Choose the Best Accelerated Technology

Intel Performance optimizations for Deep Learning

Shailen Sobhee – Deep Learning Engineer
shailen.sobhee@intel.com
15 June 2021
Quick recap of oneAPI
Overview of oneDNN

Training:
- Overview of performance-optimized DL frameworks
  - Tensorflow
  - PyTorch

Inferencing:
- Intel® Low Precision Optimization Tool
- Primer to Intel® Distribution of OpenVINO™
  - (Deeper presentation in next session)
Intel’s oneAPI Ecosystem

Built on Intel’s Rich Heritage of CPU Tools Expanded to XPU’s

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.

Application Workloads Need Diverse Hardware

Middleware & Frameworks (Powered by oneAPI)

TensorFlow  PyTorch  MODIN  Modin  NumPy  XeonPhy  @openVINO

Intel® oneAPI Product

Compatibility Tool  Languages  Libraries  Analysis & Debug Tools

Low-Level Hardware Interface

XPU’s

CPU  GPU  FPGA  Other accelerators

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

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
Develop Fast Neural Networks on Intel® CPUs & GPUs
with Performance-optimized Building Blocks

Intel® oneAPI Deep Neural Network Library (oneDNN)
Intel® oneAPI Deep Neural Network Library (oneDNN)

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

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

More information as well as sources:

https://github.com/oneapi-src/oneDNN
Intel® oneAPI Deep Neural Network Library

Basic Information

- **Features**
  - API: C, C++, SYCL
  - **Training**: float32, bfloat16
  - **Inference**: float32, bfloat16, float16, and int8
  - 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
    - Runtimes: OpenCL, DPC++

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

(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.

Examples \rightarrow MatMul \rightarrow Add \rightarrow ReLU
Weights \rightarrow MatMul
Bias
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.

<table>
<thead>
<tr>
<th>Forward</th>
<th>Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv2D</td>
<td>Conv2DGrad</td>
</tr>
<tr>
<td>Relu, TanH, ELU</td>
<td>ReLUGrad, TanHGrad, ELUGrad</td>
</tr>
<tr>
<td>MaxPooling</td>
<td>MaxPoolingGrad</td>
</tr>
<tr>
<td>AvgPooling</td>
<td>AvgPoolingGrad</td>
</tr>
<tr>
<td>BatchNorm</td>
<td>BatchNormGrad</td>
</tr>
<tr>
<td>LRN</td>
<td>LRNGrad</td>
</tr>
<tr>
<td>MatMul, Concat</td>
<td></td>
</tr>
</tbody>
</table>
Fusing computations

- On Intel processors a high percentage 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](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

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

More details: [https://oneapi-src.github.io/oneDNN/understanding_memory_formats.html](https://oneapi-src.github.io/oneDNN/understanding_memory_formats.html)
System optimizations: load balancing

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


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

```python
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.

oneDNN <-> Frameworks interaction

**TensorFlow**
- Graph Optimizer
- Op implementations
  - impl_load_data
  - impl_matmul
  - impl_tanh

**oneDNN**
- tf_model.py
  - load_data
  - matmul
  - tanh
  - print_result

- class matmul {
  matmul(bool tanh_post);
  execute(sycl::queue q, sycl::buffer A, sycl::buffer B, sycl::buffer C);
};

- All parameters are specified at creation time, so that oneDNN generate the most optimized kernel(s).

---

call impl_load_data
call onednn_gc_matmul
call onednn_gc_matmul, po=tanh
call impl_print_result
**OpenCL API is not available as part of Intel oneAPI binary distribution**

Dispatching between CPU and GPU is based on the kind of device associated with the DPC++ queue.

All GPU kernels are compiled in runtime. CM and nGEN support is not available publicly yet. Adding/migrating to DPC++ kernels is under consideration.

OpenCL GPU RT is always needed to compile OpenCL C and CM kernels.

In case of DPC++ and L0, binary kernels need to be wrapped to L0 modules to create SYCL kernels eventually.

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
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
License - Apache 2.0

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 ipex
import torch
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, reductions='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()
Data types

- **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

Speedup Ratio (Higher is better)

- **ResNet50**
- **ResNeXt-3D**
- **BERT**

- **PyTorch**
- **Operator Injection**
- **Operator Injection + Mix Precision**
- **Operator Injection + Mix Precision + JIT**
### Inference with IPEX for ResNet50

#### Worker11 (CPX)

```
LD_PRELOAD=/root/anaconda3/lib/libomp5.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**
- 96.1924
- 32-bit

**Integer**
- 96
- 8-bit

```
10110110
10110110
10110110
10110110
```

```
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:

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

<table>
<thead>
<tr>
<th>TensorFlow Model</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet50 V1</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNet50 V1.5</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNet101</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>Inception V1</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>Inception V2</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>Inception V3</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>Inception V4</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNetV2_50</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNetV2_101</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNetV2_152</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>Inception ResNet V2</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>SSD ResNet50 V1</td>
<td>Object Detection</td>
</tr>
<tr>
<td>Wide &amp; Deep</td>
<td>Recommendation</td>
</tr>
<tr>
<td>VGG16</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>VGG19</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>Style transfer</td>
<td>Style Transfer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PyTorch Model</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT-Large RTE</td>
<td>Language Translation</td>
</tr>
<tr>
<td>BERT-Large QNLI</td>
<td>Language Translation</td>
</tr>
<tr>
<td>BERT-Large CoLA</td>
<td>Language Translation</td>
</tr>
<tr>
<td>BERT-Base SST-Z</td>
<td>Language Translation</td>
</tr>
<tr>
<td>BERT-Base SST-B</td>
<td>Language Translation</td>
</tr>
<tr>
<td>BERT-Base CoLA</td>
<td>Language Translation</td>
</tr>
<tr>
<td>BERT-Base MRPC</td>
<td>Language Translation</td>
</tr>
<tr>
<td>DLRM</td>
<td>Recommendation</td>
</tr>
<tr>
<td>BERT-Large MRPC</td>
<td>Language Translation</td>
</tr>
<tr>
<td>ResNet50 V1.5</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>BERT-Large SQUAD</td>
<td>Language Translation</td>
</tr>
<tr>
<td>ResNet50 V1.5 QAT</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNet18</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>Inception V3</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>YOLO V3</td>
<td>Object Detection</td>
</tr>
<tr>
<td>Peleenet</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNest50</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>SE_ResNet50_32x4d</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNet50 V1.5 QAT</td>
<td>Image Recognition</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MxNet Model</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet50 V1</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>MobileNet V1</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>MobileNet V2</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>SSD-ResNet50</td>
<td>Object Detection</td>
</tr>
<tr>
<td>SqueezeNet V1</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNet18</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>Inception V3</td>
<td>Image Recognition</td>
</tr>
</tbody>
</table>
Auto-tuning Flow

Tunable Configurations
- Quantization
  - Quantizer
    - FP32 Model
    - Low-precision Model
  - Evaluator (Accuracy metrics, Performance etc.)

Next Config
Tuning Strategy
Optimal Solution

Model Inspect
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.

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

- Simple API is easy to integrated in original training/inference script.

YAML

- Common functions are integrated and controlled by parameters;
- Templates are easy to refer;
- Lots of advance parameters provide powerful tuning capability.

FP32 model

YAML file (template-based)

Launcher code based on API

Training/Inference script

Dataset

INT8 BF16 model

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.

```python
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

```yaml
# 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

**Model**:

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

**Quantization**:

```yaml
quantization:
calibration:
sampling_size: 50, 100
data_loader:
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**:

```yaml
evaluation:
  accuracy:
    metric:
      topk: 1
data_loader:
  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**:

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

**YAML config**

**Code change**

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

**Full example**: [https://github.com/intel/lpot/tree/master/examples/tensorflow/image_recognition](https://github.com/intel/lpot/tree/master/examples/tensorflow/image_recognition)
Demo

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

- **Intel LPOT Sample for Tensorflow:**
  - [https://github.com/oneapi-src/oneAPI-samples/tree/master/Al-and-Analytics/Getting-Started-Samples/LPOT-Sample-for-Tensorflow](https://github.com/oneapi-src/oneAPI-samples/tree/master/Al-and-Analytics/Getting-Started-Samples/LPOT-Sample-for-Tensorflow)
Infrastructure

FP32 Model

User Configurations
- Calib/QAT Configuration
- Quantization Configuration
- Tuning Configuration

Quantized Model

Transforms
- Auto-tuner
  - Calibrate
  - Quant-Aware Train
  - Quantize
  - Metrics
  - Model Inspect

Framework Adaptation Layer
- TF Adaptation Layer
- PyTorch Adaptation Layer
- MXNet Adaptation Layer

Auto-tuning
- Extensible Tuning Strategies
  - Auto-tuner
  - Extension API

Supported
- TF
- PyTorch
- MXNet

Not Yet Supported
- OpenVINO
- ONNX

Legend
- Supported
- Not Yet Supported
Working Flow

Tunable Configurations
- Calibration/QAT Configuration
- Quantization Configuration

Framework Quantization Flow
- FP32 Model
  - Calibrate or QAT
    - Calibraton/QAT Dataset
    - Post-Training Quant or QAT
    - Dynamic Quant
- Quantize
- Quantized Model

Evaluation Dataset

Metrics Evaluator

Next Config

Optimal Solution
(Conf* -> Perf*)

Tuning Strategy

Stop Criteria Met

Performance and Accuracy

Model Inspect
Product Overview

Intel® Distribution of OpenVINO™ toolkit

2019 Developer Tool of the Year
Awarded by the Edge AI and Vision Alliance
WRITE once, deploy & scale diversely

TensorFlow
ONNX
mxnet
KALDI
Caffe

Model Optimizer
Inference Engine

CPU
FPGA
Edge
GPU

*Other names and brands may be claimed as the property of others.
From a bird’s eye-view
Advanced capabilities to streamline deep learning deployments

1. Build

Trained Model
- TensorFlow
- Caffe
- Kaldi
- mxnet
- DNNX

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

Intermediate Representation
(xml, .bin)

IR Data
Read, Load, Infer

Model Optimizer

Deep Learning Optimizer Tool

Post-Training Optimization Tool

Deep Learning Workbench

Deep Learning Streamer

OpenCV

OpenCL™

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

Inference Engine
Common API that abstracts low-level programming for each hardware

CPU Plugin

GPU Plugin

GNA Plugin

Myriad Plugin

HDDL Plugin

FGPA Plugin

Deployment Manager

3. Deploy

HDDL Plugin

FPGA Plugin

Xeon

Atom

Core

Deep Learning Streamer

Code Samples & Demos

OpenCV

OpenCL™

Deployment Manager

IAGS Intel Architecture, Graphics, and Software
Get Started

Typical workflow from development to deployment

1. Train a model
2. Find a trained model
3. Run the Model Optimizer
4. Intermediate Representation (.bin, .xml)
5. Deploy using the Inference Engine
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* ➔

**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* ➔
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

Applications

Inference Engine (Common API)

Multi-device plugin (optional but recommended - for full system utilization)

mkl-dnn & oneDNN plugin
cIDNN & oneDNN plugin
GNA plugin
Myriad & HDDL plugins
FPGA plugin

Intrinsics
OpenCL™
GNA API
Movidius API
DLA
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 information 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

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

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

- 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

- 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

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

- 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

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

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

Use free Pre-training 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
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

Healthcare Access and Quality

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.

Security Against Social and Digital Attacks

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.

Sewer pipe inspection analysis

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.

Frictionless retail checkout

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.
## Resources and Community Support

Vibrant community of developers, enterprises and skills builders

### QUALIFY

- Use a trained model and **check** if framework is supported
  - or –
- Take advantage of a pre-trained model from the Open Model Zoo

### INSTALLATION

- Download the Intel® OpenVINO™ toolkit package from Intel® Developer Zone, or by YUM or APT repositories
- Utilize the Getting Started Guide

### PREPARE

- 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

### HANDS ON

- 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

### SUPPORT

- 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

<table>
<thead>
<tr>
<th>Tool/Component</th>
<th>Intel® Distribution of OpenVINO™ toolkit</th>
<th>OpenVINO™ toolkit (open source)</th>
<th>Open Source Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installer (including necessary drivers)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model Optimizer</td>
<td>✔</td>
<td></td>
<td><a href="https://github.com/openvinotoolkit/openvino/tree/master/model-optimizer">https://github.com/openvinotoolkit/openvino/tree/master/model-optimizer</a></td>
</tr>
<tr>
<td>Inference Engine - Core</td>
<td>✔</td>
<td></td>
<td><a href="https://github.com/openvinotoolkit/openvino/tree/master/inference-engine">https://github.com/openvinotoolkit/openvino/tree/master/inference-engine</a></td>
</tr>
<tr>
<td>Intel CPU plug-in</td>
<td></td>
<td>✔ Intel® Math Kernel Library (Intel® MKL) only¹</td>
<td><a href="https://github.com/openvinotoolkit/openvino/tree/master/inference-engine">https://github.com/openvinotoolkit/openvino/tree/master/inference-engine</a></td>
</tr>
<tr>
<td>Intel GPU (Intel® Processor Graphics) plug-in</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heterogeneous plug-in</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel GNA plug-in</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel® FPGA plug-in</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel® Neural Compute Stick (1 &amp; 2) VPU plug-in</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel® Vision Accelerator based on Movidius plug-in</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi-device &amp; hetero plug-ins</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Samples (APIs)</td>
<td>✔</td>
<td></td>
<td><a href="https://github.com/openvinotoolkit/openvino/tree/master/inference-engine">https://github.com/openvinotoolkit/openvino/tree/master/inference-engine</a></td>
</tr>
<tr>
<td>Demos</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Traditional Computer Vision</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenCV*</td>
<td>✔</td>
<td></td>
<td><a href="https://github.com/opencv/opencv">https://github.com/opencv/opencv</a></td>
</tr>
<tr>
<td>Intel® Media SDK</td>
<td>✔</td>
<td></td>
<td><a href="https://github.com/Intel-Media-SDK/IntelMediaSDK">https://github.com/Intel-Media-SDK/IntelMediaSDK</a></td>
</tr>
<tr>
<td>OpenCL™ Drivers &amp; Runtimes</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FPGA Runtime Environment, Deep Learning Acceleration &amp; Bitstreams (Linux* only)</td>
<td>✔</td>
<td></td>
<td><a href="https://github.com/intel/compute-runtime">https://github.com/intel/compute-runtime</a></td>
</tr>
</tbody>
</table>
# System Requirements

## Intel® Platforms

<table>
<thead>
<tr>
<th>CPU</th>
<th>Compatabile Operating Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ 6th-10th generation Intel® Core™ and Xeon® processors</td>
<td>▪ Ubuntu* 18.04.3 LTS (64 bit)</td>
</tr>
<tr>
<td>▪ 1st and 2nd generation Intel® Xeon® Scalable processors</td>
<td>▪ Microsoft Windows* 10 (64 bit)</td>
</tr>
<tr>
<td>▪ Intel® Pentium® processor N4200/5, N3350/5, N3450/5 with Intel® HD Graphics</td>
<td>▪ CentOS* 7.4 (64 bit)</td>
</tr>
<tr>
<td>▪ Intel® Xeon® processor with Intel® Iris® Pro Graphics &amp; Intel® HD Graphics (excluding E5 product family, which does not have graphics)</td>
<td>▪ macOS* 10.13 &amp; 10.14 (64 bit)</td>
</tr>
</tbody>
</table>

### Iris® Pro & Intel® HD Graphics

|▪ 6th-10th generation Intel® Core™ processor with Intel® Iris® Pro graphics & Intel® HD Graphics |▪ Ubuntu 18.04.3 LTS (64 bit) |
|▪ Intel® Xeon® processor with Intel® Iris® Pro Graphics & Intel® HD Graphics (excluding E5 product family, which does not have graphics) |▪ Windows 10 (64 bit) |
|▪ Ubuntu 18.04.2 LTS (64 bit) |▪ CentOS 7.4 (64 bit) |
|▪ Ubuntu 18.04.3 LTS (64 bit) |▪ macOS* 10.13 & 10.14 (64 bit) |
|▪ Ubuntu 18.04.2 LTS (64 bit) |▪ Raspbian (target only) |

### FPGA

|▪ Intel® Arria® FPGA 10 GX development kit |▪ Ubuntu 18.04.3 LTS (64 bit) |
|▪ Intel® Programmable Acceleration Card with Intel® Arria® 10 GX FPGA operating systems |▪ CentOS 7.4 (64 bit) |
|▪ OpenCV® & OpenVX® functions must be run against the CPU or Intel® Processor Graphics (GPU) |▪ Windows 10 (64 bit) |

### VPU: Intel® Movidius™ Neural Compute Stick; Intel® Neural Compute Stick2

### Intel® Vision Accelerator Design Products

|▪ Intel® Vision Accelerator Design with Intel® Arria10 FPGA |▪ Ubuntu 18.04.3 LTS (64 bit) |
|▪ Intel® Vision Accelerator Design with Intel® Movidius™ VPUs |▪ CentOS 7.4 (64 bit) |

### Development Platforms

|▪ 6th-10th generation Intel® Core™ and Intel® Xeon® processors |▪ Ubuntu 18.04.3 LTS (64 bit) |
|▪ 1st and 2nd generation Intel® Xeon® Scalable processors |▪ Windows 10 (64 bit) |

## Additional Software Requirements

### Linux* build environment required components

- OpenCV 3.4 or higher
- CMake* 2.8 or higher
- GNU Compiler Collection (GCC) 3.4 or higher
- Python* 3.4 or higher

### Microsoft Windows* build environment required components

- Intel® HD Graphics Driver (latest version)
- Intel® C++ Compiler 2017 Update 4
- OpenCV 3.4 or higher
- CMake 2.8 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?

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

Determine Use Case

Data Analytics
- Data ingestion and Pre-processing
  - Use AI Kit (Modin, Omnisci, Pandas, NumPy, Scipy)

Machine Learning
- Classical ML Training and Prediction
  - Use AI Kit (SciKit-learn+Daal4py, XGBoost)
- Optimize primitives for DL FWKs
  - Use Base Kit (oneDNN, oneCCL)
- Train DL model on Intel (CPU, dGPU)
  - Use AI Kit (Intel-optimized TF, Pytorch)
- Re-train a model on custom data
  - Use AI Kit (Intel-optimized TF, Pytorch)
- Pick a Pre-trained model optimized by Intel
  - Use AI Kit (Model Zoo for IA)

Deep Learning

Run DL Inference on trained model
- Further optimize
  - Use AI Kit (Intel-optimized TF, Pytorch)
- Convert to Low Precision and run inference
  - Use AI Kit (Low precision Opt Tool + Intel-optimized TF, Pytorch)
- Trained Model

Deploy DL Models on Intel® platforms
- Use OpenVINO™ Toolkit (Model Optimizer, Inference Engine, Deployment Manager, etc.)

Pick a pre-trained model in IR format (Open Model Zoo)

Alternately there are options to directly download any of the Intel optimized FWKs, ML libraries & Tools independently. Our recommendation is to get them through the toolkit for seamless interoperability and good out of box experience.

Public models trained with any FWK – TF, Caffe, ONNX, MXNet, etc.
1) We run the demo on DC
   • TF demo
   • PyTorch demo
   • future: (ATS demo)

2) What’s behind the DC

Slide on how to access DevCloud
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

Users

Access: ssh/Jupyter/Browser

Login Node

AI DevCloud: Xeon SKL/CLX Cluster

Storage Disks

Intel Opt Frameworks Available today
- TensorFlow
- Caffe
- MxNet
- Python, PyTorch

Direct Programming
- DPC++
- OpenVINO
- ToolKit1
- ToolKit2

OneAPI Programming
- MKL
- TBB
- Media SDK
- MKL-DNN
- Parallel STL
- DLDT
- DAAL
- **
- MLSL

New Additions for
One API Under NDA

Containers
- Container 1
- Container 2
- Container 3
- Container 4
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.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

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](http://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.

© Intel Corporation.

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

© Intel Corporation.

<table>
<thead>
<tr>
<th>System Board</th>
<th>Intel prototype, TGL U DDR4 SODIMM RVP</th>
<th>ASUSTeK COMPUTER INC. / PRIME Z370-A</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>11th Gen Intel® Core™ -5-1145G7E @ 2.6 GHz.</td>
<td>8th Gen Intel ® Core™ i5-8500T @ 3.0 GHz.</td>
</tr>
<tr>
<td>Sockets / Physical cores</td>
<td>1 / 4</td>
<td>1 / 6</td>
</tr>
<tr>
<td>HyperThreading / Turbo Setting</td>
<td>Enabled / On</td>
<td>Na / On</td>
</tr>
<tr>
<td>Memory</td>
<td>2 x 8198 MB 3200 MT/s DDR4</td>
<td>2 x 16384 MB 2667 MT/s DDR4</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu* 18.04 LTS</td>
<td>Ubuntu* 18.04 LTS</td>
</tr>
<tr>
<td>Kernel</td>
<td>5.8.0-050800-generic</td>
<td>5.3.0-24-generic</td>
</tr>
<tr>
<td>Software</td>
<td>Intel® Distribution of OpenVINO™ toolkit 2021.1.075</td>
<td>Intel® Distribution of OpenVINO™ toolkit 2021.1.075</td>
</tr>
<tr>
<td>BIOS</td>
<td>Intel TGLIFU11.R00.3243.A04.2006302148</td>
<td>AMI, version 2401</td>
</tr>
<tr>
<td>BIOS release date</td>
<td>Release Date: 06/30/2021</td>
<td>7/12/2019</td>
</tr>
<tr>
<td>BIOS Setting</td>
<td>Load default settings</td>
<td>Load default settings, set XMP to 2667</td>
</tr>
<tr>
<td>Test Date</td>
<td>9/9/2021</td>
<td>9/9/2021</td>
</tr>
<tr>
<td>Precision and Batch Size</td>
<td>CPU: INT8, GPU: FP16-INT8, batch size: 1</td>
<td>CPU: INT8, GPU: FP16-INT8, batch size: 1</td>
</tr>
<tr>
<td>Number of Inference Requests</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Number of Execution Streams</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Power (TDP Link)</td>
<td>28 W</td>
<td>35W</td>
</tr>
<tr>
<td>Price (USD) Link on Sep 22,2021</td>
<td>$309</td>
<td>$192</td>
</tr>
</tbody>
</table>

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