Choose the Best Accelerated Technology

Intel Performance optimizations for Deep Learning

Shailen Sobhee – Deep Learning Engineer
shailen.sobhee@intel.com
9 April 2021

Agenda

- Quick recap of oneAPI
- Overview of oneDNN
- Training:
  - Overview of performance-optimized DL frameworks
    - Tensorflow
    - PyTorch
- Inferencing:
  - Intel® 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

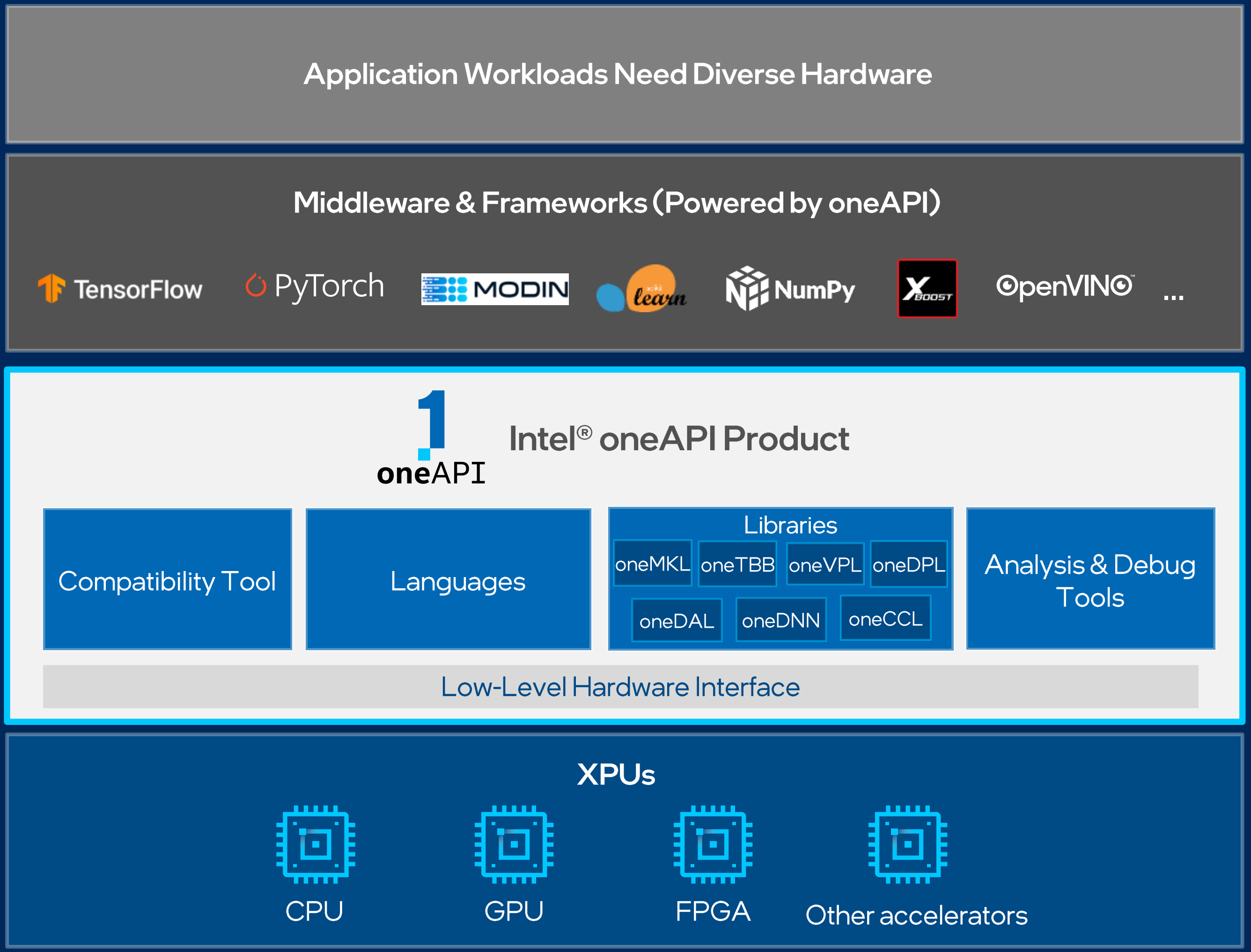
A cross-architecture language based on C++ and SYCL standards

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

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

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

Intel® 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.

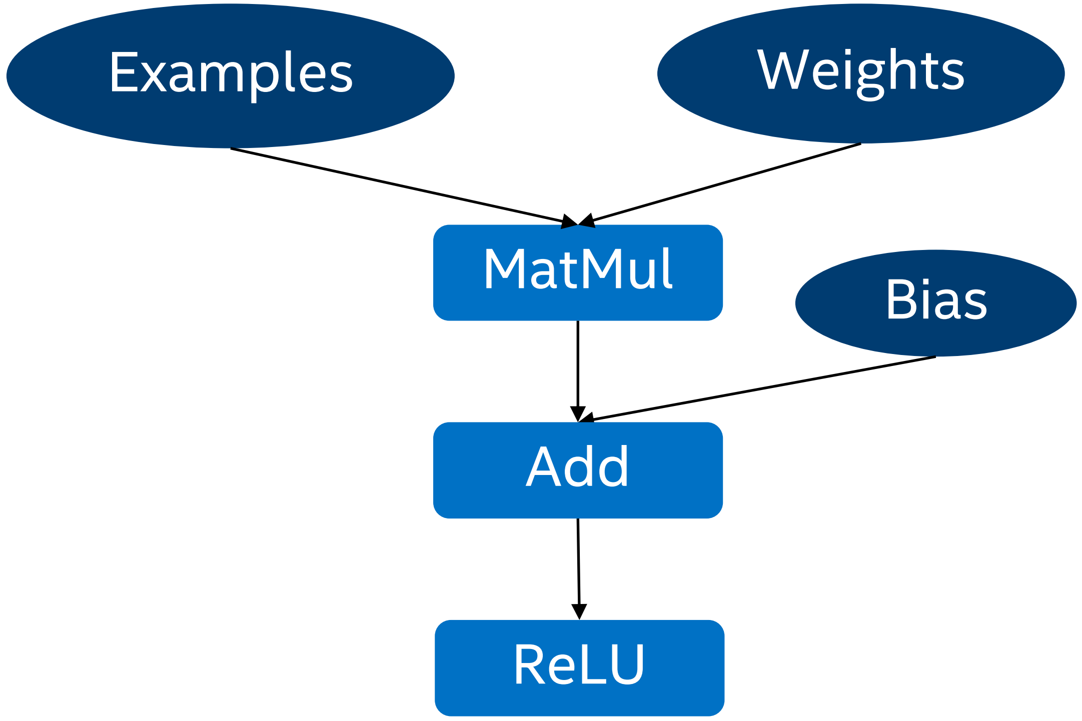
Intel® TensorFlow* optimizations

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.

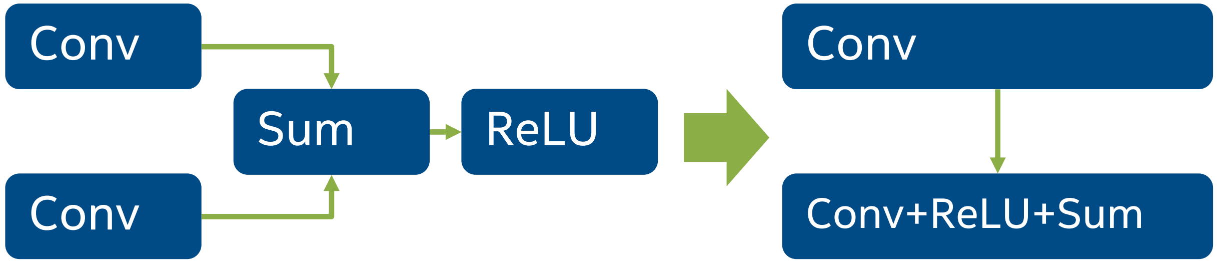


Operator optimizations

- 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.

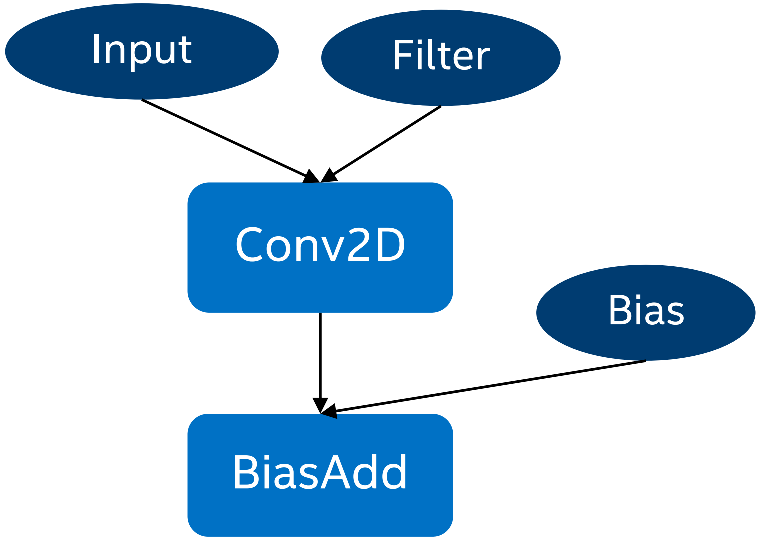
Forward	Backward
Conv2D	Conv2DGrad
Relu, TanH, ELU	ReLUGrad, TanHGrad, ELUGrad
MaxPooling	MaxPoolingGrad
AvgPooling	AvgPoolingGrad
BatchNorm	BatchNormGrad
LRN	LRNGrad
MatMul, Concat	

Fusing computations

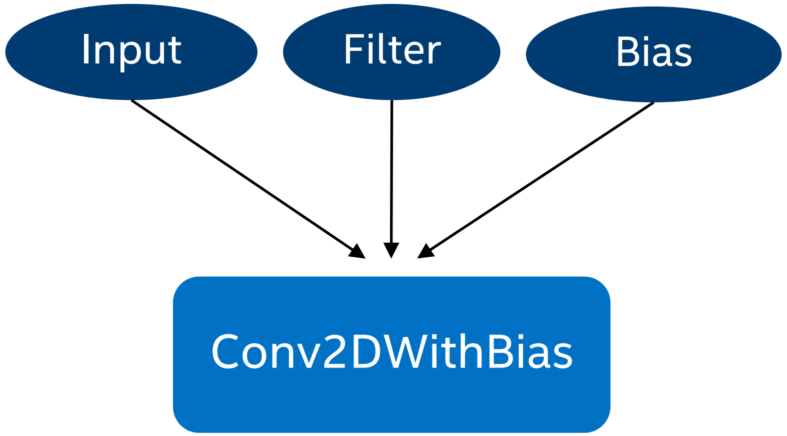


- On Intel processors a high % of time is typically spent in BW-limited ops
  - ~40% of ResNet-50, even higher for inference
- The solution is to fuse BW-limited ops with convolutions or one with another to reduce the # of memory accesses
  - Conv+ReLU+Sum, BatchNorm+ReLU, etc
- The frameworks are expected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

Graph optimizations: fusion

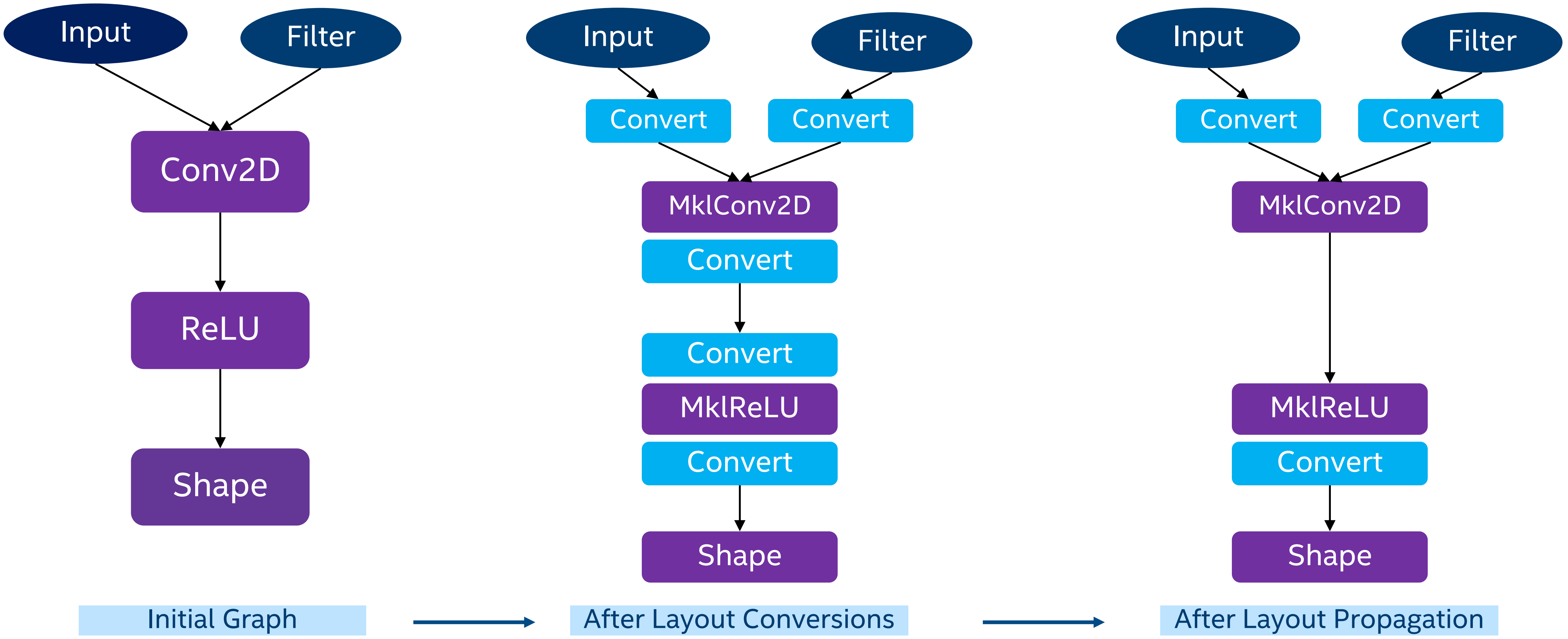


Before Merge



After Merge

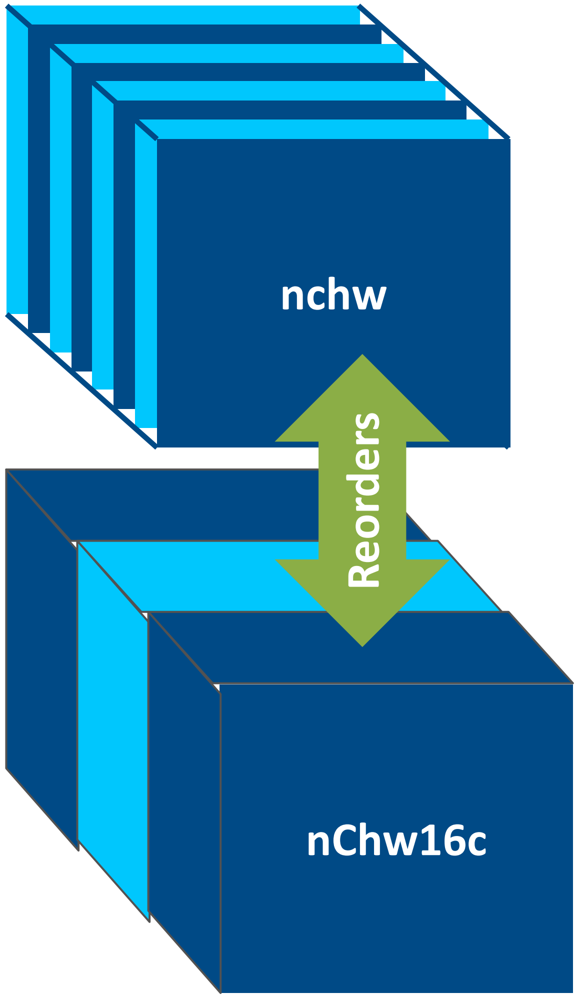
Graph optimizations: layout propagation



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

More on memory channels: Memory layouts

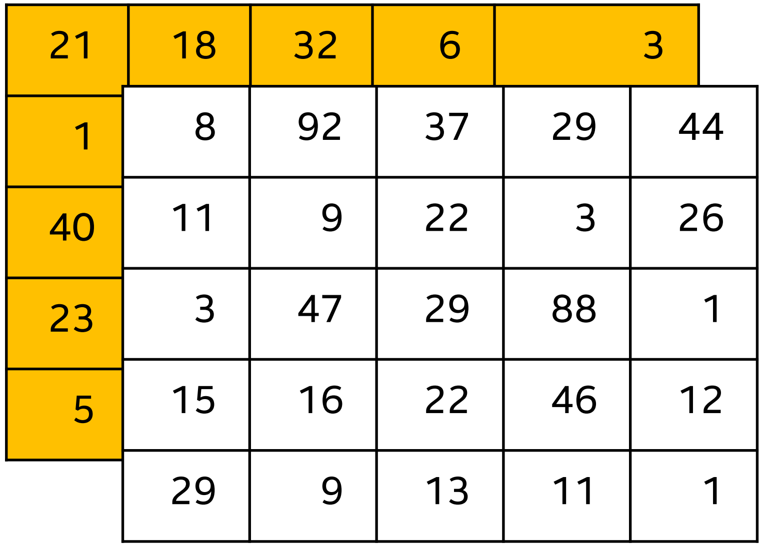
- Most popular memory layouts for image recognition are **nhwc** and **nchw**
  - Challenging for Intel processors either for vectorization or for memory accesses (cache thrashing)
- Intel oneDNN convolutions use blocked layouts
  - Example: **nhwc** with channels blocked by 16 – **nChw16c**
  - Convolutions define which layouts are to be used by other primitives
  - Optimized frameworks track memory layouts and perform reorders **only** when necessary

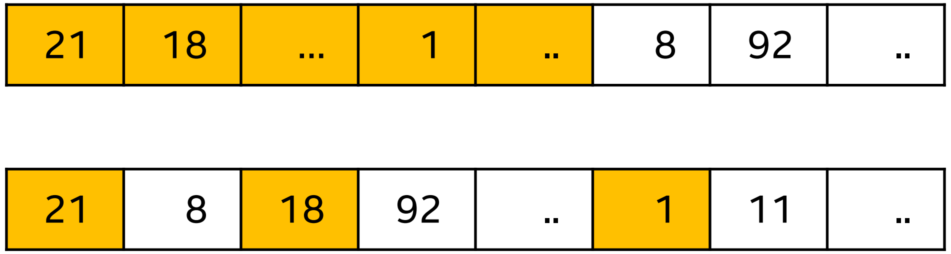


More details: https://oneapi-src.github.io/oneDNN/understanding_memory_formats.html

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





Channel based (NCHW)

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

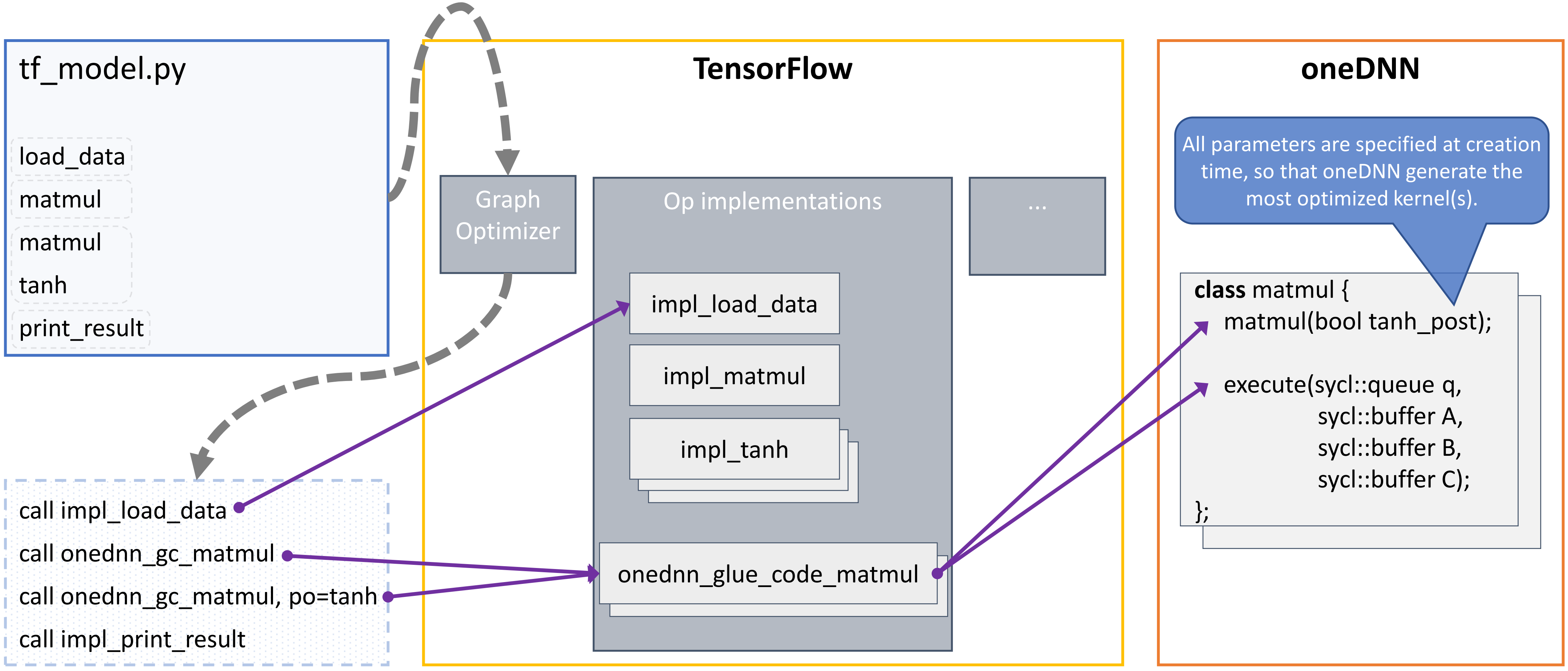
This section details the different configurations and environment variables that can be used to tune the MKL to get optimal performance. Before tweaking various environment variables make sure the model is using the `NCHW` (`channels_first`) data format. The MKL is optimized for `NCHW` and Intel is working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

MKL uses the following environment variables to tune performance:

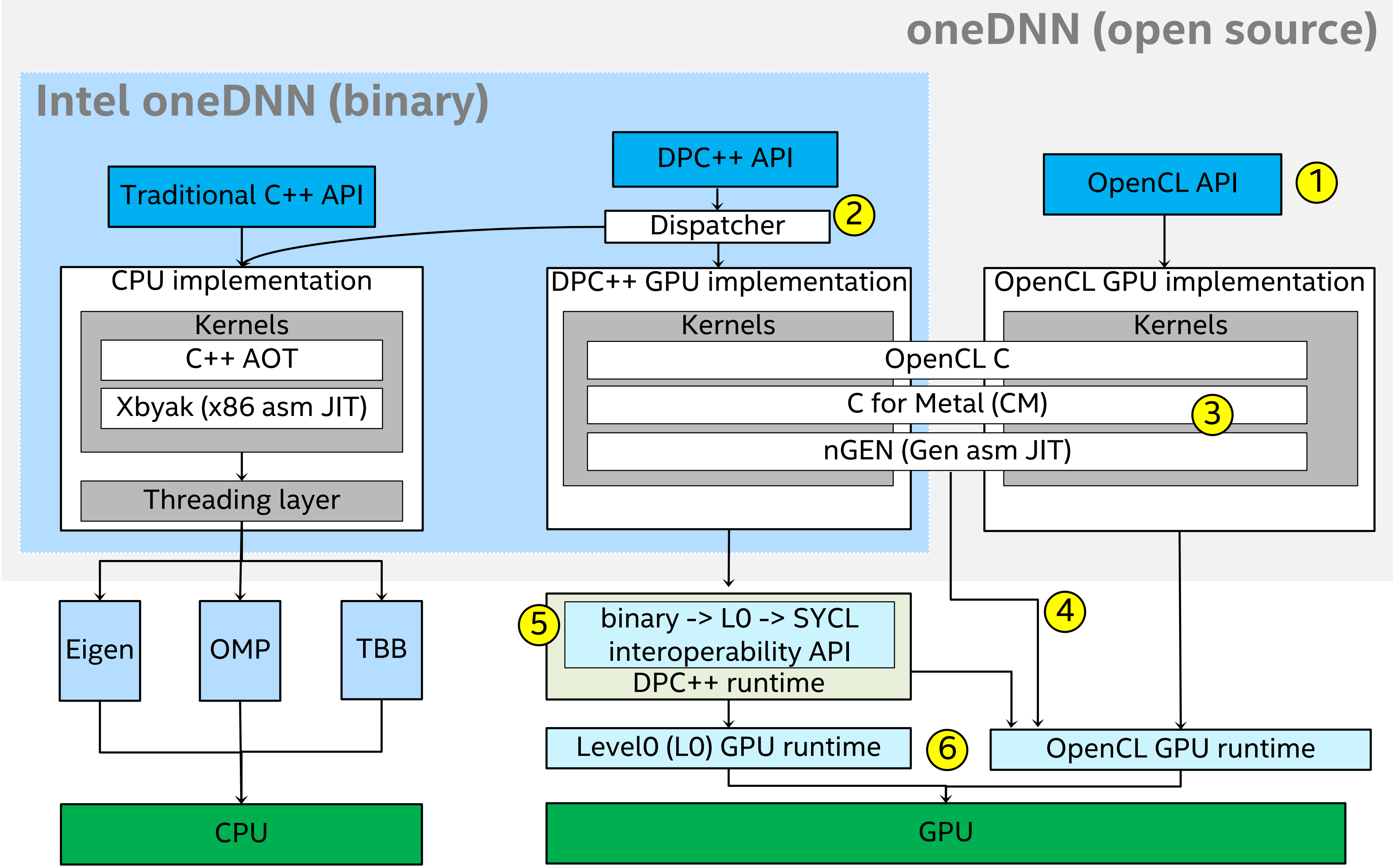
- KMP_BLOCKTIME - Sets the time, in milliseconds, that a thread should wait, after completing the execution of a parallel region, before sleeping.
- KMP_AFFINITY - Enables the run-time library to bind threads to physical processing units.
- KMP_SETTINGS - Enables (true) or disables (false) the printing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



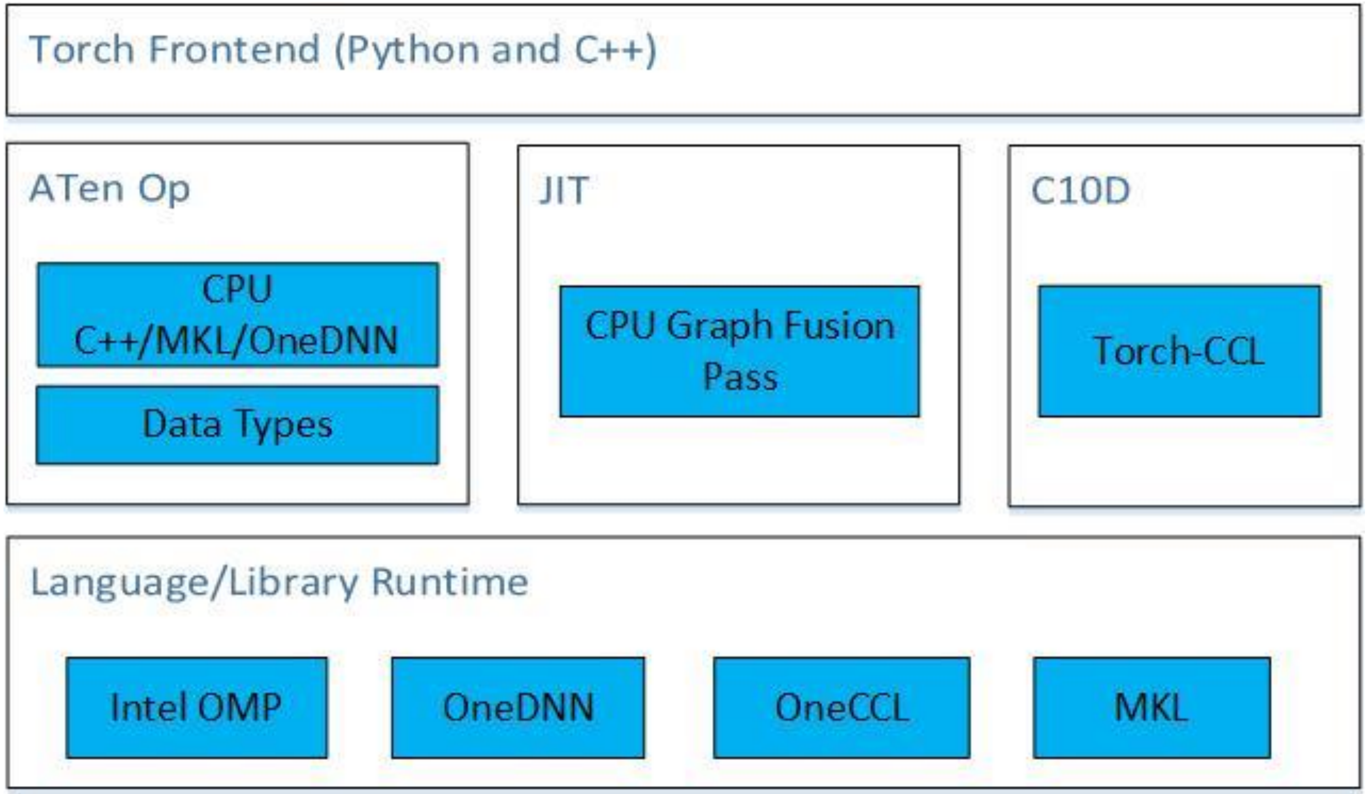
oneDNN architecture overview



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

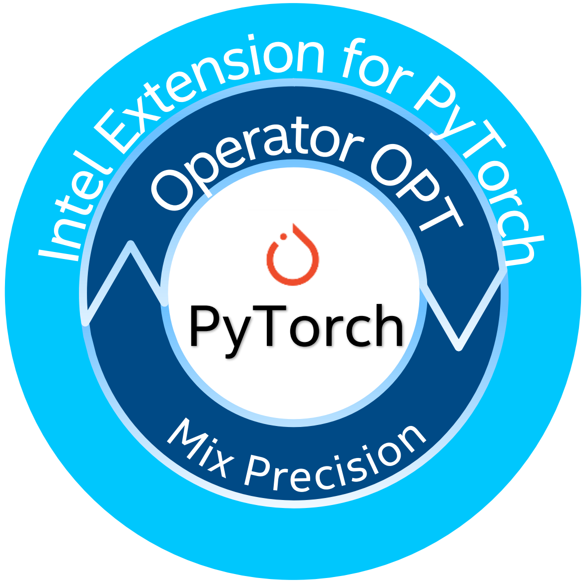
➢Auto dispatch the operators optimized by the extension backend

➢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

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 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

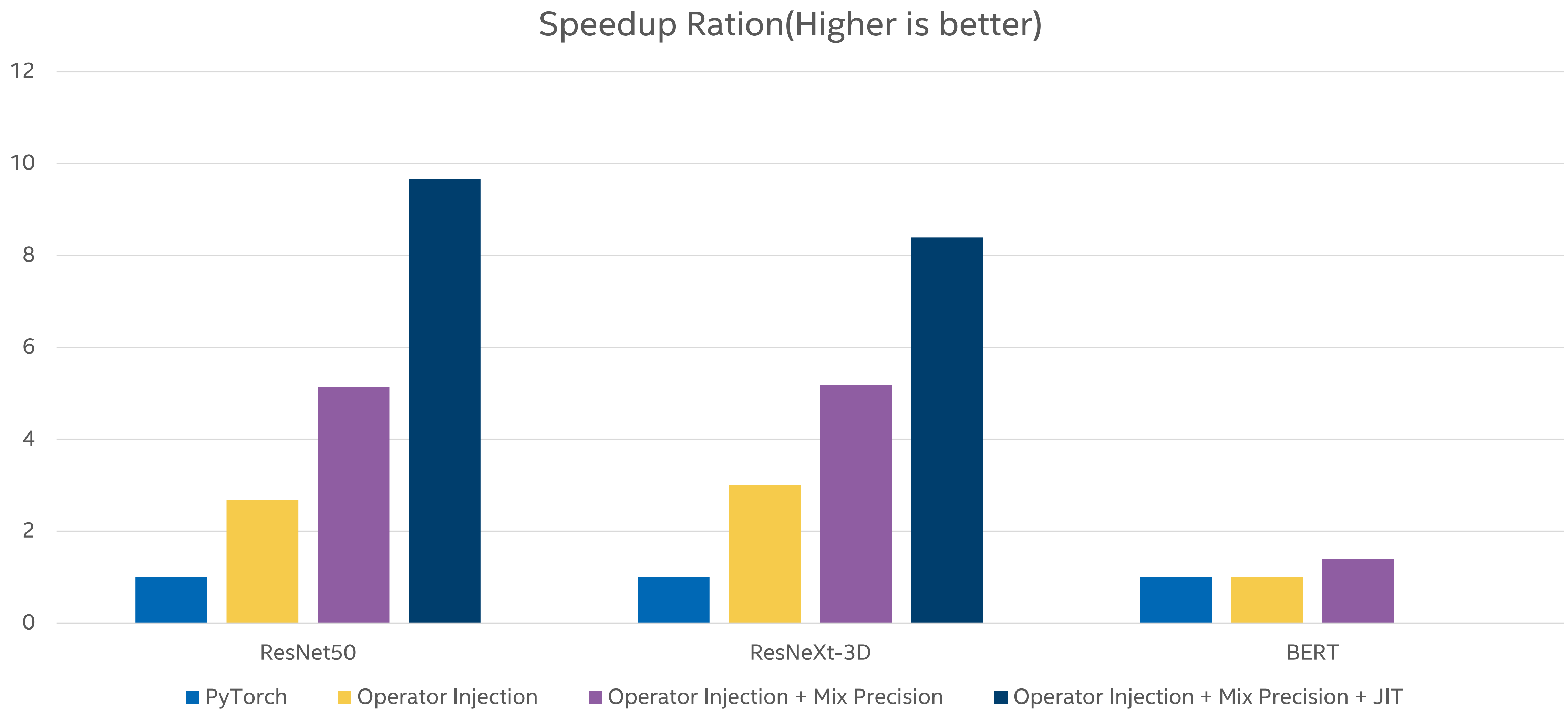
Data types

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

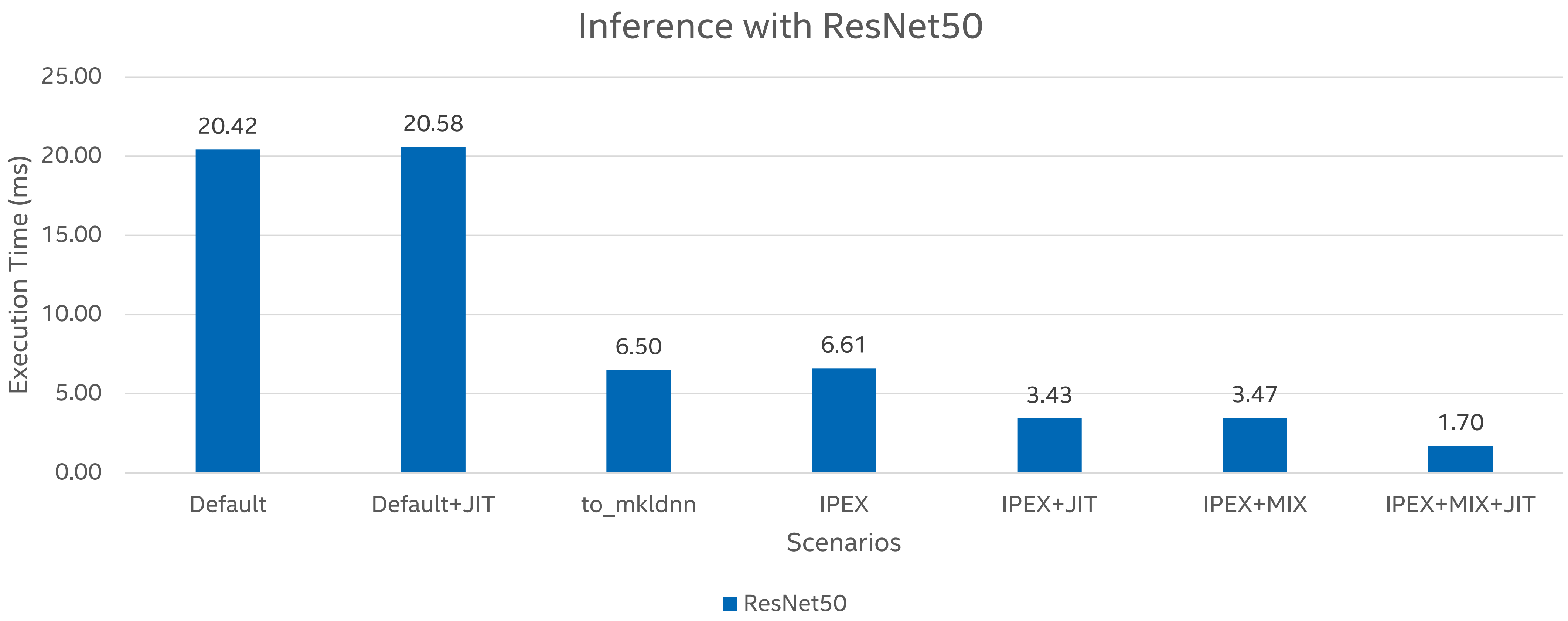
- Benefit of bfloat16
  - Performance 2x up
  - Comparable accuracy loss against fp32
  - No loss scaling, compared to fp16

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

Extension Perf



Inference with IPEX for ResNet50



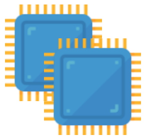
Worker11 (CPX)

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

96.1924

32 -bit

10110110 10110110

10110110 10110110

Integer

96

8 bit

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

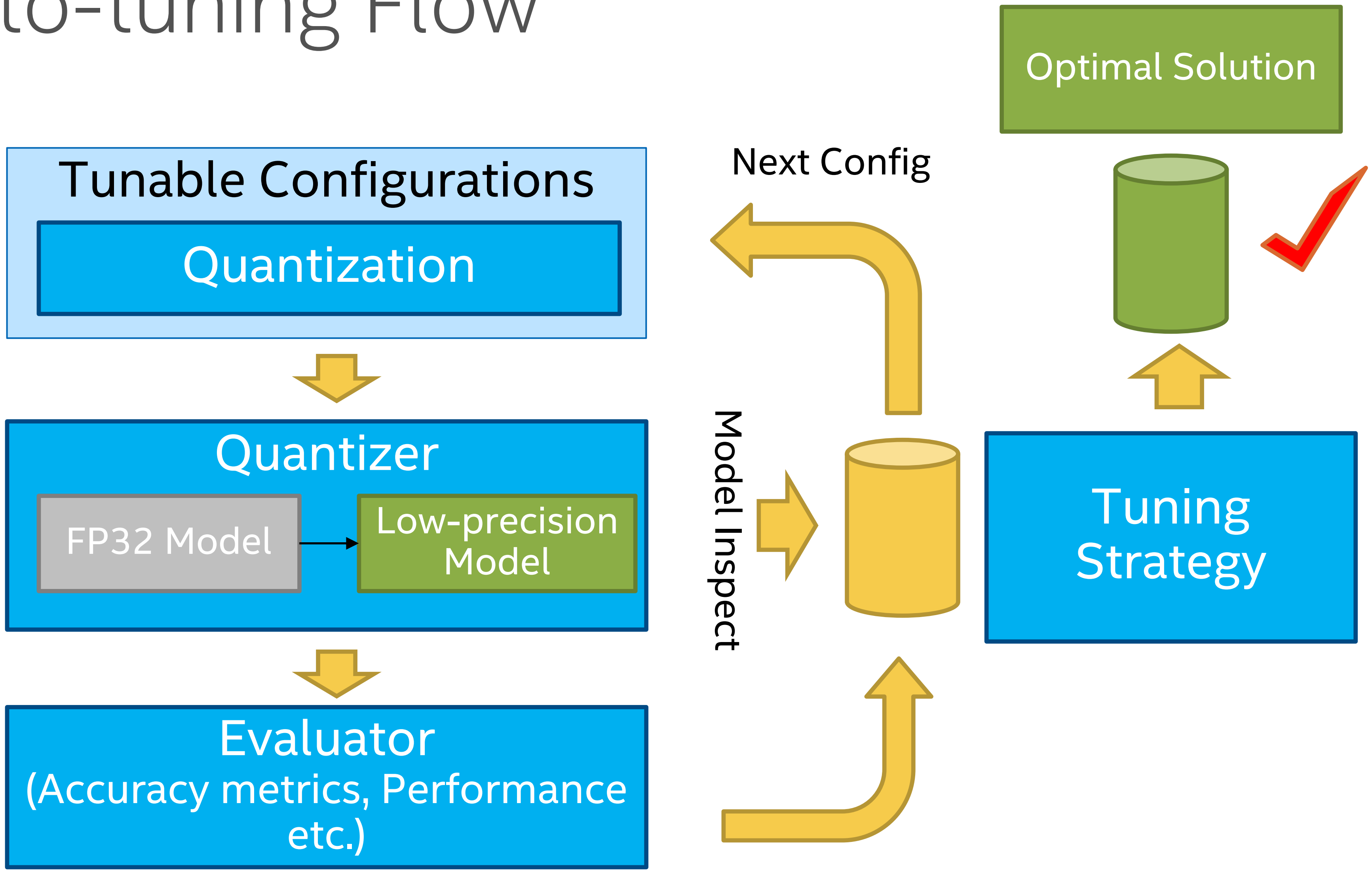
The followings are the models supported by Intel® Low Precision Optimization Tool for auto tuning.

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

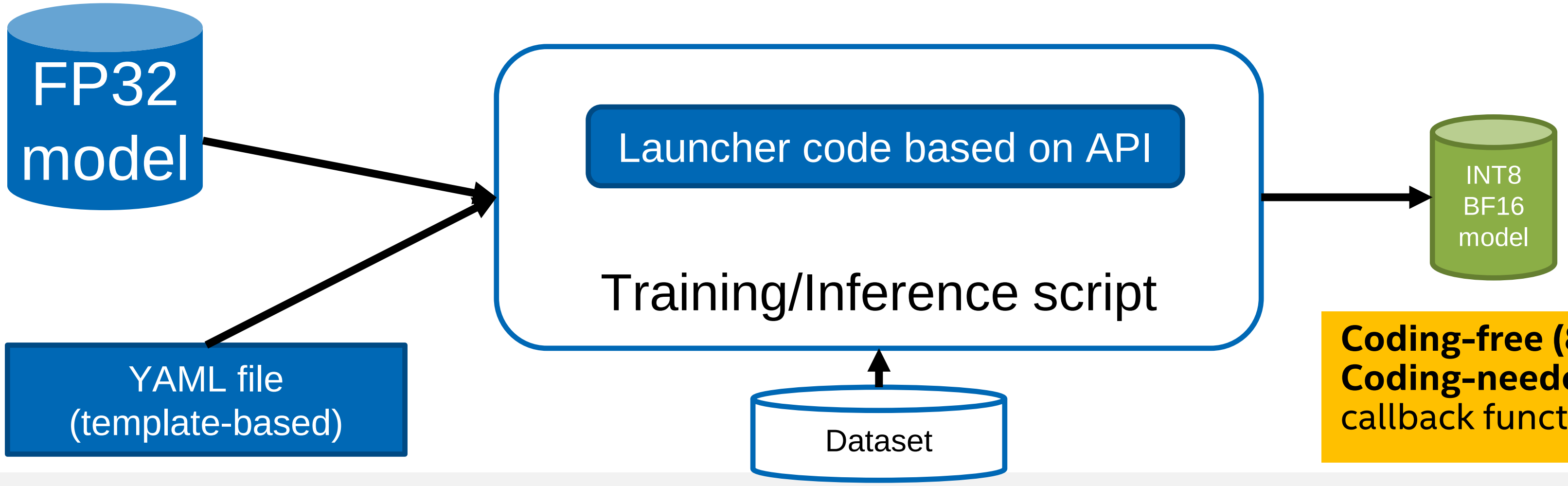
```
# install from pip
pip install lpot
# install from conda
conda install lpot -c intel -c conda-forge
```

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

Intermediate: TensorFlow HelloWorld

YAML config

```yaml
model:
  name: hello_world
  framework: tensorflow
  inputs: input
  outputs: output

quantization:
  calibration:
    sampling_size: 5, 10
  model_wise:
    activation:
      algorithm: minmax

evaluation:
  accuracy:
    metric:
      topk: 1

tuning:
  accuracy_criterion:
    relative: 0.05
  exit_policy:
    timeout: 0
  random_seed: 100
```

No dataloader related setting here, Implemented by code.

Code change

```python
(train_images, train_labels), (test_images,
                                 test_labels) =
keras.datasets.fashion_mnist.load_data()

train_images = train_images.astype(np.float32) / 255.0
test_images = test_images.astype(np.float32) / 255.0

model = tf.keras.models.load_model("../models/simple_model")

import lpot
quantizer = lpot.Quantization('./conf.yaml')
dataloader = quantizer.dataloader(dataset=list(zip(test_images,
test_labels)))
quantized_model = quantizer(model, q_dataloader=dataloader,
eval_dataloader=dataloader)
```

- This example shows how to create LPOT calibration and evaluation dataloader by code and pass them to LPOT for tune.

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

Advanced : TensorFlow SSD-RN50

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

YAML config

```yaml
model:
  name: ssd_resnet50_v1
  framework: tensorflow
  inputs: image_tensor
  outputs: num_detections,detection_boxes,detection_scores,detection_classes

quantization:
  calibration:
    sampling_size: 100
  model_wise:
    activation:
      algorithm: minmax
    weight:
      algorithm: minmax
  op_wise: {
           'FeatureExtractor/resnet_v1_50/fpn/bottom_up_block5/Conv2D': {
             'activation': {'dtype': ['fp32']},
           },
'WeightSharedConvolutionalBoxPredictor_2/ClassPredictionTower/conv2d_0/Conv2D
': {
             'activation': {'dtype': ['fp32']},
           }
         }

tuning:
  accuracy_criterion:
    relative:  0.01
  exit_policy:
    timeout: 0
    max_trials: 100
  random_seed: 9527
```

Constrain model-wise/op-wise quantization behavior in tuning space.

```python
def accuracy_check(self, input_graph=None):
    ...
    self.build_data_sess()
    evaluator = CocoDetectionEvaluator()
    with tf.compat.v1.Session(graph=self.infer_graph,
                              config=self.config) as sess:
        ...
        evaluator.add_single_ground_truth_image_info(
                image_id, ground_truth)
        num, boxes, scores, labels = sess.run(
            self.output_tensors, {self.input_tensor: input_images})
        ...
        return res['DetectionBoxes_Precision/mAP']
```

Code change

```python
infer = model_infer(args)
if args.tune:
    quantizer = Quantization(args.config)
    q_dataloader = quantizer.dataloader(infer, args.batch_size)
    output_graph = quantizer(infer.get_graph(),
                             q_dataloader=q_dataloader,
                             eval_func=infer.accuracy_check)
```

- This example shows how to customize tuning space by YAML at model-wise and op-wise level.

intel

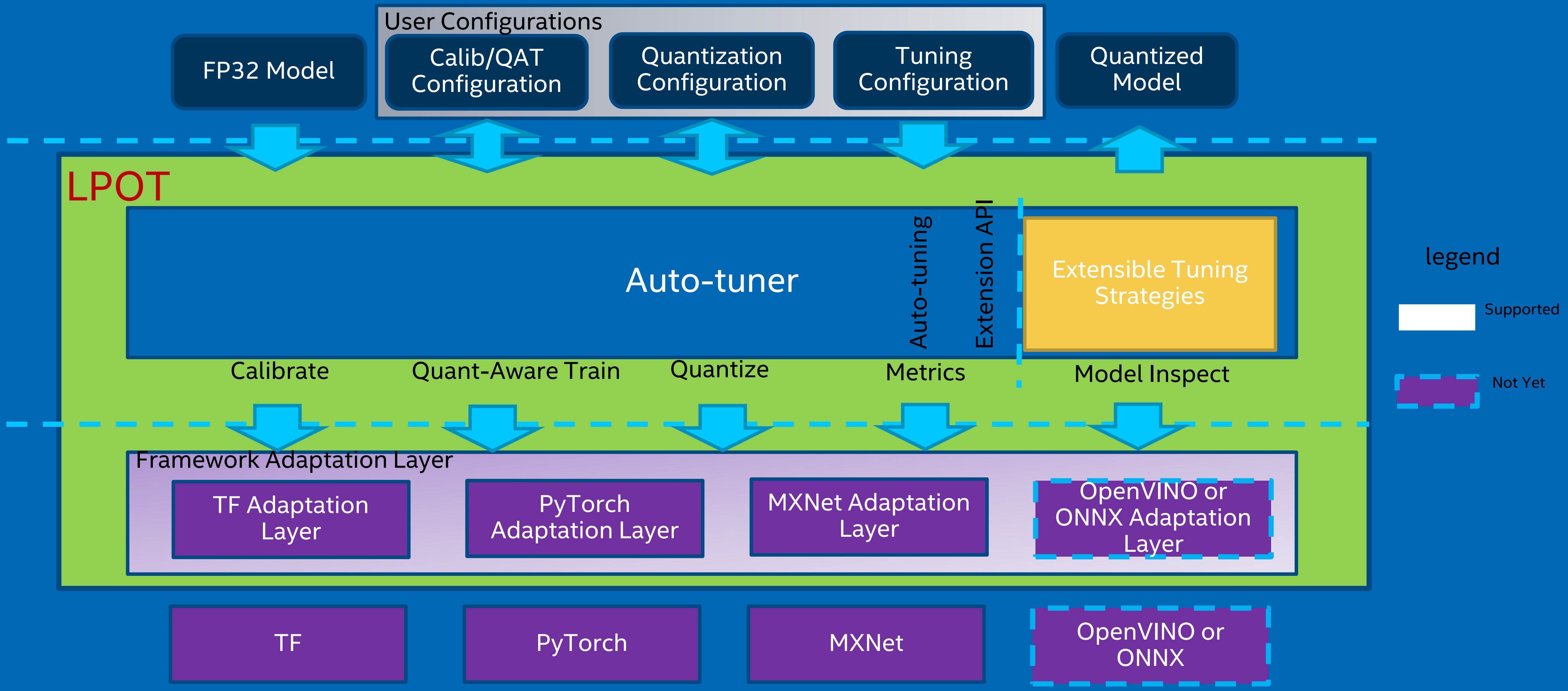
DEMO

Demo

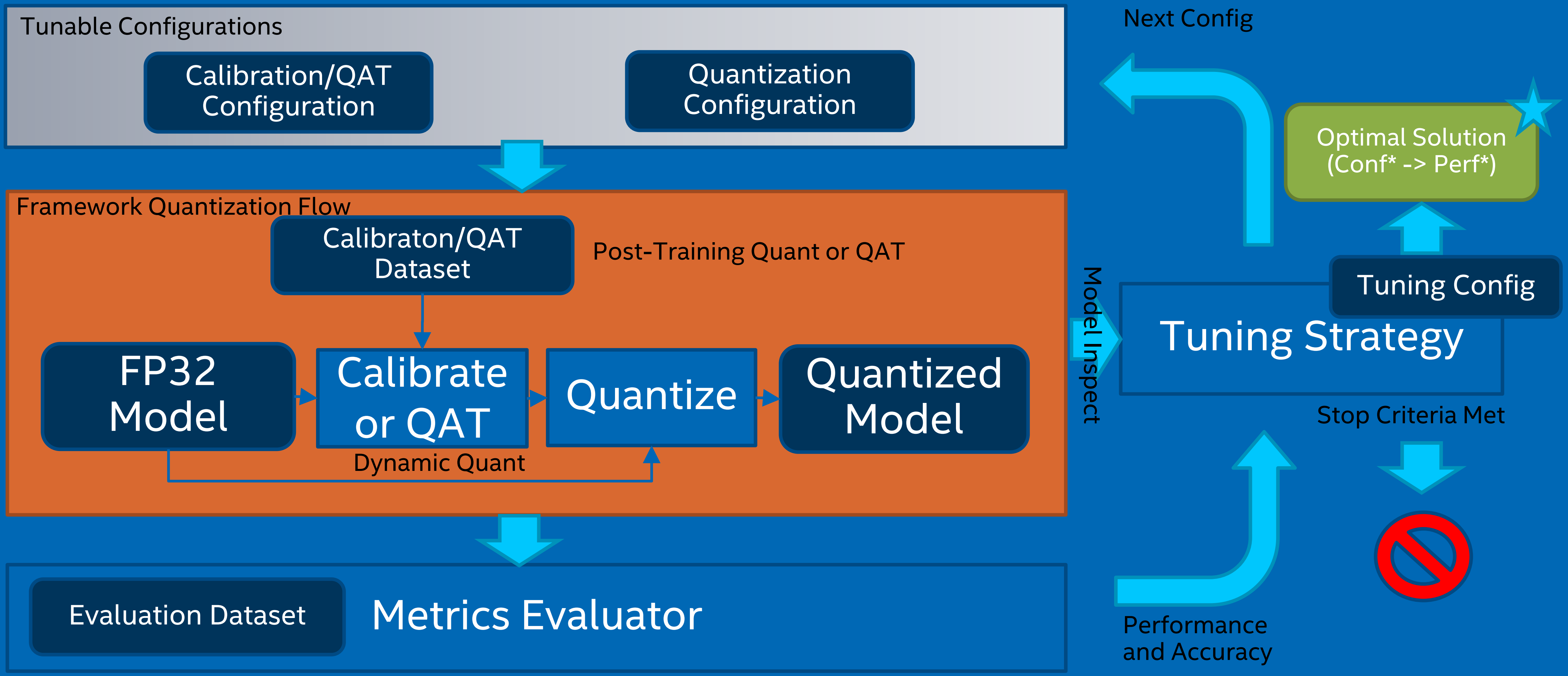
- **Intel AI Analytics Toolkit Samples:**
- https://github.com/oneapi-src/oneAPI-samples/tree/master/AI-and-Analytics

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

Infrastructure



Working Flow



intel®

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.

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.

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