

AI on Intel® Architecture

Overview of optimization packages that could be used for your solution



Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](https://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.

Your costs and results may vary.

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

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

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

PyTorch Benchmarking Configurations

4th Generation Intel® Xeon® Scalable Processors

Hardware and software configuration (measured October 24, 2022):

- **Deep Learning config:**
 - Hardware configuration for Intel® Xeon® Platinum 8480+ processor (formerly code named Sapphire Rapids): 2 sockets, 56 cores, 350 watts, 16 x 64 GB DDR5 4800 memory, BIOS version EGSDCRB1.SYS.8901.P01.2209200243, operating system: CentOS* Stream 8, using Intel® Advanced Matrix Extensions (Intel® AMX) int8 and bf16 with Intel® oneAPI Deep Neural Network Library (oneDNN) v2.7 optimized kernels integrated into Intel® Extension for PyTorch* v1.13, Intel® Extension for TensorFlow* v2.12, and Intel® Distribution of OpenVINO™ toolkit v2022.3. Measurements may vary.
 - Wall power refers to platform power consumption.
 - If the dataset is not listed, a synthetic dataset was used to measure performance. Accuracy (if listed) was validated with the specified dataset.
- **Transfer Learning config:**
 - Hardware configuration for Intel® Xeon® Platinum 8480+ processor (formerly code named Sapphire Rapids): Use DLSA single node fine tuning, Vision Transfer Learning using single node, 56 cores, 350 watts, 16 x 64 GB DDR5 4800 memory, BIOS version EGSDREL1.SYS.8612.P03.2208120629, operating system: Ubuntu 22.04.1 LT, using Intel® Advanced Matrix Extensions (Intel® AMX) int8 and bf16 with Intel® oneAPI Deep Neural Network Library (oneDNN) v2.6 optimized kernels integrated into Intel® Extension for PyTorch* v1.12, and Intel® oneAPI Collective Communications Library v2021.5.2. Measurements and some software configurations may vary.

3rd Generation Intel® Xeon® Scalable Processors

Hardware and software configuration (measured October 24, 2022):

- Hardware configuration for Intel® Xeon® Platinum 8380 processor (formerly code named Ice Lake): 2 sockets, 40 cores, 270 watts, 16 x 64 GB DDR5 3200 memory, BIOS version SE5C620.86B.01.01.0005.2202160810, operating system: Ubuntu 22.04.1 LTS, int8 with Intel® oneAPI Deep Neural Network Library (oneDNN) v2.6.0 optimized kernels integrated into Intel® Extension for PyTorch* v1.12, Intel® Extension for TensorFlow* v2.10, and Intel® oneAPI Data Analytics Library (oneDAL) 2021.2 optimized kernels integrated into Intel® Extension for Scikit-learn* v2021.2. XGBoost v1.6.2, Intel® Distribution of Modin* v0.16.2, Intel oneAPI Math Kernel Library (oneMKL) v2022.2, and Intel® Distribution of OpenVINO™ toolkit v2022.3. Measurements may vary.
- If the dataset is not listed, a synthetic dataset was used to measure performance. Accuracy (if listed) was validated with the specified dataset.

*All performance numbers are acquired running with 1 instance of 4 cores per socket

Machine Learning

Modin

Intel distribution of Modin



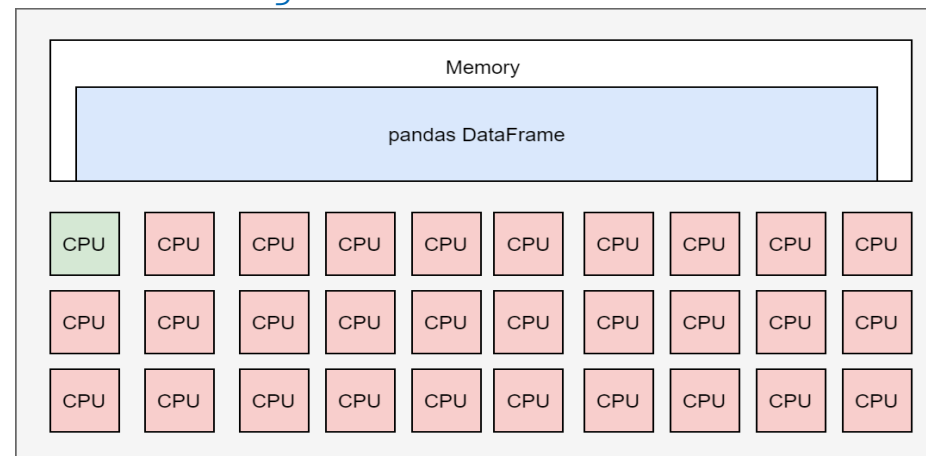
- Pandas is a Python package for data manipulation and analysis that offers data structures and operations for manipulating numerical tables and time series
- **Modin** = Pandas + Scalability
- As simple as `import modin.pandas as pd`

```
import pandas as pd
```

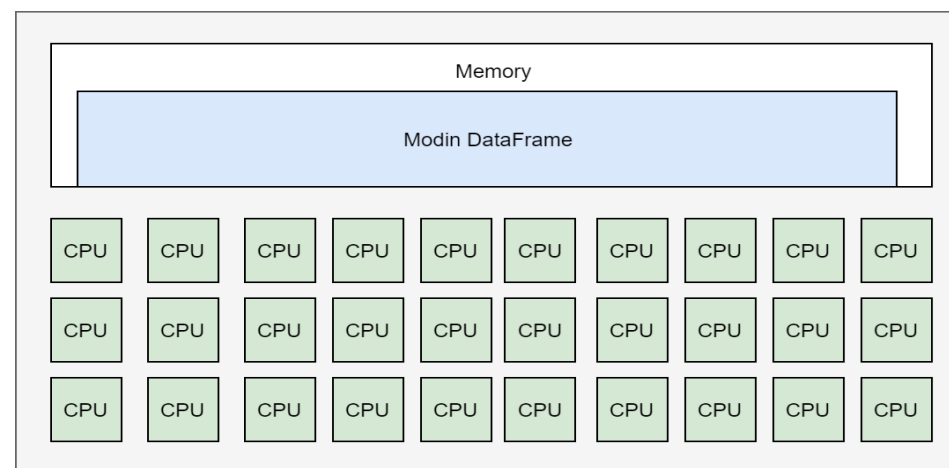
Intel distribution of Modin

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

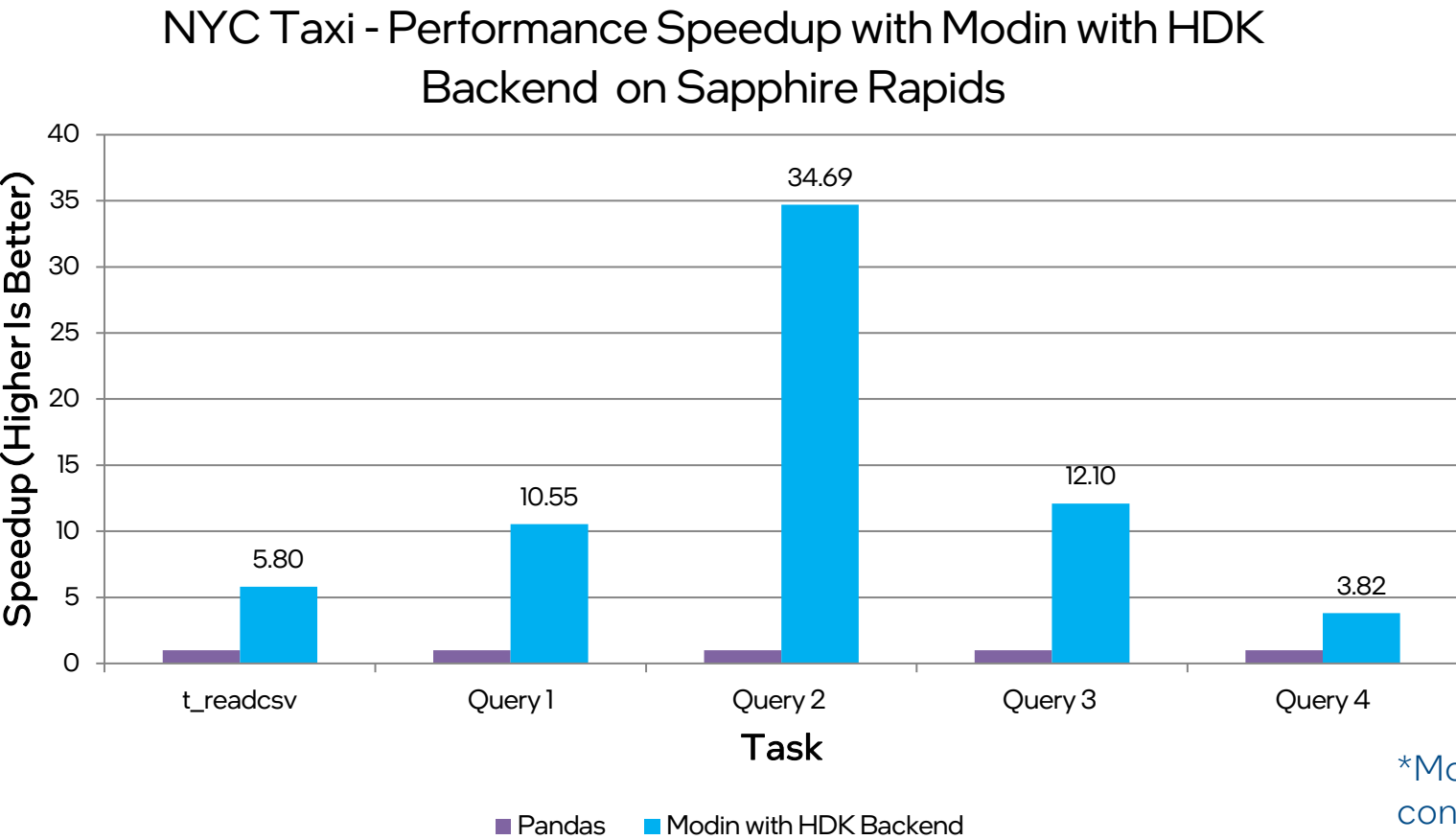
Pandas* on Big Machine



Modin on Big Machine



Intel® Distribution of Modin NYC Taxi Benchmark on SPR



Testing Date: Performance results are based on testing by Intel as of March 03, 2023 and may not reflect all publicly available security updates


Configuration Details and Workload Setup: 2.0 GHz Intel Xeon Platinum 8468, two sockets, 48 cores per socket, 1024 GB DDR5 4800MT/s, 16x64GB DDR5 4800 MT/s, Hyperthreading enabled, Turbo mode enabled, NUMA nodes per socket=2, **BIOS:** SE5C7411.86B.9223.D04.2211291343, **Microcode:** 0x2b000111, **OS:** Ubuntu 22.04.2 LTS, **Kernel:** 5.15.0-60-generic, Python 3.9, Modin 0.18, pyhdk 0.3.1, pandas 1.5.3, Dataset <https://github.com/toddwscneider/nyc-taxi-data>. Query 1 operation utilizes select, count(), and groupby() operations on column cab_type(), Query 2 operation utilizes select, agg(), and groupby() operations on passenger_count and total_amount columns, Query 3 operation utilizes select, count, and groupby operations on passenger_count and pickup_datetime columns, Query 4 operation utilizes select, groupby(), count(), reset_index(), sort_values() operations on passenger_count, pickup_datetime, distance columns

Performance results are based on testing as of dates shown in configurations. See configuration disclosure for details. Not product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex. Your costs and results may vary

Intel® Extension for Scikit-learn

THE MOST POPULAR ML PACKAGE FOR PYTHON*

[Install](#) [User Guide](#) [API](#) [Examples](#) [More ▾](#)

Go

scikit-learn
Machine Learning in Python

Getting Started

Release Highlights for 0.24

GitHub

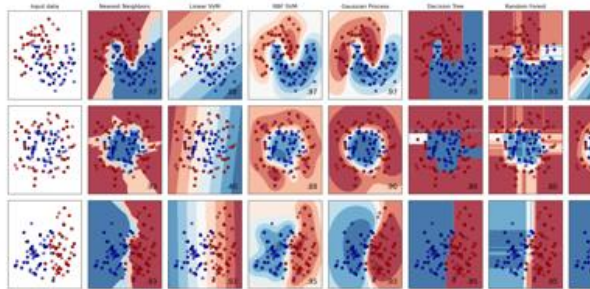
- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.

Algorithms: SVM, nearest neighbors, random forest, and more...

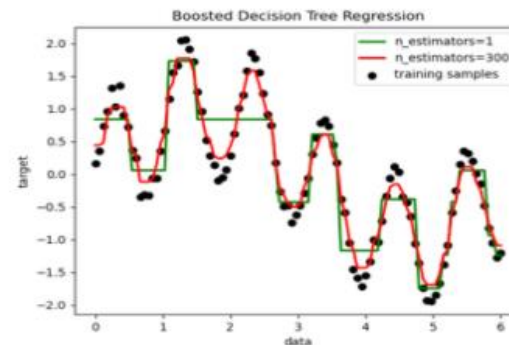


Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, nearest neighbors, random forest, and more...

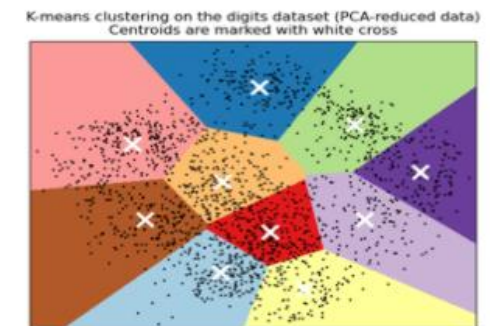


Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, and more...



Intel® Extension for Scikit-learn

Add in the code

```
from sklearnex import patch_sklearn
patch_sklearn()
```

OR

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

```
python -m sklearnex my_application.py
```

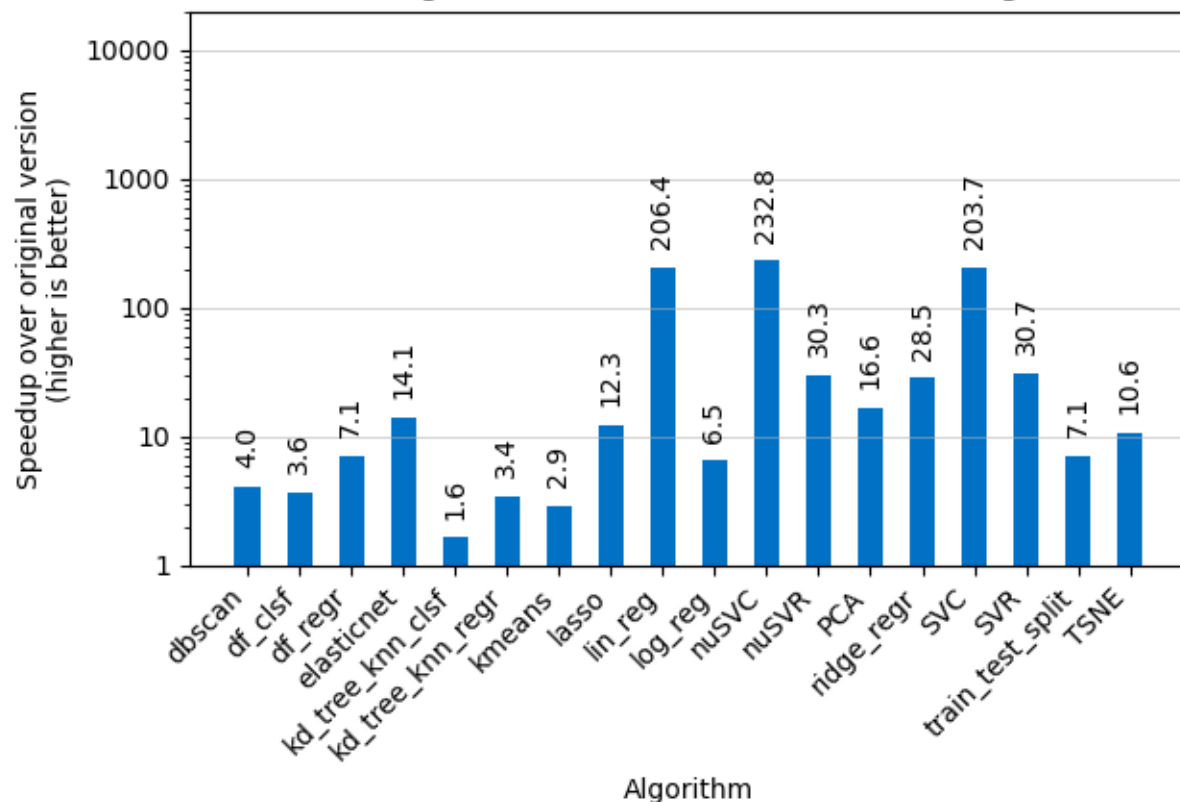
Same Code,
Same Behavior



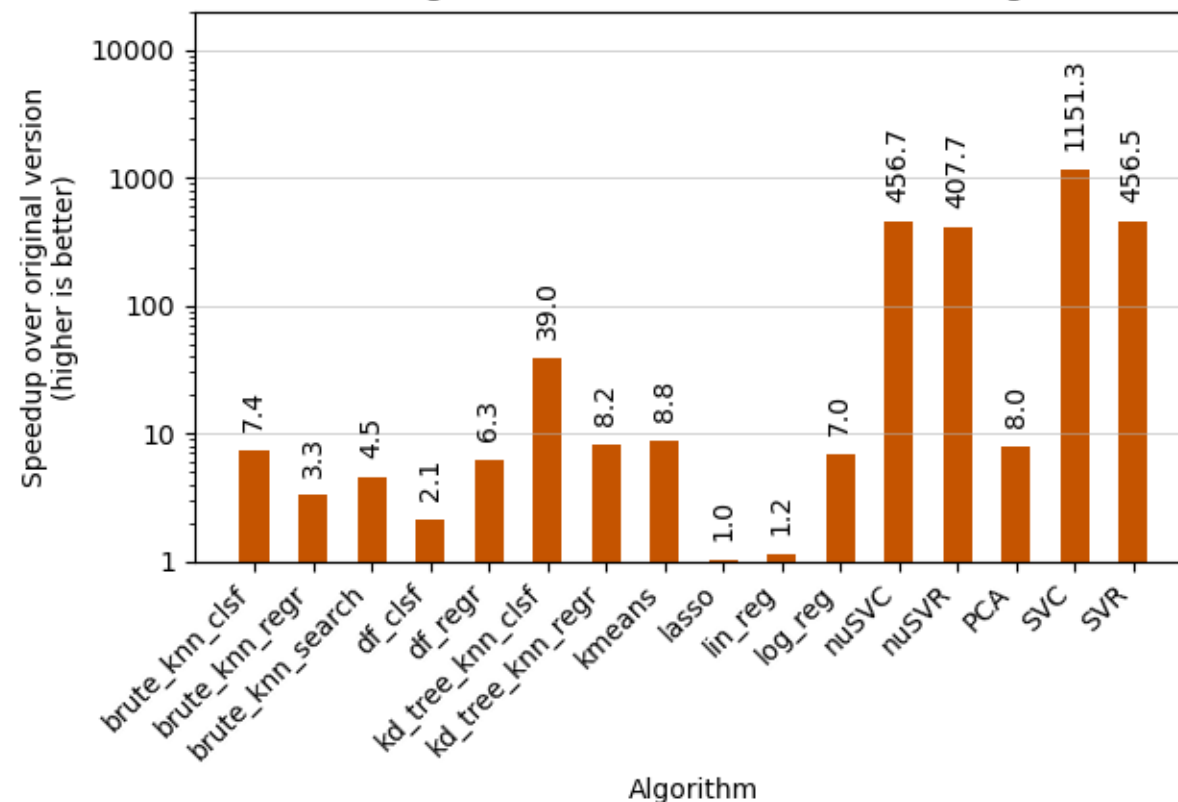
- Scikit-learn, not scikit-learn-like
- Scikit-learn conformance (mathematical equivalence) defined by Scikit-learn Consortium, continuously vetted by public CI

Speedup summaries (float32 and float64 data combined)

Training speedup of Intel® Extension for Scikit-learn* over the original Scikit-learn* for different ML algorithms



Inference speedup of Intel® Extension for Scikit-learn* over the original Scikit-learn* for different ML algorithms



Testing Date: Performance results are based on testing by Intel as of March 21, 2023 and may not reflect all publicly available security updates.

Configuration Details and Workload Setup: bare metal (2.0 GHz Intel Xeon Platinum 8480+, two sockets, 56 cores per socket), 512 GB DDR5 4800MT/s, Python 3.10, scikit-learn 1.2.0, scikit-learn-intelx 2023.0.1. Intel optimizations include use of multi-threading implementation for SKLearn algorithms (which are typically single-threaded), as well as other HW/SW optimizations.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. Not product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex. Your costs and results may vary

XGBoost

Gradient Boosting - Overview

Gradient Boosting:

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

Gradient Boosting optimizations

- XGBoost Training -> upstreamed
- XGBoost Inference -> have to switch to oneDAL backend with daal4py package
- Supported implementations: XGBoost, LightGBM, CatBoost

XGBoost* and LightGBM* Prediction Acceleration with Daal4Py

- Custom-trained XGBoost* and LightGBM* Models utilize Gradient Boosting Tree (GBT) from Daal4Py library for performance on CPUs
- No accuracy loss; 23x performance boost by simple model conversion into daal4py GBT:

```
# Train common XGBoost model as usual
xgb_model = xgb.train(params, X_train)

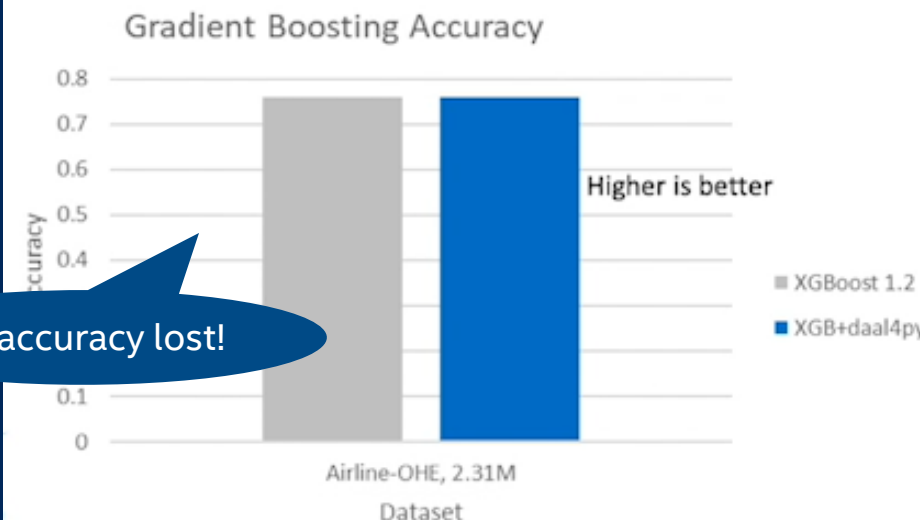
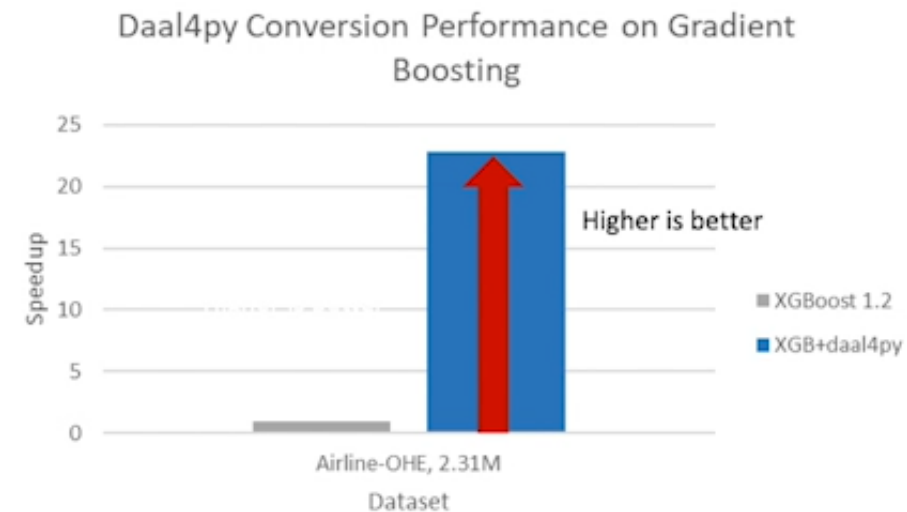
import daal4py as d4p

# XGBoost model to DAAL model
daal_model = d4p.get_gbt_model_from_xgboost(xgb_model)

# make fast prediction with DAAL
daal_prediction = d4p.gbt_classification_prediction(...).compute(X_test, daal_model)
```

- Advantages of daal4py GBT model:
 - More efficient model representation in memory
 - Avx512 instruction set usage
 - Better L1/L2 caches locality

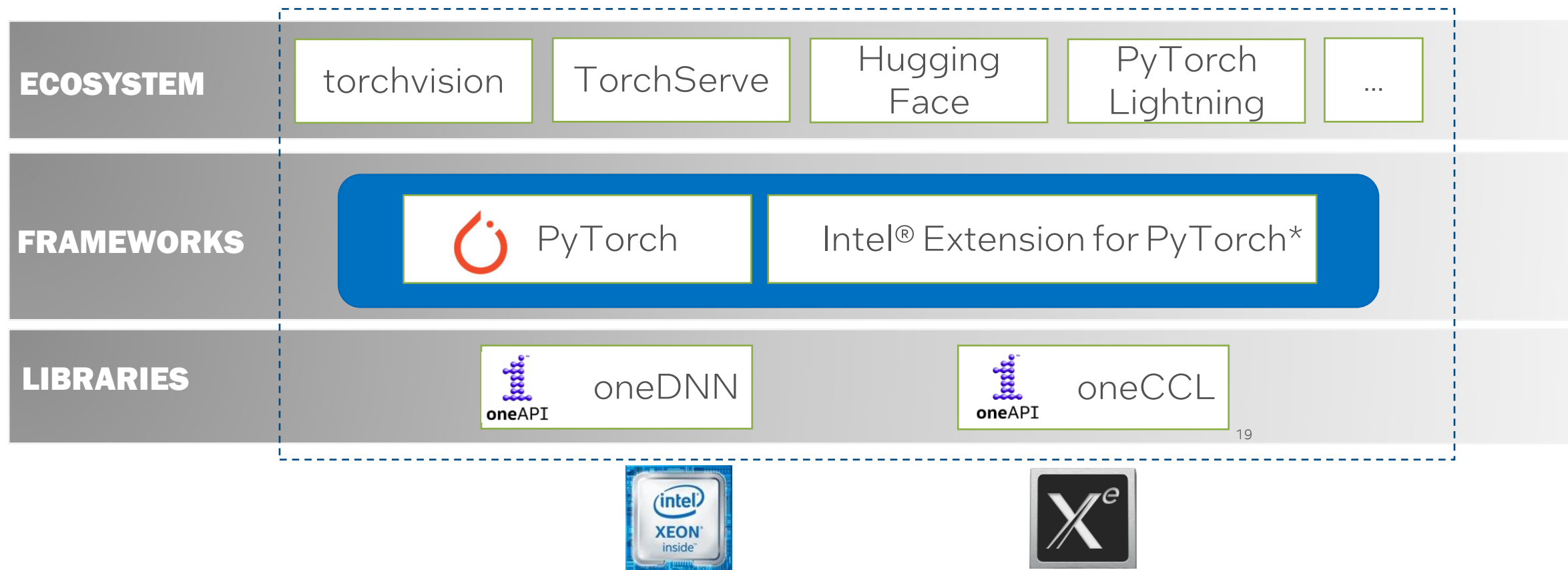
For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.
See backup for configuration details.



Deep Learning

Intel® Optimization for PyTorch

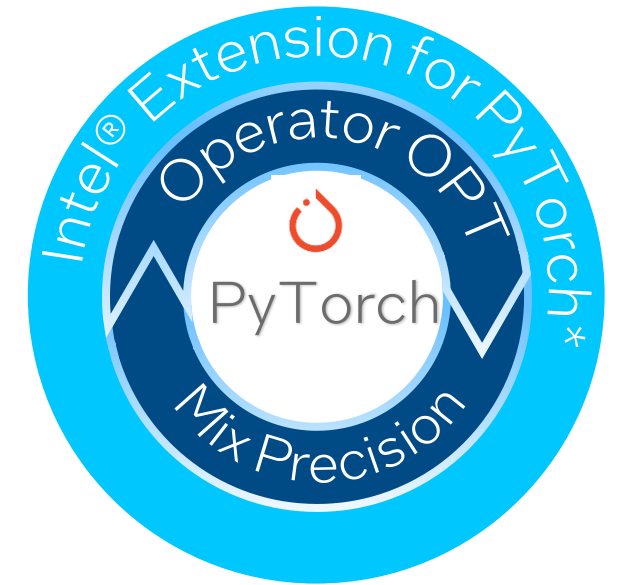
Intel® Optimization for PyTorch



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

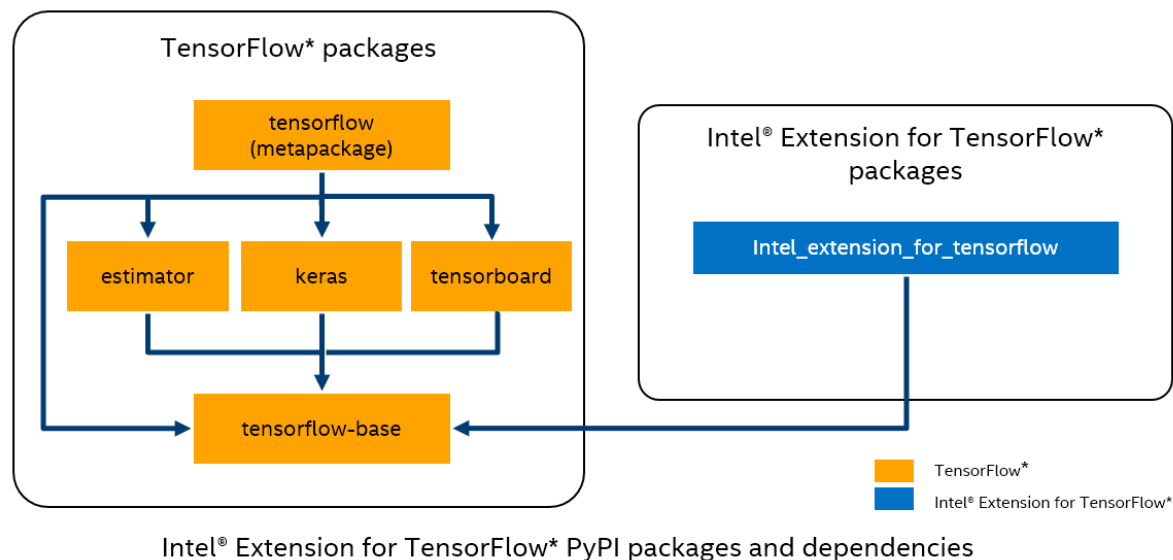
Intel® Extension for PyTorch* (IPEX)

- 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



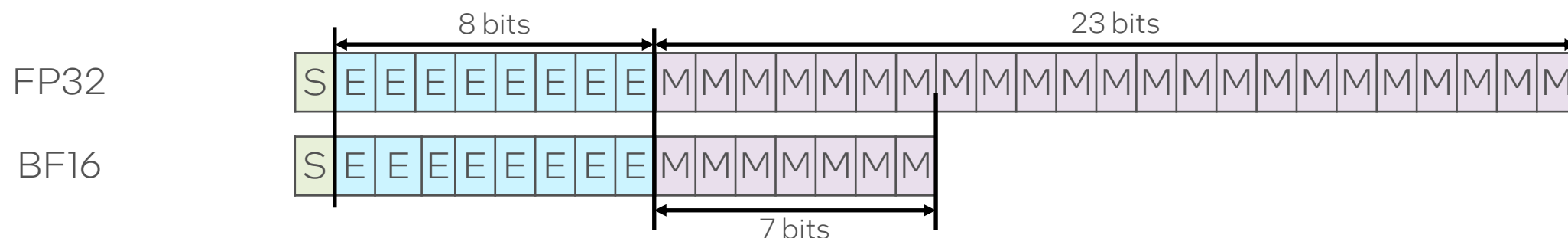
Intel® Extension for TensorFlow* (ITEX)

- 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



Building and Deploying with BF16

Low-precision Optimization – BF16



BF16 has the same range as FP32 but less precision due to 16 less mantissa bits. Running with 16 bits can give significant performance speedup.

Inference with Intel® Extension for PyTorch

Usage Example

Resnet50

```
import torch
import torchvision.models as models
##### code changes #####
import intel_extension_for_pytorch as ipex
##### code changes #####

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

##### code changes #####
model = model.to("xpu")
data = data.to("xpu")
model = ipex.optimize(model, dtype=torch.bfloat16)
##### code changes #####

with torch.no_grad():
    d = torch.rand(1, 3, 224, 224)
    ##### code changes #####
    d = d.to("xpu")
    with torch.xpu.amp.autocast(enabled=True, dtype=torch.bfloat16):
        ##### code changes #####
        model = torch.jit.trace(model, d)
        model = torch.jit.freeze(model)
        model(data)
```

BERT

*The .to("xpu") is needed for GPU only
**Use torch.cpu.amp.autocast() for CPU
***Channels last format is automatic

```
import torch
from transformers import BertModel
##### code changes #####
import intel_extension_for_pytorch as ipex
##### code changes #####

model = BertModel.from_pretrained(args.model_name)
model.eval()

vocab_size = model.config.vocab_size
batch_size = 1
seq_length = 512
data = torch.randint(vocab_size, size=[batch_size, seq_length])

##### code changes #####
model = model.to("xpu")
data = data.to("xpu")
model = ipex.optimize(model, dtype=torch.bfloat16)
##### code changes #####

with torch.no_grad():
    d = torch.randint(vocab_size, size=[batch_size, seq_length])
    ##### code changes #####
    d = d.to("xpu")
    with torch.xpu.amp.autocast(enabled=True, dtype=torch.bfloat16):
        ##### code changes #####
        model = torch.jit.trace(model, (d,), strict=False)
        model = torch.jit.freeze(model)

    model(data)
```


Training with Intel Extension for PyTorch Usage Example

```
import torch
import torchvision
import intel_extension_for_pytorch as ipex

LR = 0.001
DOWNLOAD = True
DATA = 'datasets/cifar10/'

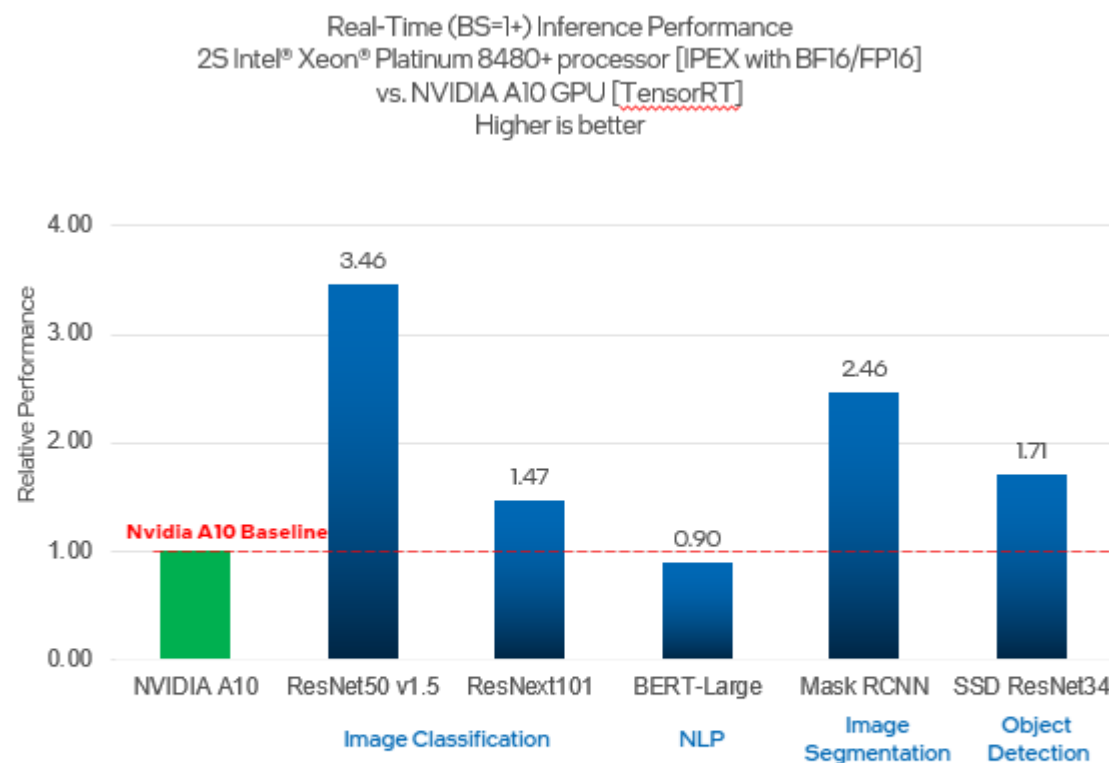
transform = torchvision.transforms.Compose([
    torchvision.transforms.Resize((224, 224)),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
train_dataset = torchvision.datasets.CIFAR10(
    root=DATA,
    train=True,
    transform=transform,
    download=DOWNLOAD,
)
train_loader = torch.utils.data.DataLoader(
    dataset=train_dataset,
    batch_size=128
)
```

```
model = torchvision.models.resnet50()
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr = LR, momentum=0.9)
model.train()

model, optimizer = ipex.optimize(model, optimizer=optimizer, dtype=torch.bfloat16)

for batch_idx, (data, target) in enumerate(train_loader):
    optimizer.zero_grad()
    with torch.cpu.amp.autocast():
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        print(batch_idx)
torch.save({
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
}, 'checkpoint.pth')
```

Intel Extension for PyTorch Performance



1.8x higher average* BF16/FP16 inference
performance vs Nvidia A10 GPU³

Benchmark data for the Intel® 4th Gen Xeon Scalable Processors can be found [here](#).

PyTorch AMX Training/Inference Code Samples

Training

GitHub: https://github.com/oneapi-src/oneAPI-samples/tree/master/AI-and-Analytics/Features-and-Functionality/IntelPyTorch_TrainingOptimizations_AMX_BF16

Trains a ResNet50 model with Intel Extension for PyTorch and shows performance speedup with AMX BF16

Inference

GitHub: https://github.com/oneapi-src/oneAPI-samples/tree/master/AI-and-Analytics/Features-and-Functionality/IntelPyTorch_InferenceOptimizations_AMX_BF16_INT8

Performs inference on ResNet50 and BERT with Intel Extension for PyTorch and shows performance speedup with AMX BF16 and INT8 over VNNI INT8

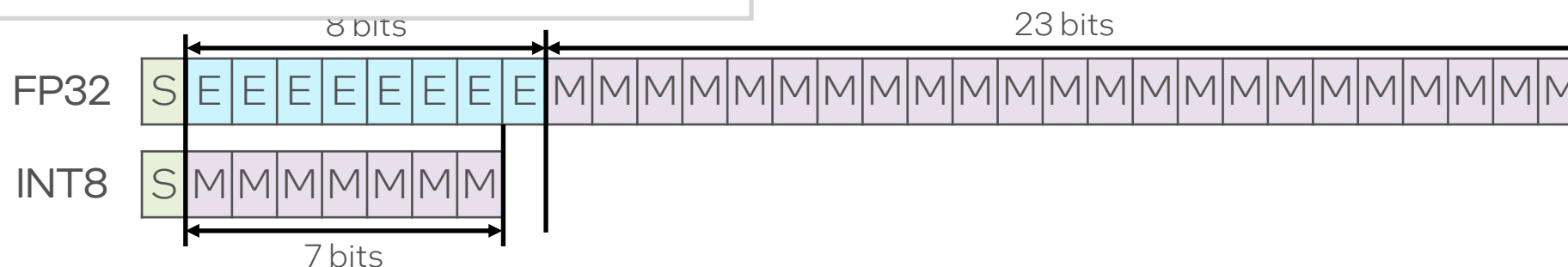
Deploying with INT8

Low-precision Optimization – INT8

- An approximation method
- The process of mapping values from a large set (e.g., continuous, FP64/FP32) to those with smaller set (e.g., countable, BF16, INT8)

- PyTorch quantization
- **IPEX quantization (with or w/o INC integration)**
- Inter Neural Compressor (INC)

- Significant performance increase with similar accuracy



TorchScript and torch.compile()

TorchScript

- Converts PyTorch model into a graph for faster execution
- torch.jit.trace() traces and records all operations in the computational graph; requires a sample input
- torch.jit.script() parses the Python source code of the model and compiles the code into a graph; sample input not required

torch.compile() – in BETA

- Makes PyTorch code run faster by just-in-time (JIT)-compiling PyTorch code into optimized kernels

Resnet50

```
import torch
import torchvision.models as models

model = models.resnet50(weights='ResNet50_Weights.DEFAULT')
model.eval()
data = torch.rand(1, 3, 224, 224)

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

with torch.no_grad(), torch.cpu.amp.autocast():
    model = torch.jit.trace(model, torch.rand(1, 3, 224, 224))
    model = torch.jit.freeze(model)

model(data)
```

Intel® Neural Compressor

Intel® Neural Compressor



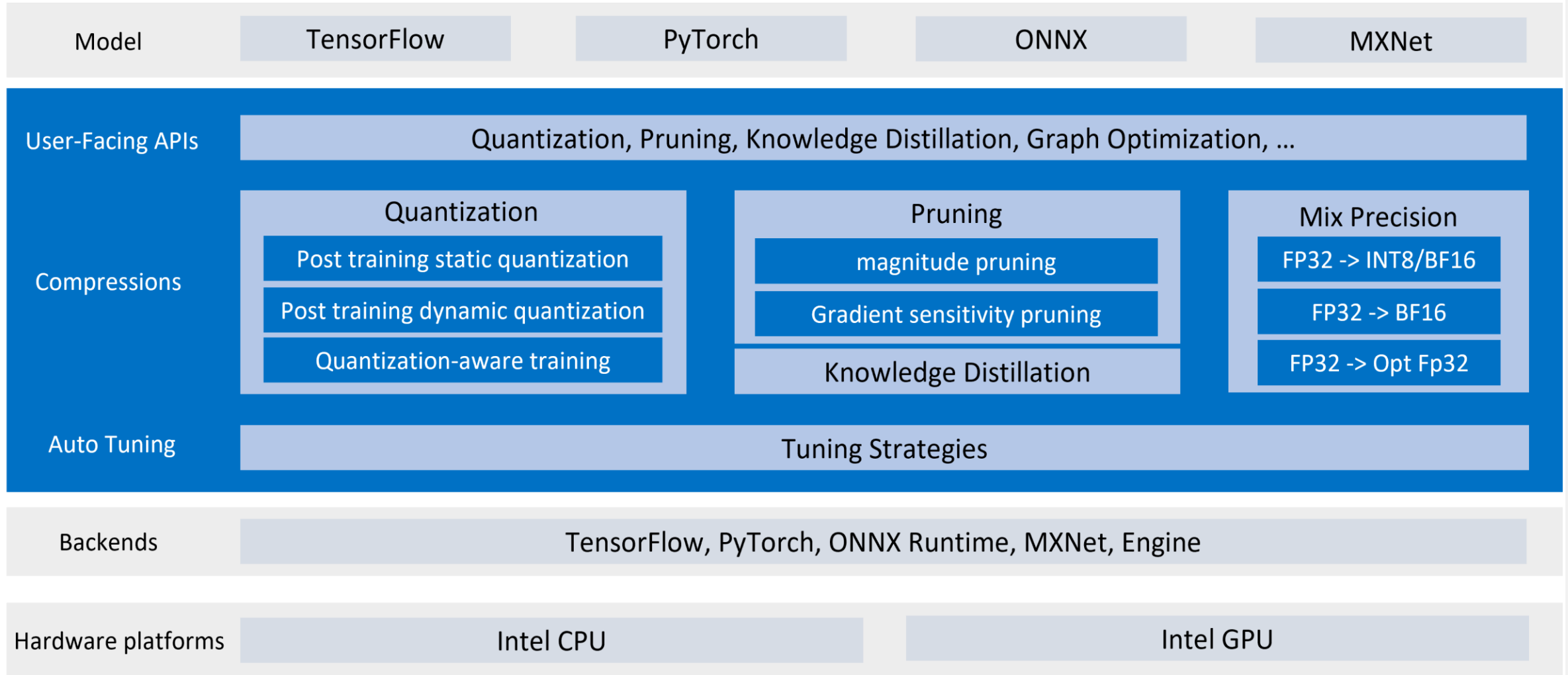
The Intel® Neural Compressor is an open-source Python library, which delivers unified interfaces across multiple deep learning frameworks for popular network optimization technologies.



It supports quantization, mixed precision, pruning, knowledge distillation, and graph optimizations, and uses accelerations by Intel Deep Learning Boost or Intel Advanced Matrix Extension in SPR.

Intel® Neural Compressor

Intel® Neural Compressor
Architecture



Use cases

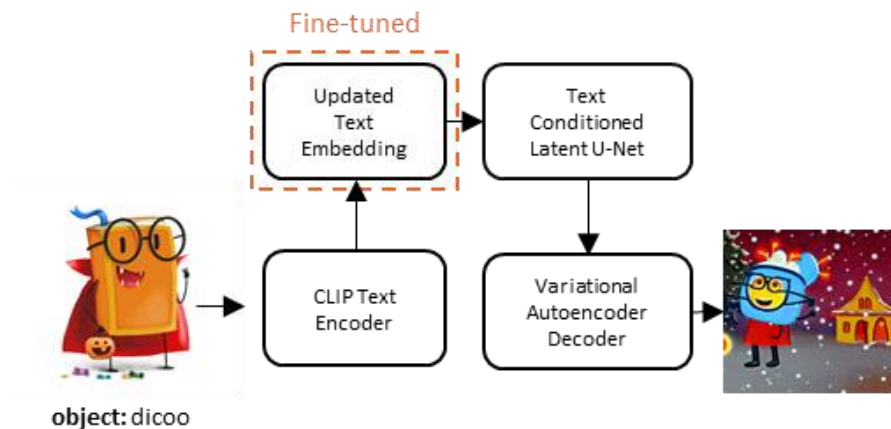
Stable Diffusion(SD) Use case

Create Your Own Stable Diffusion

Few-shot Fine-tuning

5 minutes

4 SPR nodes



Accelerated Stable Diffusion Inference

Low-precision Inference

5 seconds

1 SPR chip

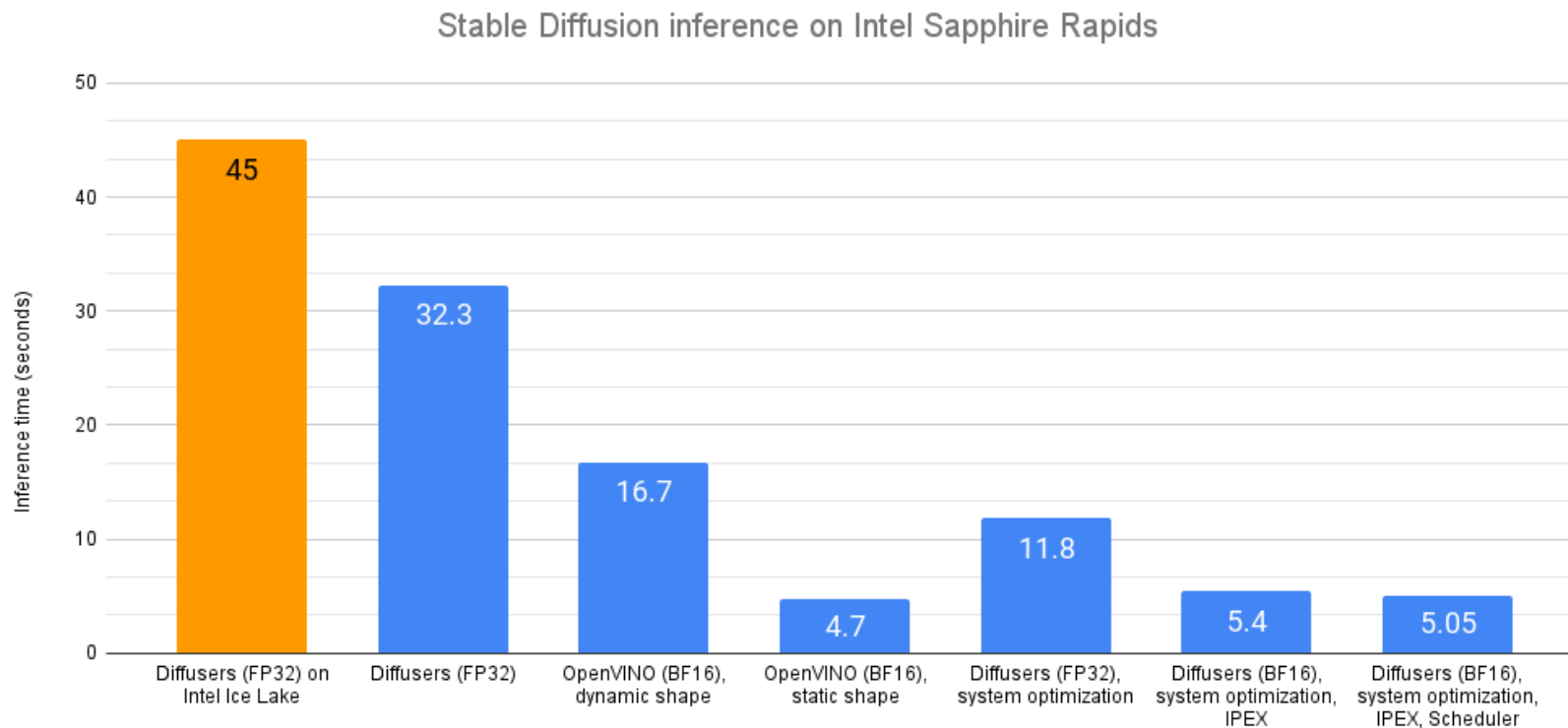


Optimizations upstreamed to Hugging Face Diffusers and Optimum-Intel

Try SD demo here: <https://huggingface.co/spaces/Intel/Stable-Diffusion-Side-by-Side>

HF Blog: SD @ Intel

This article compares different options for inference and acceleration of Stable Diffusion pipeline on Intel CPUs. Overall, usage of BF16 instead of FP32 allows to improve inference in 3 times!



Model Zoo for Intel® Architecture

Available on GitHub

Runs out-of-the-box

PyTorch use cases

- Image Recognition, Image Segmentation, Language Modeling/Translation, Object Detection, Recommendation, Text-to-Speech, Shot Boundary Detection, AI Drug Design
- Supported on dGPU: INT8 inference on ResNet50v1.5, SSD-MobileNet, Yolo V4

[Model Zoo: https://github.com/intelAI/models/tree/master](https://github.com/intelAI/models/tree/master)

Image Recognition				
Model	Framework	Mode	Model Documentation	Benchmark/Test Dataset
DenseNet169	TensorFlow	Inference	FP32	ImageNet 2012
Inception V3	TensorFlow	Inference	Int8 FP32	ImageNet 2012
Inception V4	TensorFlow	Inference	Int8 FP32	ImageNet 2012
MobileNet V1*	TensorFlow	Inference	Int8 FP32 BFloat16	ImageNet 2012
ResNet 101	TensorFlow	Inference	Int8 FP32	ImageNet 2012
ResNet 50	TensorFlow	Inference	Int8 FP32	ImageNet 2012
ResNet 50v1.5	TensorFlow	Inference	Int8 FP32 BFloat16 dGPU Int8	ImageNet 2012
ResNet 50v1.5 Sapphire Rapids	TensorFlow	Inference	Int8 FP32 BFloat16	ImageNet 2012
ResNet 50v1.5	TensorFlow	Training	FP32 BFloat16	ImageNet 2012
Inception V3	TensorFlow Serving	Inference	FP32	Synthetic Data
ResNet 50v1.5	TensorFlow Serving	Inference	FP32	Synthetic Data
GoogLeNet	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
Inception v3	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
MNASNet 0.5	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
MNASNet 1.0	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
ResNet 50	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
ResNet 50	PyTorch	Training	FP32 BFloat16	ImageNet 2012
ResNet 101	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
ResNet 152	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
ResNext 32x4d	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
ResNext 32x16d	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
VGG-11	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
VGG-11 with batch normalization	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
Wide ResNet-50-2	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
Wide ResNet-101-2	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
ResNet 50 v1.5	PyTorch	Inference	dGPU Int8	ImageNet 2012


Recipe for Intel® Optimizations

Easy Recipe for faster Intel® Optimizations with IPEX

- Add IPEX
- Add some Warmup steps for oneDNN initialization
- Utilize AMX or XMX instruction sets with efficient bfloat16 data type
- Utilize graph mode with TorchScript
- Quantize model to INT8
- Runtime optimizations using ipexrun
- Distributed training with oneCCL/ DDP/Horovod
- Profile with oneDNN verbose / Pytorch Profiler / VTune for further analysis.

Conclusion

Data Preprocessing


MODIN
38X faster ETL
compared to Pandas


Model Training

 PyTorch
1.55X faster
DLRM training (FP32 vs
BF16)


100X faster
training compared to
stock scikit-learn

Model Inference

 PyTorch
2.8X faster DLRM
inference (FP32 vs
BF16)


TensorFlow
2.8X faster DLRM
inference (FP32 vs
BF16)


21X faster
prediction


4.5X faster
prediction



You can contact us for any help:

Vladimir Kilyazov

AI Software Solutions Engineer

vladimir.kilyazov@intel.com

In: Vladimir Kilyazov

Intel Developer Cloud experts will
be also available on Discord!