Intel® AI Analytics Toolkit Deep Learning Optimizations

Yuning Qiu, AI Software Solutions Engineer

Aaryan Kothapalli, AI Software Solutions Engineer

intel®

Deep Learning Framework (Optimizations by Intel)

SCALING

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

UTILIZE ALL THE CORES

- OpenMP, MPI
- Reduce synchronization events, serial code
- Improve load balancing

VECTORIZE / SIMD

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

EFFICIENT MEMORY / CACHE USE

- Blocking
- Data reuse
- Prefetching
- Memory allocation

Graph optimizations

- fuse
- Batch normalization
- Memory allocation

TensorFlow

mxnet

Caffe2

PyTorch

BigDL for Spark

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

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

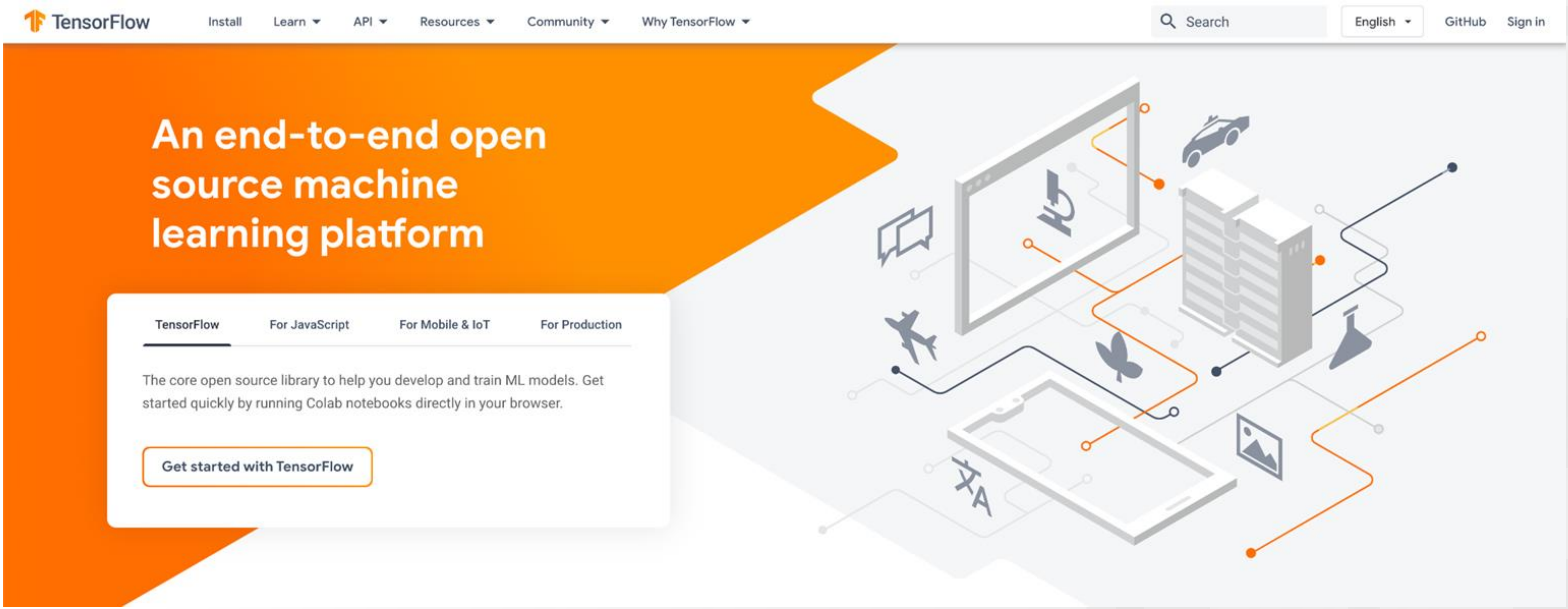
SEE ALSO: Machine Learning Libraries for Python (Scikit-learn, Pandas, NumPy), R (Cart, randomForest, e1071), Distributed (MlLib on Spark, Mahout)
*Limited availability today

Agenda

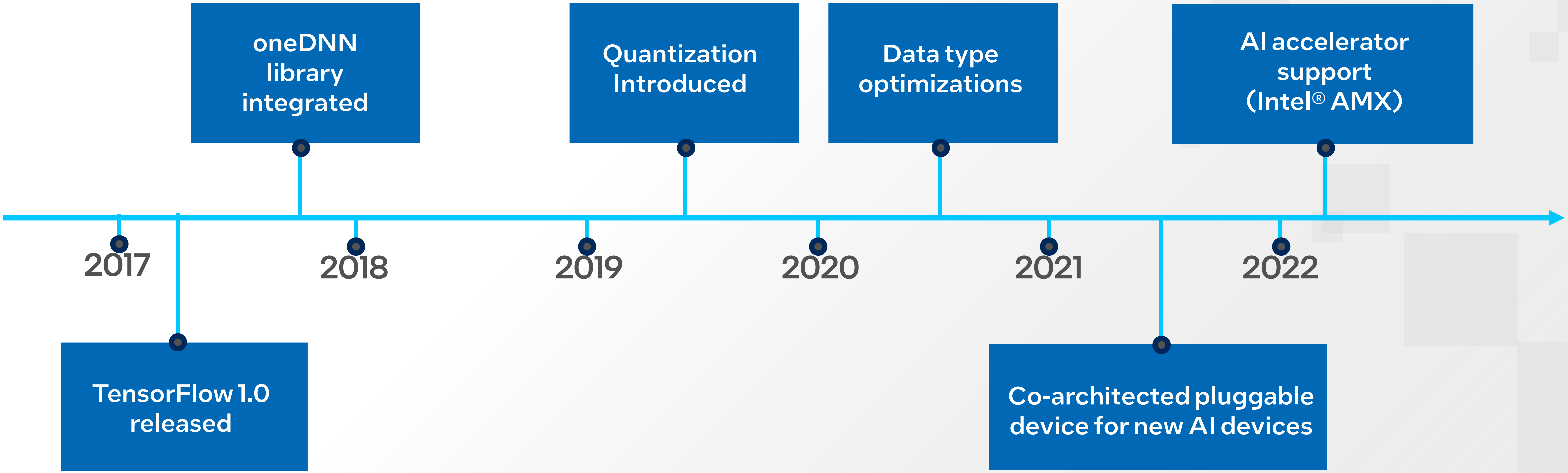
- Intel® Optimization for TensorFlow
- Intel® Optimization for PyTorch
- Speedup DL inference via Intel® Neural Compressor
- Exercises

Intel Optimization for TensorFlow

What is TensorFlow?



Intel Contributions in TensorFlow



4+ years of close collaboration between Intel and Google

oneAPI Deep Neural Network Library (oneDNN)

Features

- Supports FP32, FP16, Bfloat16, and int8.
- Leverages Intel® DL Boost, AVX512 instructions and processor capabilities
- Fused operations for optimized performance

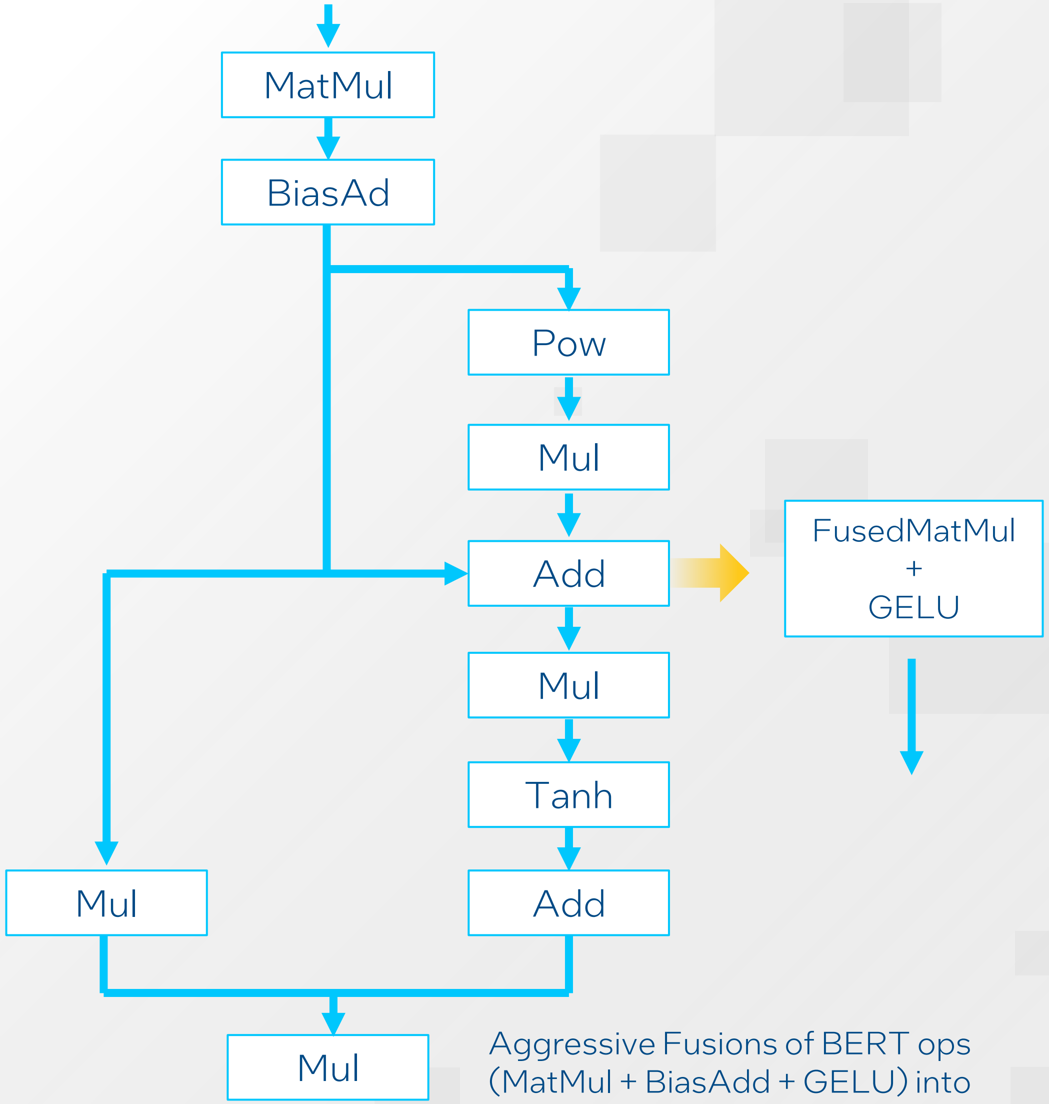
Support Matrix

- Compilers: Intel® oneAPI DPC++ / C++ Compilers
- OS: Linux, Windows, macOS
- CPU: Intel Atom, Intel® Core™, Intel® Xeon®, Intel® Xeon® Scalable processors
- GPU: Intel® Processor Graphics Gen9, Intel® Processor Graphics Gen 12

Category	Functions
Compute intensive operations	• (De-)Convolution • Inner Product • RNN (Vanilla, LSTM, GRU) • GEMM
Memory bandwidth limited operations	• Pooling • Batch Normalization • Local Response Normalization • Layer Normalization • Elementwise • Binary elementwise • Softmax • Sum • Concat • Shuffle
Data manipulation	• Reorder

oneDNN Integration with TensorFlow

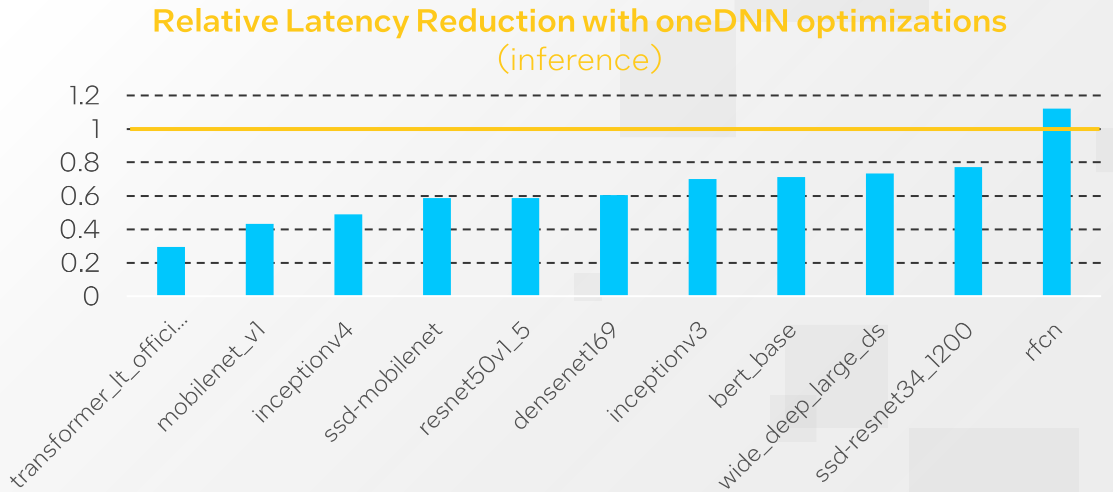
- Replaces compute-intensive standard TF ops with highly optimized custom oneDNN ops
- Aggressive op fusions to improve performance of Convolutions and Matrix Multiplications
- Primitive caching to reduce overhead of calling oneDNN

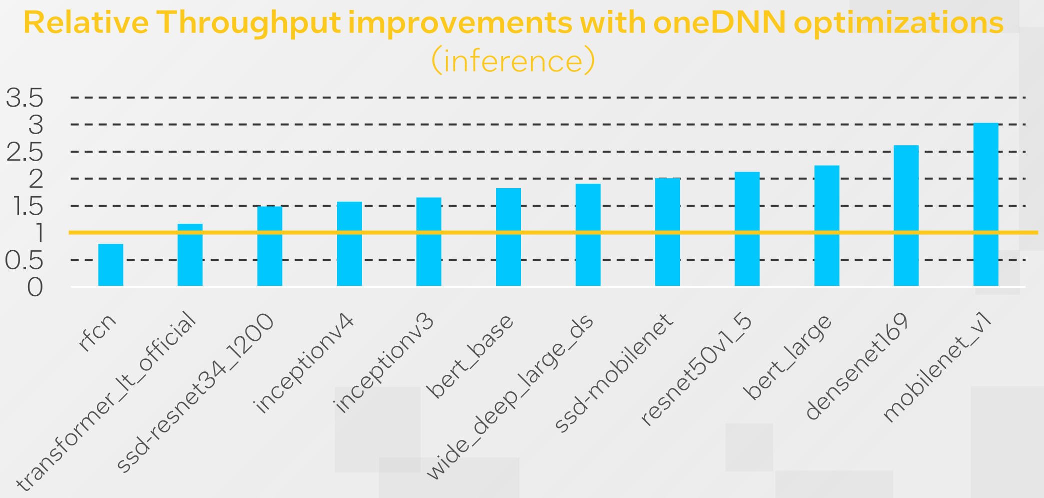


Aggressive Fusions of BERT ops (MatMul + BiasAdd + GELU) into oneDNN op

oneDNN Optimizations in TensorFlow

- Turn on oneDNN optimizations at runtime in official TensorFlow distributions by setting an environment variable TF_ENABLE_ONEDNN_OPTS=1.
- Up to 3X performance improvement
- Supported in TensorFlow-based packages as well (TensorFlow serving, TensorFlow Extended - TFX)





* https://github.com/tensorflow/community/pull/400

https://medium.com/intel-analytics-software/leverage-intel-deep-learning-optimizations-in-tensorflow-129faa80ee07

Intel Optimization for PyTorch

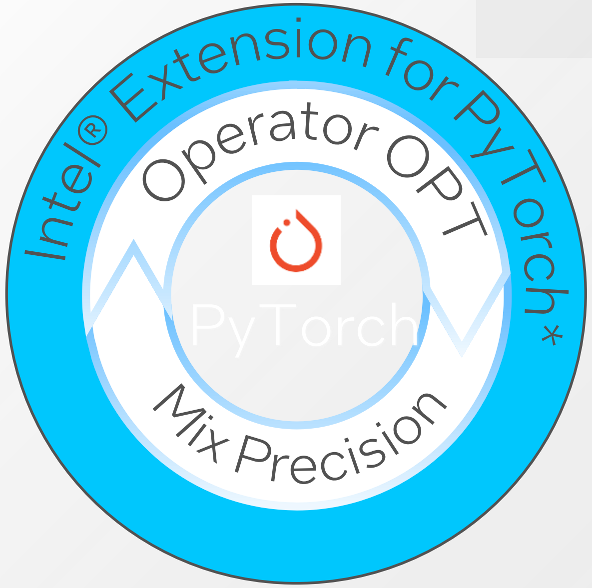
What is Intel Optimization for PyTorch ?

- Intel regularly upstreams most of the CPU optimizations to standard PyTorch
- Intel releases additional features and performance improvements through the Intel Extension for PyTorch
  - As the extension presents more aggressive optimizations, it offers bigger speed-up for training and inference

Intel® Optimization for PyTorch* = Stock PyTorch (with CPU optimizations upstreamed) + Intel® Extension for PyTorch* (IPEX)

Intel® Extension for PyTorch* (IPEX)

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



Operator Optimization
- ➢ Customized operators
- ➢ Auto graph optimization

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

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

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

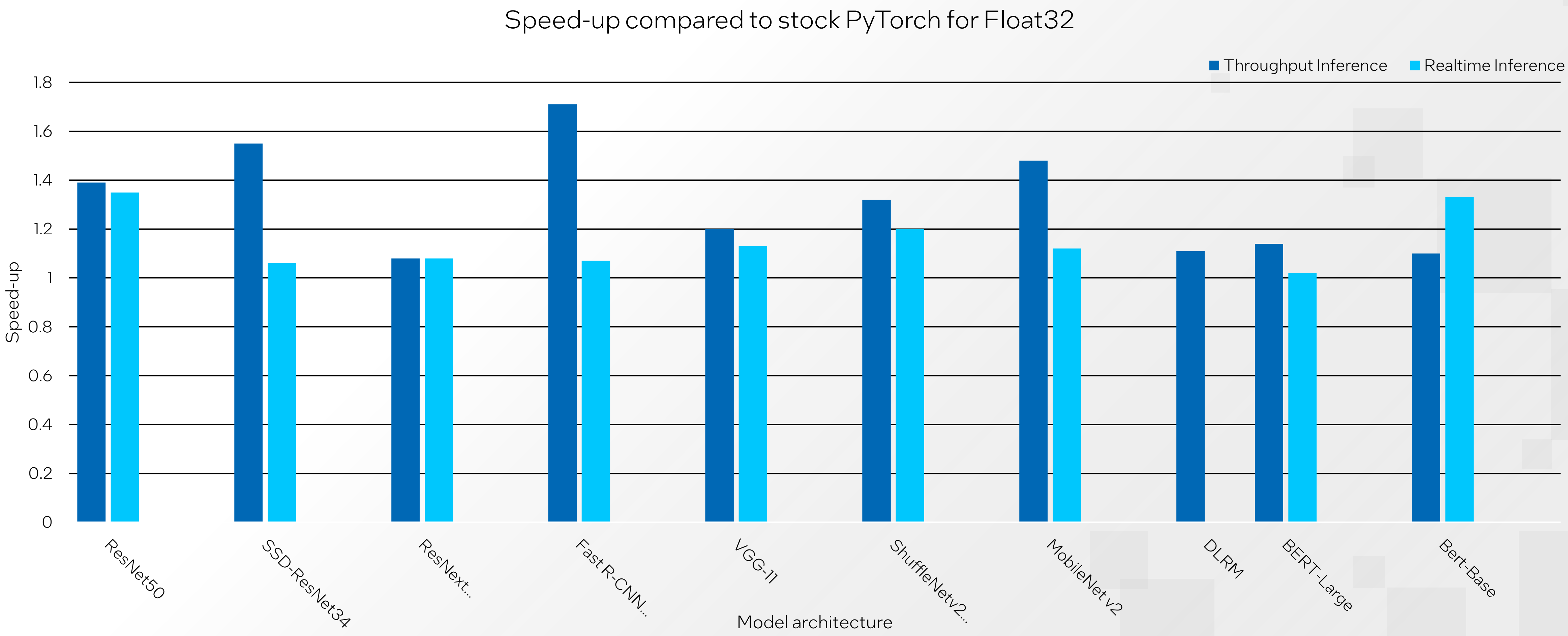
fp32

```python
import torch
import torchvision.models as models

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

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

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

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

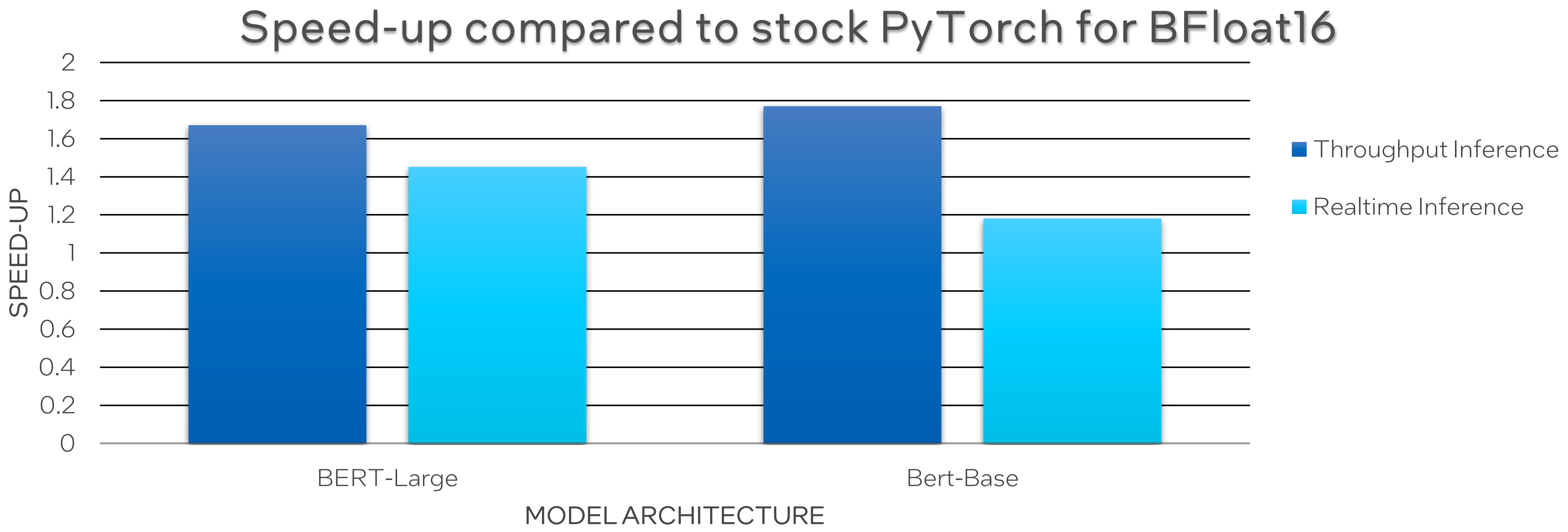
bfloat16

```python
import torch
import torchvision.models as models

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

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

#################### code changes ####################
import intel_extension_for_pytorch as ipex
model = ipex.optimize(model, dtype=torch.bfloat16)
######################################################

with torch.no_grad():
  with torch.cpu.amp.autocast():
    model(data)
```

Intel Extension for PyTorch benchmark for fp32

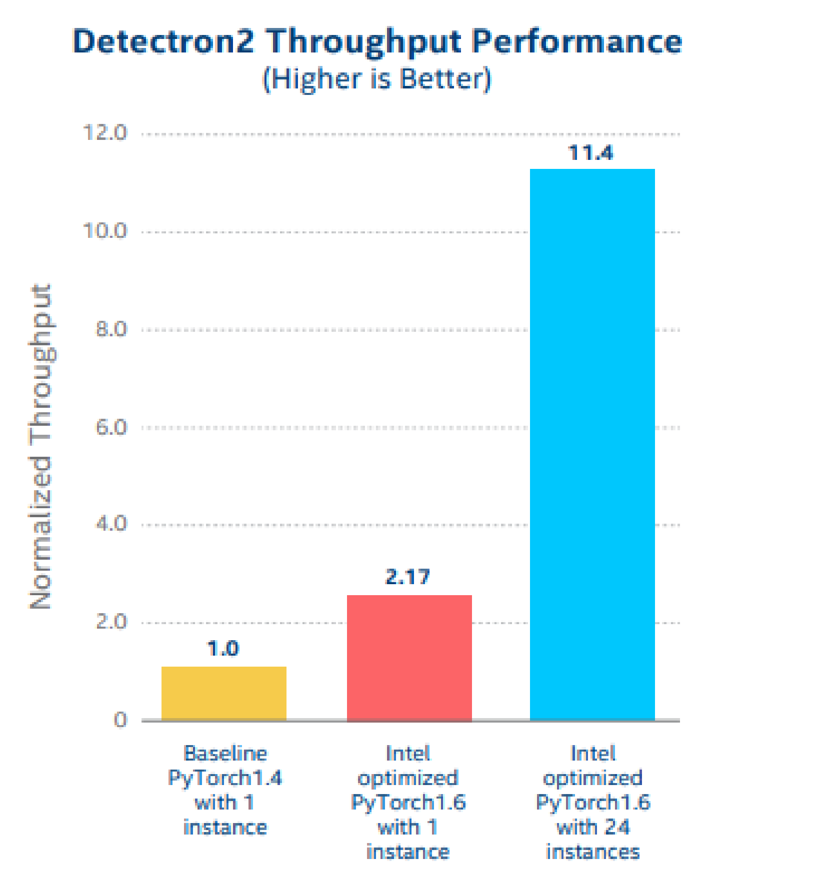


Intel Extension for PyTorch benchmark for BFloat16



Use case with Intel-optimized PyTorch

- Mycobacterium Tuberculosis Detection with Intel-optimized PyTorch

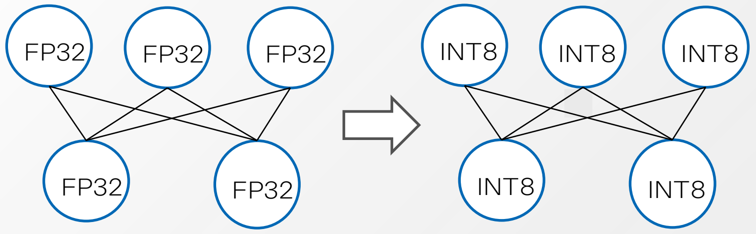


Speedup DL inference via Intel Neural Compressor

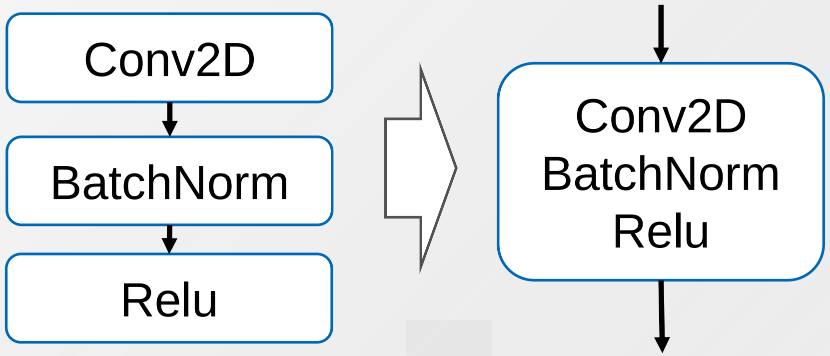
Low-Precision (8-bit Integer) Inference Optimization

- Quantized models using 8-bit integers gaining adoption
  - Improved performance
  - Trade off accuracy for performance
- Additional post-training quantization step needed
- Intel Neural Compressor
  - Automatically quantizes pre-trained model
  - Picks quantization scheme to meet specific performance and accuracy needs

Quantization



Graphic Optimization



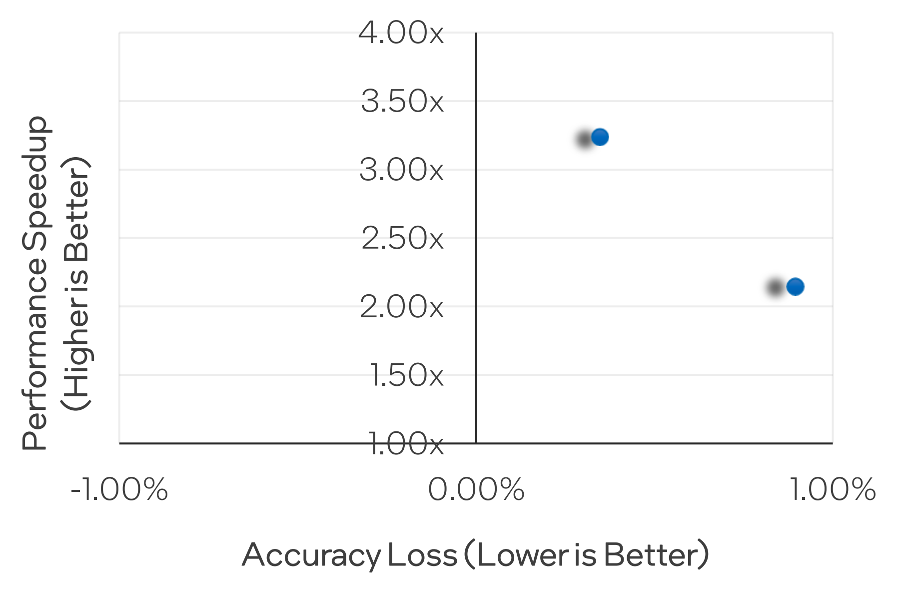
Performance

Stock TF on ICX (FP32)	oneDNN Enabled (FP32)	INC Quantization (INT8)
Baseline	**1.5x**	**3.9x**
13.06	20.54	81.66

6.25x Total Perf Gain

Customer/LZ model: SSD-ResNet-34
(BS=1)

Model Accuracy & Performance



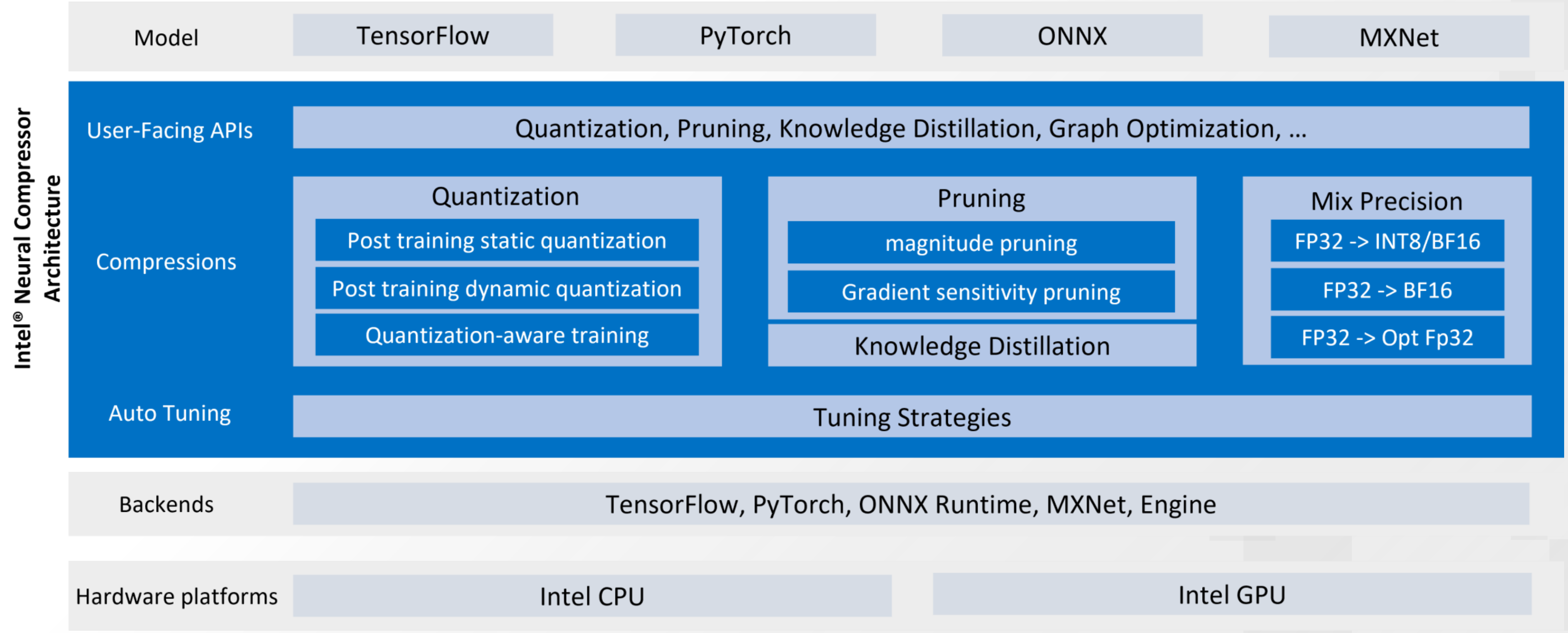
OOB Random Models
(w/ VNNI example)

2.2x geomean and up to 4x performance

Architecture

- Technology
  - Quantization
  - Pruning
  - Knowledge distillation
  - Graphic Optimization
  - Mix Precision

- Platform: Intel CPU and GPU (on going)
- Framework
  - TensorFlow*, including 1.15.0 UP3, 2.5.0, 2.7.0, Official TensorFlow 2.6.2, Official TensorFlow 2.7.0
  - PyTorch*, including 1.8.0+cpu, 1.9.0+cpu, 1.10.0+cpu
  - Apache* MXNet, including 1.6.0, 1.7.0, 1.8.0
  - ONNX* Runtime, including 1.7.0, 1.8.0, 1.9.0
  - Execution Engine, a reference bare metal solution(./engine) for domain-specific NLP models.



Adoption

Alibaba Group 阿里巴巴集团 | ByteDance | CERN | GREE | Haier Inspired Living

Hugging Face | IBM | JD Cloud & AI | Lenovo | 美团网 meituan.com

Meta | Microsoft | NAVER | NETFLIX | ONNX

ORACLE | paloalto NETWORKS | sina 新浪网 .com.cn | SK telecom | Tencent Cloud

Meta

*"**happy to see out of the box performance** by quantizing NLP models on PyTorch **using LPOT/INC**"*

Hugging Face

*"deliver in an **easy and accessible way** through Optimum, as we did with **Intel and Neural Compressor**"*

ONNX

*"support **first INT8 model** quantized by **Intel Neural Compressor** in ONNX model zoo"*

Exercises

- Intel Optimization for PyTorch
- Intel Optimization for TensorFlow
- Intel Neural Compressor

Notices and Disclaimers

- Performance varies by use, configuration and other factors.
  Learn more at www.Intel.com/PerformanceIndex.
- 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.
- Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.
- Your costs and results may vary.
- Intel technologies may require enabled hardware, software or service activation.
-

Workloads and Configurations

See all benchmarks and configurations: https://software.intel.com/content/www/us/en/develop/articles/blazing-fast-python-data-science-ai-performance.html. Each performance claim and configuration data is available in the body of the article listed under sections 1, 2, 3, 4, and 5. Please also visit this page for more details on all scores, and measurements derived.

Testing Date: Performance results are based on testing by Intel as of October 16, 2020 and may not reflect all publicly available updates. **Configurations details and Workload Setup**: 2 x Intel® Xeon® Platinum 8280 @ 28 cores, OS: Ubuntu 19.10.5.3.0-64-generic Mitigated 384GB RAM (192 GB RAM (12x 32GB 2933). SW: Modin 0.81. Scikit-learn 0.22.2. Pandas 1.01, Python 3.8.5, DAL(DAAL4Py) 2020.2, Census Data, (21721922.45) Dataset is from IPUMS USA, University of Minnesota, www.ipums.org [Steven Ruggles, Sarah Flood, Ronald Goeken, Josiah Grover, Erin Meyer, Jose Pacas and Matthew Sobek. IPUMS USA: Version 10.0 [dataset], Minneapolis, MN. IPUMS, 2020. https//doc.org/10.18128/D010.V10.0]

Testing Date: Performance results are based on testing by Intel® as of October 23, 2020 and may not reflect all publicly available updates. **Configuration Details and Workload Setup**: Intel® oneAPI Data Analytics Library 2021.1 (oneDAL). Scikit-learn 0.23.1, Intel® Distribution for Python 3.8; Intel® Xeon® Platinum 8280LCPU @ 270GHz, 2 sockets, 28 cores per socket, 10M samples, 10 features, 100 clusters, 100 iterations, float32.

Testing Date: Performance results are based on testing by Intel® as of October 23, 2020 and may not reflect all publicly available updates. **Configuration Details and Workload Setup**: Intel® AI Analytics Toolkit v2021.1; Intel® oneAPI Data Analytics Library (oneDAL) beta10, Scikit-learn 0.23.1, Intel® Distribution for Python 3.7, Intel® Xeon® Platinum 8280 CPU @ 2.70GHz, 2 sockets, 28 cores per socket, microcode: 0x4003003, total available memory 376 GB, 12X32GB modules, DDR4. **AMD Configuration**: AMD Rome 7742 @2.25 GHz, 2 sockets, 64 cores per socket, microcode: 0x8301038, total available memory 512 GB, 16X32GB modules, DDR4, oneDAL beta10, Scikit-learn 0.23.1, Intel® Distribution for Python 3.7. **NVIDIA Configuration**: NVIDIA Tesla V100 – 16 Gb, total available memory 376 GB, 12X32GB modules, DDR4, Intel® Xeon Platinum 8280 CPU @ 2.70GHz, 2 sockets, 28 cores per socket, microcode: 0x5003003, cuDF 0.15, cuML 0.15, CUDA 10.2.89, driver 440.33.01, Operation System: CentOS Linux 7 (Core), Linux 4.19.36 kernel.

Testing Date: Performance results are based on testing by Intel® as of October 13, 2020 and may not reflect all publicly available updates. **Configurations details and Workload Setup**: CPU: c5.18xlarge AWS Instance (2 x Intel® Xeon® Platinum 8124M @ 18 cores. OS: Ubuntu 20.04.2 LTS, 193 GB RAM. GPU: p3.2xlarge AWS Instance (GPU: NVIDIA Tesla V100 16GB, 8 vCPUs, OS: Ubuntu 18.04.2LTS, 61 GB RAM. SW: XGBoost 1.1: build from sources compiler – G++ 7.4, nvcc 9.1 Intel® DAAL: 2019.4 version: Python env: Python 3.6, Numpy 1.16.4, Pandas 0.25 Scikit-learn 0.21.2.

Workloads and Configurations

Testing Date: Performance results are based on testing by Intel® as of October 26, 2020 and may not reflect all publicly available updates. **Configuration Details and Workload Setup**: Intel® Optimization for Tensorflow v2.2.0; oneDNN v1.2.0; Intel® Low Precision Optimization Tool v1.0; Platform; Intel® Xeon® Platinum 8280 CPU; #Nodes 1; #Sockets: 2; Cores/socket: 28; Threads/socket: 56; HT: On; Turbo: On; BIOS version:SE5C620.86B.02.01.0010.010620200716; System DDR Mem Config: 12 slots/16GB/2933; OS: CentOS Linux 7.8; Kernel: 4.4.240-1.el7.elrepo x86_64.

Testing Date: Performance results are based on testing by Intel® as of February 3, 2021 and may not reflect all publicly available updates. **Configuration Details and Workload Setup:** Intel® Optimization for PyTorch v1.5.0; Intel® Extension for PyTorch (IPEX) 1.1.0; oneDNN version: v1.5; DLRM: Training batch size (FP32/BF16): 2K/instance, 1 instance; DLRM dataset (FP32/BF16): Criteo Terabyte Dataset; BERT-Large: Training batch size (FP32/BF16): 24/Instance. 1 Instance on a CPU socket. Dataset (FP32/BF16): WikiText-2 [https://www.salesforce.com/products/einstein/ai-research/the-wiktext-dependency-language-modeling-dataset/]: ResNext101-32x4d: Training batch size (FP32/BF16): 128/Instance, 1 instance on a CPU socket, Dataset (FP32/BF16): ILSVRC2012; DLRM: Inference batch size (INT8): 16/instance, 28 instances, dummy data. Intel® Xeon® Platinum 8380H Processor, 4 socket, 28 cores HT On Turbo ON Total memory 768 GB (24 slots/32GB/3200 MHz), BIOS; WLYDCRBLSYS.0015.P96.2005070242 (ucode: OX 700001b), Ubuntu 20.04 LTS, kernel 5.4.0-29-genen: ResNet50: [https://github.com/Intel/optimized-models/tree/master/pytorch/ResNet50]: ResNext101 32x4d: [https://github.com/intel/optimized-models/tree/master/pytorch/ResNext101_32x4ct: DLRM: https//github.com/intel/optimized-models/tree/master/pytorch/dlrm].

Configuration Details

NYCTaxi Workload performance:

For 20 million rows: Dual socket Intel(R) Xeon(R) Platinum 8280L CPUs (S2600WFT platform), 28 cores per socket, hyperthreading enabled, turbo mode enabled, NUMA nodes per socket=2, BIOS: SE5C620.86B.02.01.0013.121520200651, kernel: 5.4.0-65-generic, microcode: 0x4003003, OS: Ubuntu 20.04.1 LTS, CPU governor: performance, transparent huge pages: enabled, System DDR Mem Config: slots / cap / speed: 12 slots / 32GB / 2933MHz, total memory per node: 384 GB DDR RAM, boot drive: INTEL SSDSC2BB800G7. For 1 billion rows: Dual socket Intel Xeon Platinum 8260M CPU, 24 cores per socket, 2.40GHz base frequency, DRAM memory: 384 GB 12x32GB DDR4 Samsung @ 2666 MT/s 1.2V, Optane memory: 3TB 12x256GB Intel Optane @ 2666MT/s, kernel: 4.15.0-91-generic, OS: Ubuntu 20.04.4

End-to-End Census Workload performance (Stock):

Tested by Intel as of 2/19/2021. 2 x Intel® Xeon Platinum 8280L @ 28 cores, OS: Ubuntu 20.04.1 LTS Mitigated, 384GB RAM (384GB RAM: 12x 32GB 2933MHz), kernel: 5.4.0-65-generic, microcode: 0x4003003, CPU governor: performance. SW: Scikit-learn 0.24.1, Pandas 1.2.2, Python 3.9.7, Census Data, (21721922, 45) Dataset is from IPUMS USA, University of Minnesota, www.ipums.org [*Steven Ruggles, Sarah Flood, Ronald Goeken, Josiah Grover, Erin Meyer, Jose Pacas and Matthew Sobek. IPUMS USA: Version 10.0 [dataset]. Minneapolis, MN: IPUMS, 2020. https://doi.org/10.18128/D010.V10.0]*

End-to-End Census Workload performance (Optimized):

Tested by Intel as of 2/19/2021. 2 x Intel® Xeon Platinum 8280L @ 28 cores, OS: Ubuntu 20.04.1 LTS Mitigated, 384GB RAM (384GB RAM: 12x 32GB 2933MHz), kernel: 5.4.0-65-generic, microcode: 0x4003003, CPU governor: performance. SW: Scikit-learn 0.24.1 accelerated by daal4py (now sklearnex) 2021.2, modin 0.8.3, omniscidbe v5.4.1, Python 3.9.7, Census Data, (21721922, 45) Dataset is from IPUMS USA, University of Minnesota, www.ipums.org [*Steven Ruggles, Sarah Flood, Ronald Goeken, Josiah Grover, Erin Meyer, Jose Pacas and Matthew Sobek. IPUMS USA: Version 10.0 [dataset]. Minneapolis, MN: IPUMS, 2020. https://doi.org/10.18128/D010.V10.0]*