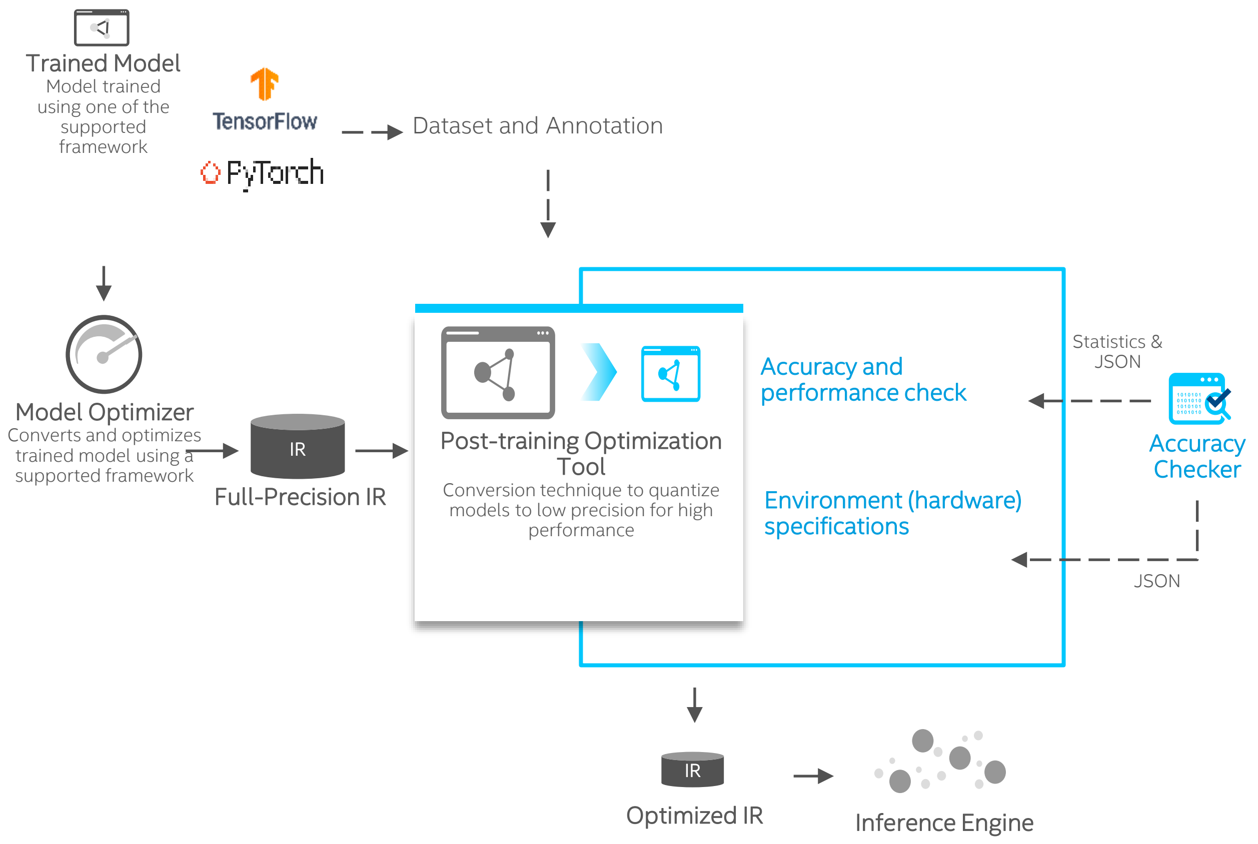


Post-Training Optimization Tool

Conversion technique that reduces model size into low-precision without re-training

Reduces model size **while also improving latency, with little degradation** in model accuracy and without model re-training.

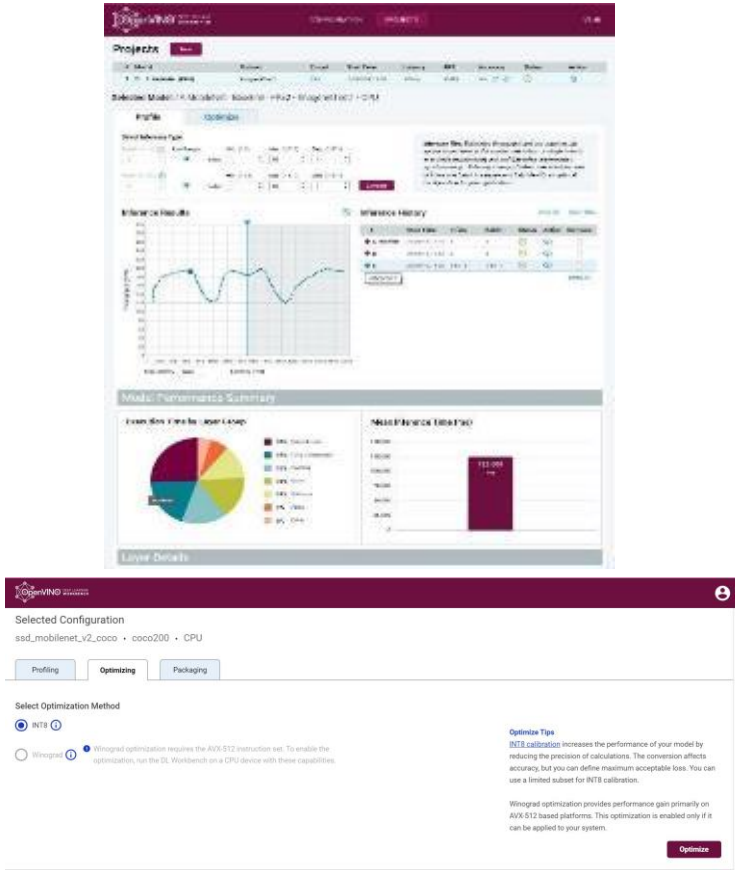
Different optimization approaches are supported: quantization algorithms, sparsity, etc.



Deep Learning Workbench

Web-based UI extension tool for model analyses and graphical measurements

- **Visualizes performance data for** topologies and layers to aid in model analysis
- **Automates analysis** for optimal performance configuration (streams, batches, latency)
- **Experiment with INT8 or Winograd calibration** for optimal tuning using the Post Training Optimization Tool
- Provide **accuracy informatio**n through accuracy checker
- **Direct access to models** from public set of Open Model Zoo
- Enables **remote profiling**, allowing the collection of performance data from multiple different machines without any additional set-up.



Additional Tools and Add-ons

Streamlined development experience and ease of use

Model Downloader

- Provides an easy way of accessing a number of public models as well as a set of pre-trained Intel models

Deployment Manager

- Generate an optimal, minimized runtime package for deployment
- Deploy with smaller footprint compared to development package

Benchmark App

- Measure performance (throughput, latency) of a model
- Get performance metrics per layer and overall basis

Accuracy Checker

- Check for accuracy of the model (original and after conversion) to IR file using a known data set

Computer Vision Annotation Tool
This web-based tool helps annotate videos and images before training a model

Deep Learning Streamer
Streaming analytics framework to create and deploy complex media analytics pipelines

OpenVINO™ Model Server
Scalable inference server for serving optimized models and applications

Dataset Management Framework
Use this add-on to build, transform and analyze datasets

Neural Network Compression Framework
Training framework based on PyTorch* for quantization-aware training

Training Extensions
Trainable deep learning models for training with custom data

Write Once, Deploy Anywhere

Common high-level inference runtime for cross-platform flexibility

Write once, deploy across different platforms with the same API and framework-independent execution

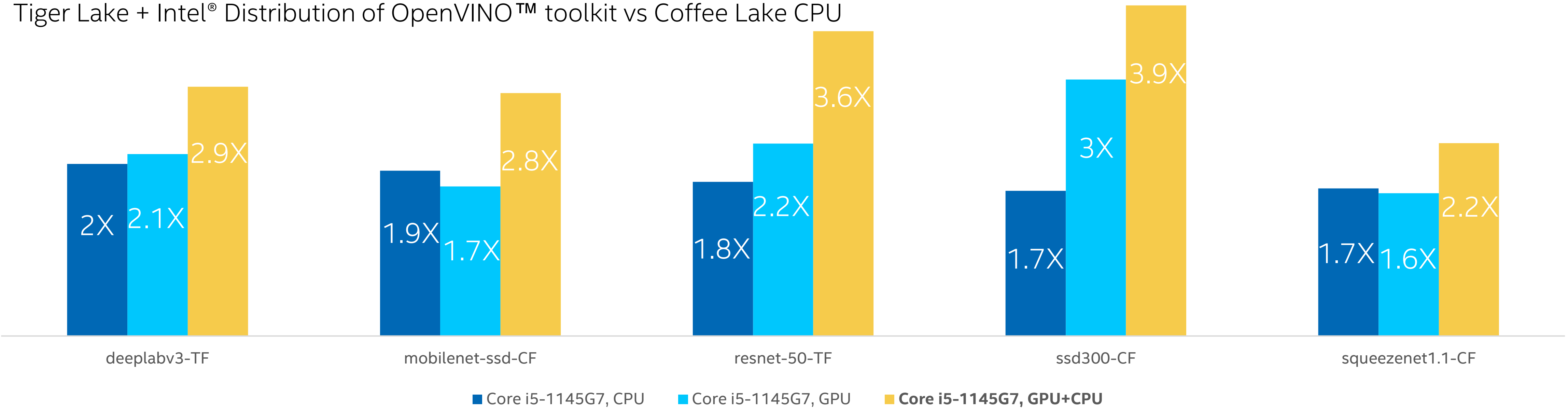
Consistent accuracy, performance and functionality across all target devices with no re-training required

Full environment utilization, or multi-device plugin, across available hardware for greater performance results

Compounding Effect of Hardware and Software

Use Intel® Xᵉ Graphics + CPU combined for maximum inferencing

Tiger Lake + Intel® Distribution of OpenVINO™ toolkit vs Coffee Lake CPU



11th Gen Intel® Core™ (Tiger Lake) Core i5-1145G7 relative inference FPS compared to Coffee Lake, Core i5-8500

Using the Multi-device plugin

The above is preliminary performance data based on pre-production components. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks. See backup for configuration details.

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

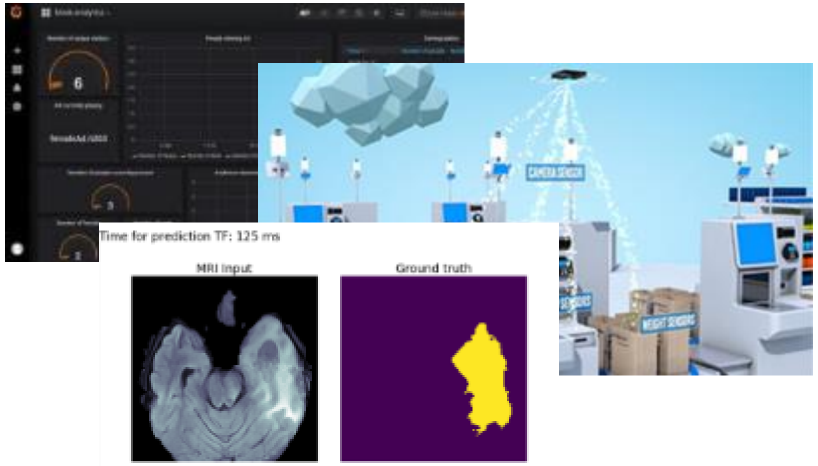
Classification
Segmentation
Object Detection
Human Pose Estimation
Monocular Depth Estimation
Image Inpainting
Style Transfer
Action Recognition
Colorization

DEMO

Demos and Reference Implementations

Quickly get started with example demo applications and reference implementations

Take advantage of **pre-built, open-sourced** example implementations with step-by-step guidance and required components list



- [Face Access Control - C++]
- [Intruder Detector - C++]
- [Machine Operator Monitor - C++]
- [Machine Operator Monitor - Go]
- [Motor Defect Detector - Python]
- [Object Flaw Detector - C++]
- [Object Size Detector - C++]
- [Object Size Detector - Go]
- [Parking Lot Counter - C++]
- [Parking Lot Counter - Go]
- [People Counter - C++]
- [Restricted Zone Notifier - Go]
- [Shopper Gaze Monitor - C++]
- [Shopper Mood Monitor - Go]
- [Store Aisle Monitor - C++]
- [Store Traffic Monitor - C++]
- [Store Traffic Monitor - Python]

Case Studies

Use cases and successful implementation across a variety of industries powered by the Intel® Distribution of OpenVINO™ toolkit

JLK INSPECTION **Healthcare Access and Quality** Solution Brief

Reduced average inference time on Intel® NUC (with no GPU) from 4.23 seconds to just 2.81 seconds, which helps medical professionals reach more people, accelerate screening and help improve quality of care.

ZEROFOX **Security Against Social and Digital Attacks** Solution Brief

Performance improvements of up to 2.3x faster, reducing latency by up to 50 percent for threat detection and remediation to protect businesses against targeted social and digital attacks.

dc water is life® **Sewer pipe inspection analysis** Solution Brief

Inference time was improved with a reduction of up to 80% using Intel Xeon processors with the OpenVINO toolkit, while not producing significant loss in model precision or accuracy.

Retail Business Services **Frictionless retail checkout** Solution Brief

Using existing Intel-based point-of-sale systems, automatic inventory and shopper tracking with cashier-less checkout at a physical retail store was deployed at Quincy, Massachusetts.

Success Stories ▸ https://intel.com/openvino-success-stories

Resources and Community Support

Vibrant community of developers, enterprises and skills builders

QUALIFY	INSTALLATION	PREPARE	HANDS ON	SUPPORT
▪ Use a trained model and check if framework is supported - or – ▪ Take advantage of a pre-trained model from the Open Model Zoo	▪ Download the Intel® OpenVINO™ toolkit package from Intel® Developer Zone, or by YUM or APT repositories ▪ Utilize the Getting Started Guide	▪ Understand sample demos and tools included ▪ Understand performance ▪ Choose hardware option with Performance Benchmarks ▪ Build, test and remotely run workloads on the Intel® DevCloud for the Edge before buying hardware	▪ Visualize metrics with the Deep Learning Workbench ▪ Utilize prebuilt, Reference Implementations to become familiar with capabilities ▪ Optimize workloads with these performance best practices ▪ Use the Deployment Manager to minimize deployment package	▪ Ask questions and share information with others through the Community Forum ▪ Engage using #OpenVINO on Stack Overflow ▪ Visit documentation site for guides, how to's, and resources ▪ Attend training and get certified ▪ Ready to go to market? Tell us how we can help

Ready to get started?

Download directly from Intel for free

Intel® Distribution of OpenVINO ™ toolkit (Recommended)

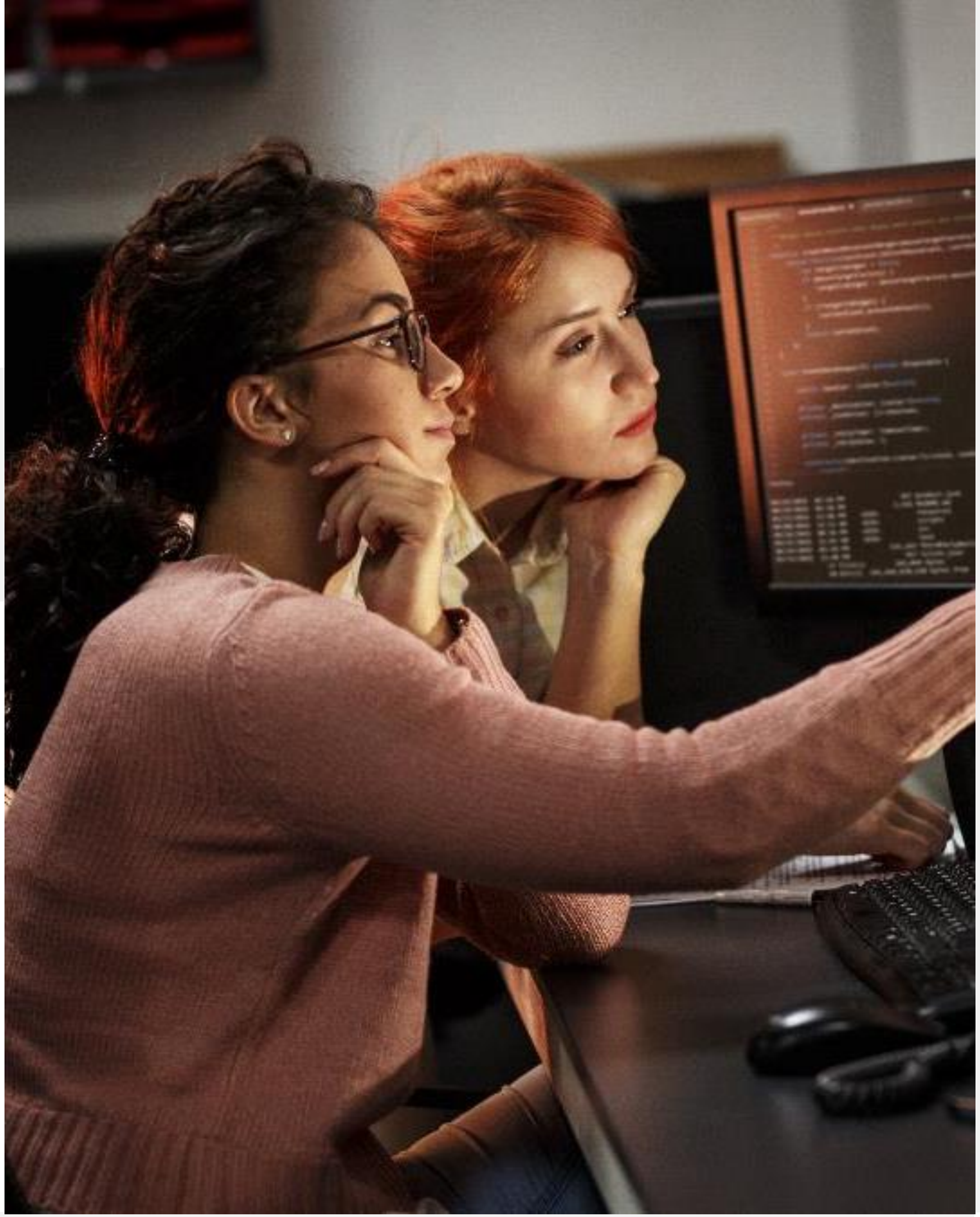
Also available from

Intel's Edge Software Hub | Intel® DevCloud for the Edge | PIP | DockerHub | Dockerfile | Anaconda Cloud | YUM | APT

Build from source

GitHub | Gitee (for China)

Choose & Download



Choose between Distributions

Tool/Component	Intel® Distribution of OpenVINO™ toolkit	OpenVINO™ toolkit (open source)	Open Source Directory
Installer (including necessary drivers)	✓		
Model Optimizer	✓	✓	https://github.com/openvinotoolkit/openvino/tree/master/model-optimizer
Inference Engine - Core	✓	✓	https://github.com/openvinotoolkit/openvino/tree/master/inference-engine
Intel CPU plug-in	✓ Intel® Math Kernel Library (Intel® MKL) only[1]	✓ BLAS, Intel® MKL[1], jit (Intel MKL)	https://github.com/openvinotoolkit/openvino/tree/master/inference-engine
Intel GPU (Intel® Processor Graphics) plug-in	✓	✓	https://github.com/openvinotoolkit/openvino/tree/master/inference-engine
Heterogeneous plug-in	✓	✓	https://github.com/openvinotoolkit/openvino/tree/master/inference-engine
Intel GNA plug-in	✓	✓	https://github.com/openvinotoolkit/openvino/tree/master/inference-engine
Intel® FPGA plug-in	✓		
Intel® Neural Compute Stick (1 & 2) VPU plug-in	✓	✓	https://github.com/openvinotoolkit/openvino/tree/master/inference-engine
Intel® Vision Accelerator based on Movidius plug-in	✓		
Multi-device & hetero plug-ins	✓	✓	
Public and Pretrained Models - incl. Open Model Zoo (IR models that run in IE + open sources models)	✓	✓	https://github.com/openvinotoolkit/open_model_zoo
Samples (APIs)	✓	✓	https://github.com/openvinotoolkit/openvino/tree/master/inference-engine
Demos	✓	✓	https://github.com/openvinotoolkit/open_model_zoo
Traditional Computer Vision			
OpenCV*	✓	✓	https://github.com/opencv/opencv
Intel® Media SDK	✓	✓[2]	https://github.com/Intel-Media-SDK/MediaSDK
OpenCL™ Drivers & Runtimes	✓	✓[2]	https://github.com/intel/compute-runtime
FPGA Runtime Environment, Deep Learning Acceleration & Bitstreams (Linux* only)	✓		

System Requirements

	Intel® Platforms	Compatible Operating Systems
Target Solution Platforms	**CPU** ▪ 6^{th}-10^{th} generation Intel® Core™ and Xeon® processors ▪ 1^{st} and 2^{nd} generation Intel® Xeon® Scalable processors	▪ Ubuntu* 18.04.3 LTS (64 bit) ▪ Microsoft Windows* 10 (64 bit) ▪ CentOS* 7.4 (64 bit) ▪ macOS* 10.13 & 10.14 (64 bit)
	▪ Intel® Pentium® processor N4200/5, N3350/5, N3450/5 with Intel® HD Graphics	▪ Yocto Project* Poky Jethro v2.0.3 (64 bit)
	Iris® Pro & Intel® HD Graphics ▪ 6^{th}-10^{th} generation Intel® Core™ processor with Intel® Iris™ Pro graphics & Intel® HD Graphics ▪ Intel® Xeon® processor with Intel® Iris™ Pro Graphics & Intel® HD Graphics (excluding E5 product family, which does not have graphics[1])	▪ Ubuntu 18.04.3 LTS (64 bit) ▪ Windows 10 (64 bit) ▪ CentOS 7.4 (64 bit)
	FPGA ▪ Intel® Arria® FPGA 10 GX development kit ▪ Intel® Programmable Acceleration Card with Intel® Arria® 10 GX FPGA operating systems ▪ OpenCV* & OpenVX* functions must be run against the CPU or Intel® Processor Graphics (GPU)	▪ Ubuntu 18.04.2 LTS (64 bit) ▪ CentOS 7.4 (64 bit)
	VPU: Intel Movidius™ Neural Compute Stick:, Intel® Neural Compute Stick2	▪ Ubuntu 18.04.3 LTS (64 bit) CentOS 7.4 (64 bit) ▪ Windows 10 (64 bit) macOS* (64 bit) Raspbian (target only)
	Intel® Vision Accelerator Design Products ▪ Intel® Vision Accelerator Design with Intel® Arria10 FPGA	▪ Ubuntu 18.04.2 LTS (64 bit)
	▪ Intel® Vision Accelerator Design with Intel® Movidius™ VPUs	▪ Ubuntu 8.04.3 LTS (64 bit) ▪ Windows 10 (64 bit)
Development Platforms	▪ 6^{th}-10^{th} generation Intel® Core™ and Intel® Xeon® processors ▪ 1st and 2nd generation Intel® Xeon® Scalable processors	▪ Ubuntu* 18.04.3 LTS (64 bit) ▪ Windows® 10 (64 bit) ▪ CentOS* 7.4 (64 bit) ▪ macOS* 10.13 & 10.14 (64 bit)
Additional Software Requirements	Linux* build environment required components ▪ OpenCV 3.4 or higher ▪ GNU Compiler Collection (GCC) 3.4 or higher ▪ CMake* 2.8 or higher ▪ Python* 3.4 or higher Microsoft Windows* build environment required components ▪ Intel® HD Graphics Driver (latest version)† ▪ OpenCV 3.4 or higher ▪ Intel® C++ Compiler 2017 Update 4 ▪ CMake 2.8 or higher ▪ Python 3.4 or higher ▪ Microsoft Visual Studio* 2015	
External Dependencies/Additional Software		View Product Site, detailed System Requirements

Commonly Asked Questions

Can I use the Intel® Distribution of OpenVINO™ toolkit for commercial usage? Yes, the Intel® Distribution of OpenVINO™ toolkit is licensed under Intel's End User License Agreements and the open-sourced OpenVINO™ toolkit is licensed under Apache License 2.0. For information, review the licensing directory inside the package.

Is the Intel® Distribution of OpenVINO™ toolkit subject to export control? Yes, the ECCN is EAR99.

How often does the software get updated? Standard releases are updated 3-4 times a year, while LTS releases are updated once a year.

What is the difference between Standard and LTS releases? Standard Releases are recommended for new users and users currently prototyping. It offers new features, tools and support to stay current with deep learning advancements. LTS Releases are recommended for experienced users that are ready to take their application into production and who do not require new features and capabilities for their application.

For technical questions, visit the Model Optimizer FAQ and Performance Benchmarks FAQ. If you don't find an answer, please visit the following community and support links.

Get Help

- Ask on the Community Forum
- Contact Intel Support
- File an Issue on GitHub*
- Get Answers on StackOverflow*

Get Involved

- Contribute to the Code Base
- Contribute to Documentation

Stay Informed

- Join the Mailing List
- Read the Documentation
- Read the Knowledge Base
- Read the Blog

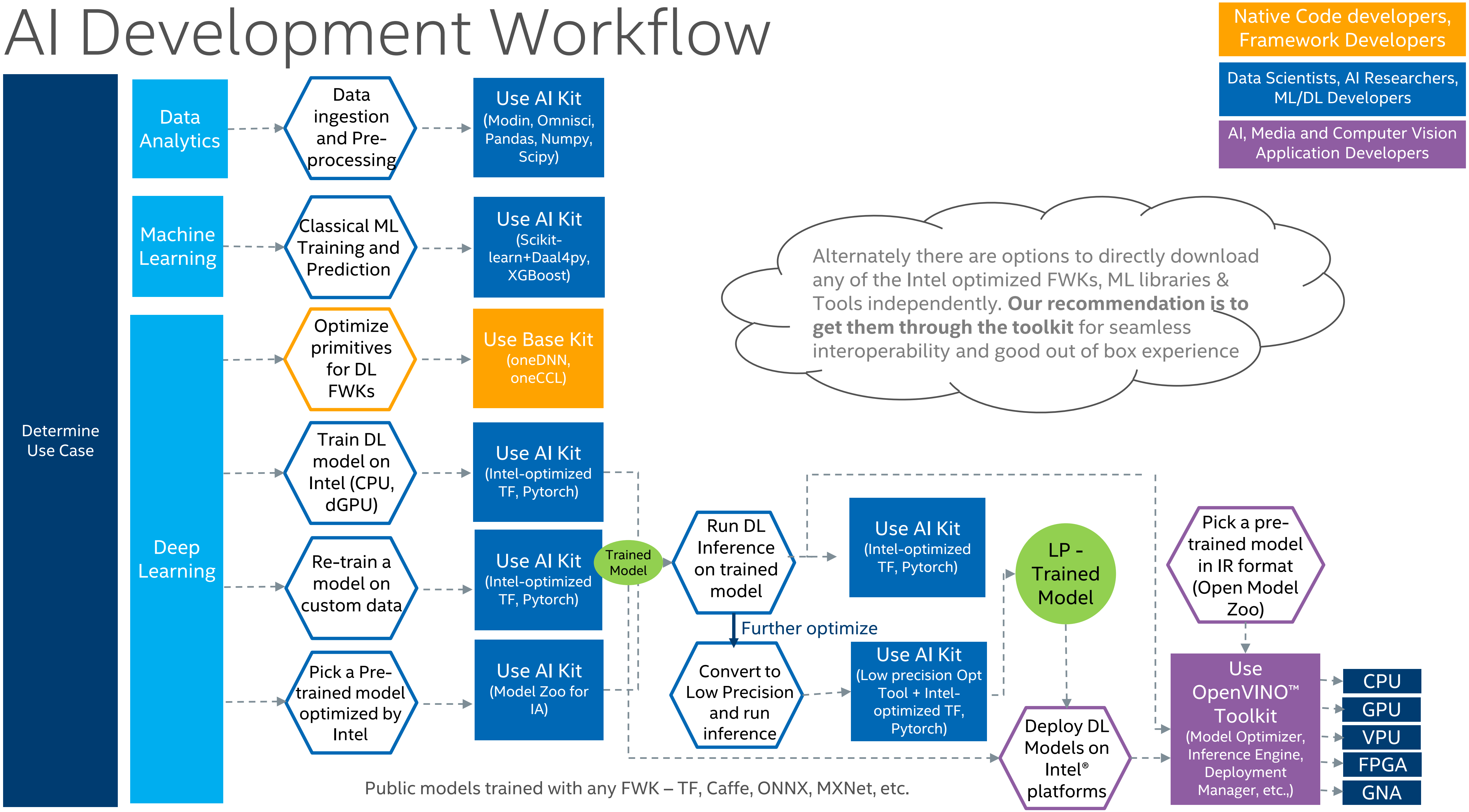
Which Toolkit should I use

Which Toolkit to Use When ?

	Intel® AI Analytics Toolkit	OpenVINO™ Toolkit
Key Value Prop	• Provide performance and easy integration across end-to-end data science pipeline for efficient AI model development • Maximum compatibility with opensource FWKs and Libs with drop-in acceleration that require minimal to no code changes • Audience: Data Scientists; AI Researchers; DL/ML Developers	• Provide leading performance and efficiency for DL inference solutions to deploy across any Intel HW (cloud to edge). • Optimized package size for deployment based on memory requirements • Audience: AI Application Developers; Media and Vision Developers
Use Cases	• Data Ingestion, Data pre-processing, ETL operations • Model training and inference • Scaling to multi-core / multi-nodes / clusters	• Inference apps for vision, Speech, Text, NLP • Media streaming / encode, decode • Scale across HW architectures – edge, cloud, datacenter, device
HW Support	• CPUs - Datacenter and Server segments – Xeons, Workstations • GPU - ATS and PVC (in future)	• CPU - Xeons, Client CPUs and Atom processors • GPU - Gen Graphics; DG1 (current), ATS, PVC (in future) • VPU - NCS & Vision Accelerator Design Products, • FPGA • GNA
Low Precision Support	**Use Intel® Low Precision Optimization Tool when using AI Analytics Toolkit** • Supports BF16 for training and FP16, Int8 and BF16 for Inference • Seamlessly integrates with Intel optimized frameworks • Available in the AI toolkit and independently	**Use Post Training Optimization Tool when using OpenVINO** • Supports FP16, Int8 and BF16 for inference • Directly works with Intermediate Representation Format • Available in the Intel Distribution of OpenVINO toolkit • Provides Training extension via NNCF for PyTorch with FP16, Int8

Exception: If a model is not supported by OpenVINO™ toolkit for Inference deployment, build custom layers for OV or fall back to the AI Analytics Toolkit and use optimized DL frameworks for inference.

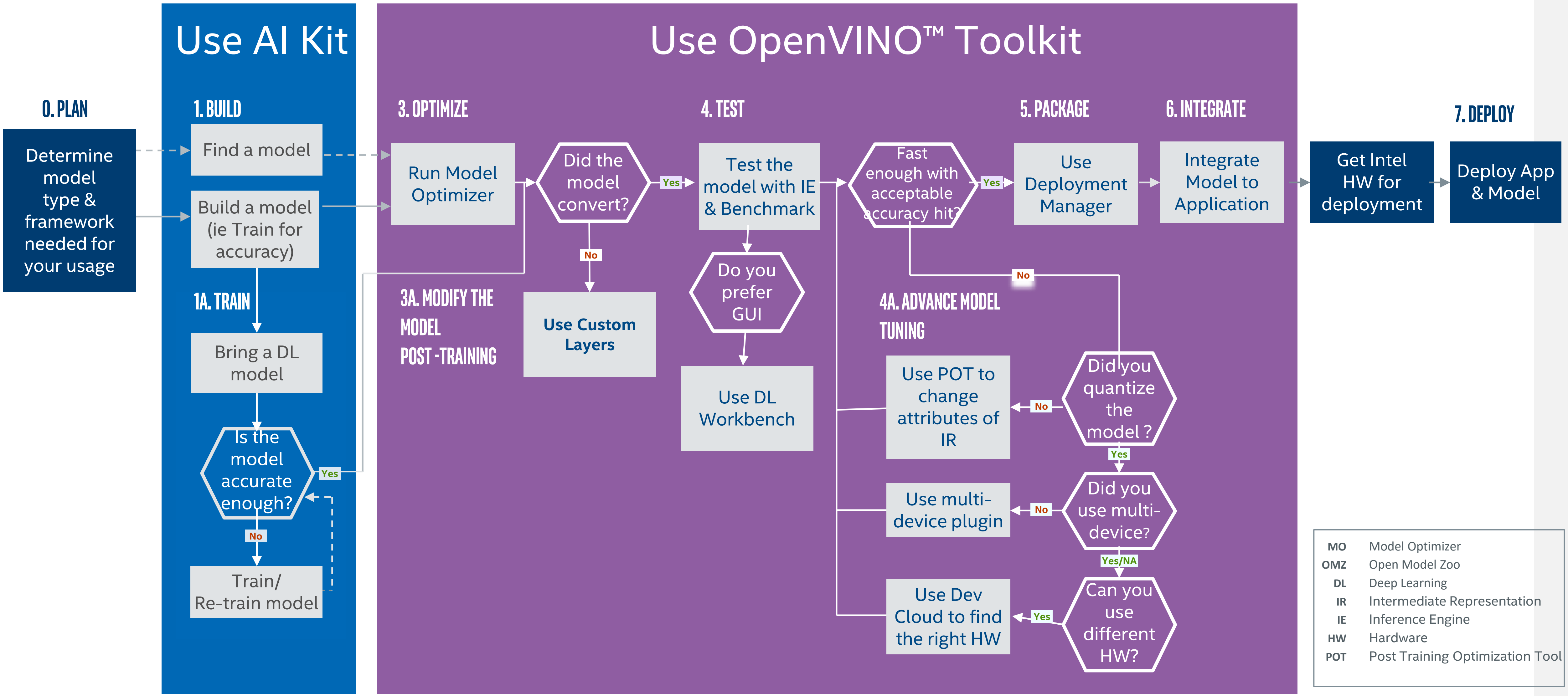
AI Development Workflow



AI Model Deployment Workflow

Data Scientists, AI Researchers, ML/DL Developers

AI, Media and Computer Vision Application Developers



MO	Model Optimizer
OMZ	Open Model Zoo
DL	Deep Learning
IR	Intermediate Representation
IE	Inference Engine
HW	Hardware
POT	Post Training Optimization Tool

A comprehensive workflow to optimize your DL model for the Intel Hardware that will be used for running inference

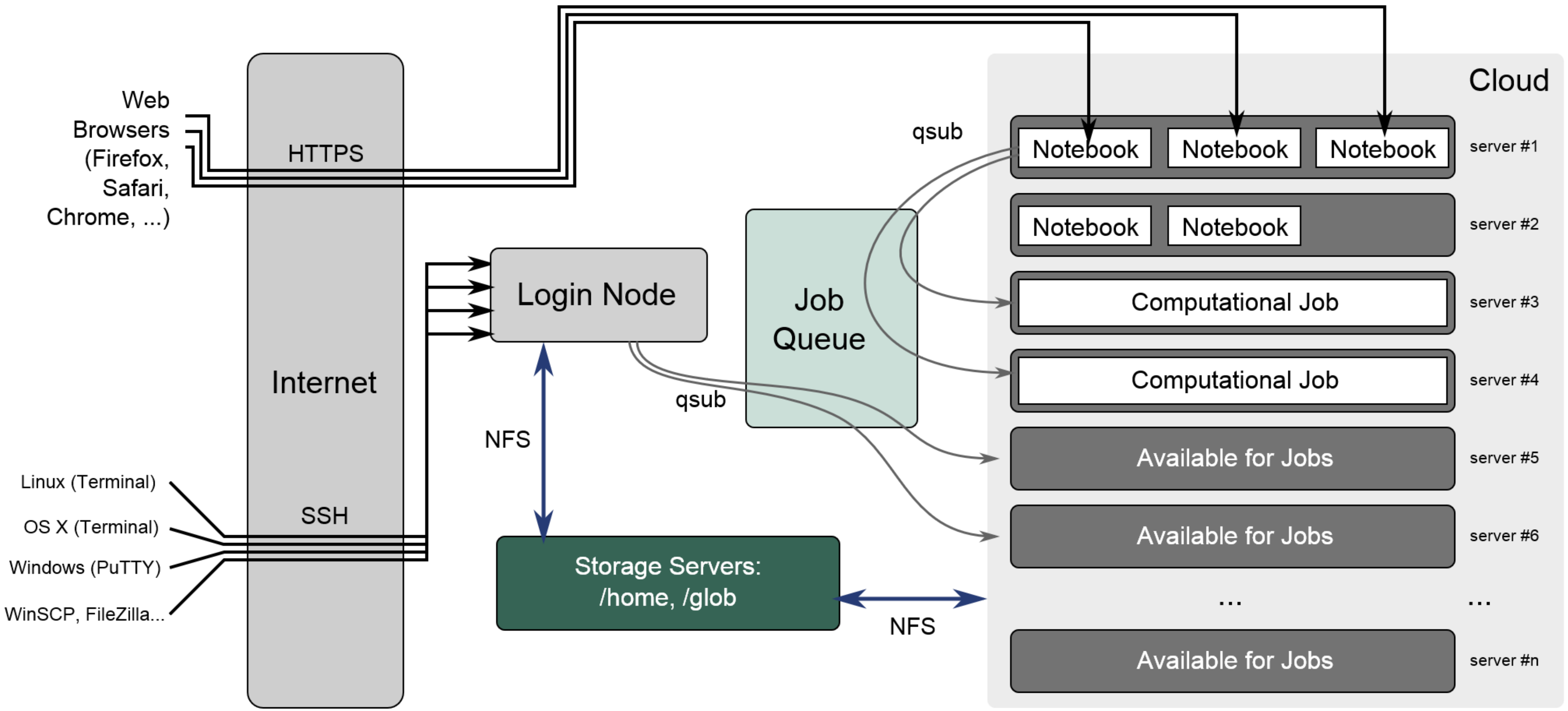
- 1) We run the demo on DC
  - TF demo
  - PyTorch demo
  - future: (ATS demo)
- Slide on how to access DevCloud
- 2) What’s behind the DC

Intel DevCloud: Getting started with oneAPI

Objectives of the External DevCloud Strategy

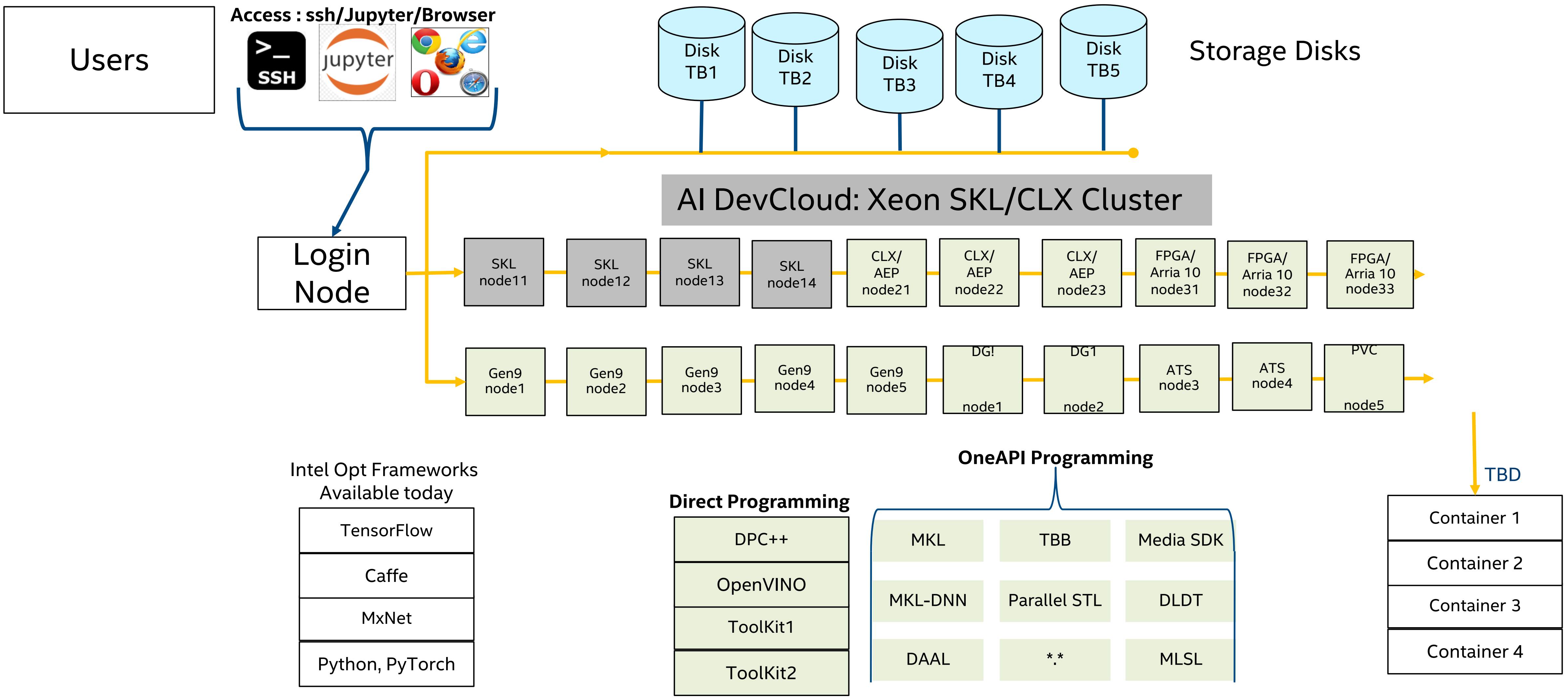
1. Demonstrate the promise of oneAPI.
2. Provide developers easy access to oneAPI h/w & s/w environment
3. Get high value feedback on oneAPI tools, libraries, language.
4. Seed research, papers, curriculum, lighthouse apps (the metrics output).
5. Support two tracks with web front end for consistent experience:
   - oneAPI production hardware/software
   - NDA SDP hardware/software

Network Arch



One API DevCloud Architecture

New Additions for One API Under NDA



Feedback Survey

Notices and Disclaimers

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.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

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

Optimization Notice

[1] 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.

[2] Software and workloads used in performance tests may have been optimized for performance only on microprocessors from Intel. 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. Consult other information and performance tests while evaluating potential purchases, including performance when combined with other products. For more information, see Performance Benchmark Test Disclosure. Source: Intel measurements, as of June 2017.

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. Check with your system manufacturer or retailer or learn more at www.intel.com.

Slide Reference	1	2	3
System Board	Intel® Server S2600 (Dual socket)	Supermicro / X11SPL-F	Supermicro / X11SPL-F
Product	Xeon Silver 4216	Intel(R) Xeon(R) Silver 4112	Intel(R) Xeon(R) Silver 4112
CPU sockets	2	-	1
Physical cores	2 x 16	4	4
Processor Base Frequency	2.10 GHz	2.60GHz	2.60GHz
HyperThreading	enabled	-	enabled
Turbo	On	-	On
Power-Performance Mode	Performance Mode	-	-
Total System Memory size	12 x 64GB	16384	16384
Memory speed	2400MHz	2400MHz	2400MHz
Software OS	Ubuntu 18.04	Ubuntu 16.04.3 LTS	Ubuntu 16.04.6 LTS
Software Kernel	4.15.0-66-generic x86_64	4.13.0-36-generic	4.15.0-29-generic
Test Date	27 September 2019	25 May 2018	18 April 2019
Precision (IntMode)	Int 8 (Throughput Mode)	FP32	Int 8 (Throughput Mode)
Power (TDP)	200W	85W	85W
Price Link on 30 Sep 2019 (Prices may vary)	$2,024	$483	$483
Network	Mobilenet SSD	Mobilenet SSD	Mobilenet SSD

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. Check with your system manufacturer or retailer or learn more at www.intel.com.

Intel, the Intel logo, Xeon™, Arria™ and Movidius™ are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

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

System Board	Intel prototype, TGL U DDR4 SODIMM RVP	ASUSTeK COMPUTER INC. / PRIME Z370-A
CPU	11th Gen Intel® Core™ -5-1145G7E @ 2.6 GHz.	8th Gen Intel ® Core™ i5-8500T @ 3.0 GHz.
Sockets / Physical cores	1 / 4	1 / 6
HyperThreading / Turbo Setting	Enabled / On	Na / On
Memory	2 x 8198 MB 3200 MT/s DDR4	2 x 16384 MB 2667 MT/s DDR4
OS	Ubuntu* 18.04 LTS	Ubuntu* 18.04 LTS
Kernel	5.8.0-050800-generic	5.3.0-24-generic
Software	Intel® Distribution of OpenVINO™ toolkit 2021.1.075	Intel® Distribution of OpenVINO™ toolkit 2021.1.075
BIOS	Intel TGLIFUI1.R00.3243.A04.2006302148	AMI, version 2401
BIOS release date	Release Date: 06/30/2021	7/12/2019
BIOS Setting	Load default settings	Load default settings, set XMP to 2667
Test Date	9/9/2021	9/9/2021
Precision and Batch Size	CPU: INT8, GPU: FP16-INT8, batch size: 1	CPU: INT8, GPU: FP16-INT8, batch size: 1
Number of Inference Requests	4	6
Number of Execution Streams	4	6
Power (TDP Link)	28 W	35W
Price (USD) Link on Sep 22,2021 Prices may vary	$309	$192

1): Memory is installed such that all primary memory slots are populated.
2): Testing by Intel as of September 9, 2021