AI on Intel® Architecture

Vladimir Kilyazov, AI Software Solutions Engineer

intel®

Notices & 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.

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

<u>4th Generation Intel® Xeon® Scalable Processors</u>

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.

<u>3rd Generation Intel® Xeon® Scalable Processors</u>

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

Agenda

- Introduction to Intel® AI stack
- Classic Machine Learning Libraries
- Deep Learning Frameworks
- BF16 and INT8 training and inference
- Under the hood of oneDNN & IPEX
- Recipe for Intel® Optimizations
- Use cases
- Conclusion

Introduction to Intel® AI stack

The AI Pipeline Runs on Intel

Data	Model	Deploy
Majority of data prep already happens on Intel® Xeon® processors and that continues today	**Traditional Machine Learning** Intel Xeon's large memory and fast cores excel running ML workloads **Deep Learning** GPUs are well-suited, Intel Xeon's built-in AI capabilities deliver compelling TCO	**70%** of inferencing is run on **Intel Xeon Processors**[1]

Database, Data Warehouse, Data Lake, Streaming Data, Feature Store, Model Registry

1 Based on Intel market modeling of the worldwide installed base of data center servers running AI Inference workloads as of December 2021.

Intel® oneAPI Toolkits

oneAPI

Intel® oneAPI Base Toolkit	intel. oneAPI BASE TOOLKIT	A core set of high-performance libraries and tools for building C++, SYCL, C/OpenMP, and Python applications

Add-on Domain-specific Toolkits

For HPC developers

Intel® oneAPI Tools for HPC
Deliver fast Fortran, OpenMP & MPI applications that scale

For visual creators, scientists & engineers

Intel® oneAPI Rendering Toolkit
Accelerate visual compute, deliver high-performance, high-fidelity visualization applications.

For edge & IoT developers

Intel® oneAPI Tools for IoT
Build efficient, reliable solutions that run at network's edge

Toolkits powered by oneAPI

For AI developers & data scientists

Intel® AI Analytics Toolkit
Accelerate machine learning & data science pipelines end-to-end with optimized DL & ML frameworks & high-performing Python libraries

OpenVINO

For deep learning inference developers

Intel® OpenVINO™ toolkit
Deploy high performance inference & applications from edge to cloud

Download at **intel.com/oneAPI**
Or visit Intel® DevCloud for oneAPI

Intel AI Software

Data → Model → Deploy

AI Platforms, MLOPs
(Productivity)

AI Libraries, Tools, Frameworks
(Most AI developers operate here)

Low-level Performance Libraries, Compilers, Kernel
(Most hardware enabling and performance tuning happens here)

Engineer Data → Create Machine Learning & Deep Learning Models → Deploy

AI Platforms & Kits

Most Popular Tools and Frameworks

Performance Libraries

intel ATOM | intel CORE | intel XEON | intel ARC GRAPHICS | intel DATA CENTER GPU | intel habana

Note: not all components are necessarily compatible with all other components in other layers

Engineer Data | Create Machine Learning & Deep Learning Models | Deploy

AI Platforms & Kits

Most Popular Tools and Frameworks

SYCLomatic | oneDAL | oneDNN | oneCCL | oneMKL | SynapseAI™

intel ATOM | intel CORE | intel XEON | intel ARC GRAPHICS | intel DATA CENTER GPU | intel habana

Note: not all components are necessarily compatible with all other components in other layers

oneDAL – Intel oneAPI Data Analytics Library, oneDNN – Intel oneAPI Deep Neural Networks Library, oneCCL – Intel oneAPI Collective Communications Library, oneMKL - Intel oneAPI Math Kernel Library
AVX – Advanced Vector Extensions, VNNI – Vector Neural Network Instructions, AMX – Advanced Matrix Extensions, XMX – Xe Matrix Extensions

Engineer Data → Create Machine Learning & Deep Learning Models → Deploy

AI Platforms & Kits

Data Analytics Scale

MODIN
SciPy
pandas
NumPy

Optimized Frameworks and Middleware

TensorFlow
PyTorch
mxnet
PaddlePaddle
scikit learn
ONNX
LightGBM
XGBoost
CatBoost

Optimize Models

Automate Model Tuning AutoML
Automate Low-Precision Optimization
SigOpt
Intel Neural Compressor

w/ Intel Optimizations

SYCLomatic | oneDAL | oneDNN | oneCCL | oneMKL | SynapseAI™

intel ATOM | intel CORE | intel XEON | intel ARC GRAPHICS | intel DATA CENTER GPU | intel habana

Note: not all components are necessarily compatible with all other components in other layers

oneDAL – Intel oneAPI Data Analytics Library, oneDNN – Intel oneAPI Deep Neural Networks Library, oneCCL – Intel oneAPI Collective Communications Library, oneMKL - Intel oneAPI Math Kernel Library
AVX – Advanced Vector Extensions, VNNI – Vector Neural Network Instructions, AMX – Advanced Matrix Extensions, XMX – Xe Matrix Extensions

Engineer Data | Create Machine Learning & Deep Learning Models | Deploy

Accelerate End-to-End Data Science and AI — AI Analytics Toolkit

Data Analytics Scale

MODIN, SciPy, pandas, NumPy

Optimized Frameworks and Middleware

TensorFlow, PyTorch, mxnet, PaddlePaddle, scikit learn, ONNX, LightGBM, XGBoost, CatBoost

Optimize Models

Automate Model Tuning AutoML

Automate Low-Precision Optimization

SigOpt

Intel Neural Compressor

w/ Intel Optimizations

SYCLomatic | oneDAL | oneDNN | oneCCL | oneMKL | SynapseAI™

intel ATOM | intel CORE | intel XEON | intel ARC GRAPHICS | intel DATA CENTER GPU | intel habana

Note: not all components are necessarily compatible with all other components in other layers

oneDAL – Intel oneAPI Data Analytics Library, oneDNN – Intel oneAPI Deep Neural Networks Library, oneCCL – Intel oneAPI Collective Communications Library, oneMKL - Intel oneAPI Math Kernel Library
AVX – Advanced Vector Extensions, VNNI – Vector Neural Network Instructions, AMX – Advanced Matrix Extensions, XMX – Xe Matrix Extensions

Engineer Data → Create Machine Learning & Deep Learning Models → Deploy

Connect AI to Big Data | Apache Spark | BigDL (previously "Analytics Zoo")

Accelerate End-to-End Data Science and AI | AI Analytics Toolkit

Data Analytics Scale

MODIN, SciPy, pandas, NumPy

Optimized Frameworks and Middleware

TensorFlow, PyTorch, mxnet, PaddlePaddle, scikit learn, ONNX, LightGBM, XGBoost, CatBoost

Optimize Models

Automate Model Tuning AutoML — SigOpt

Automate Low-Precision Optimization — Intel Neural Compressor

w/ Intel Optimizations

SYCLomatic | oneDAL | oneDNN | oneCCL | oneMKL | SynapseAI™

intel ATOM | intel CORE | intel XEON | intel ARC GRAPHICS | intel DATA CENTER GPU | intel habana

Note: not all components are necessarily compatible with all other components in other layers

oneDAL – Intel oneAPI Data Analytics Library, oneDNN – Intel oneAPI Deep Neural Networks Library, oneCCL – Intel oneAPI Collective Communications Library, oneMKL - Intel oneAPI Math Kernel Library
AVX – Advanced Vector Extensions, VNNI – Vector Neural Network Instructions, AMX – Advanced Matrix Extensions, XMX – Xe Matrix Extensions

Engineer Data → Create Machine Learning & Deep Learning Models → Deploy

Connect AI to Big Data | Spark | BigDL (previously "Analytics Zoo")

Accelerate End-to-End Data Science and AI | AI Analytics Toolkit

Data Analytics Scale

MODIN
SciPy
pandas
NumPy

Optimized Frameworks and Middleware

TensorFlow
PyTorch
mxnet
PaddlePaddle
scikit learn
ONNX
LightGBM
XGBoost
CatBoost

Optimize Models

Automate Model Tuning AutoML

Automate Low-Precision Optimization

SigOpt

Intel Neural Compressor

OpenVINO™ Toolkit

Write Once

Deploy Auto-Optimized

Anywhere

w/ Intel Optimizations

SYCLomatic | oneDAL | oneDNN | oneCCL | oneMKL | SynapseAI™

intel ATOM | intel CORE | intel XEON | intel ARC GRAPHICS | intel DATA CENTER GPU | intel habana

Note: not all components are necessarily compatible with all other components in other layers

oneDAL – Intel oneAPI Data Analytics Library, oneDNN – Intel oneAPI Deep Neural Networks Library, oneCCL – Intel oneAPI Collective Communications Library, oneMKL - Intel oneAPI Math Kernel Library
AVX – Advanced Vector Extensions, VNNI – Vector Neural Network Instructions, AMX – Advanced Matrix Extensions, XMX – Xe Matrix Extensions

Engineer Data | Create Machine Learning & Deep Learning Models | Deploy

Container Repository
oneContainer

oneAPI powered
AI Reference Kits

MLOps
Cnvrg.io

Developer Sandbox
Intel® Developer Cloud

Annotation/Training/Optimization
Intel® GETi

Connect AI to Big Data — Apache Spark — BigDL (previously "Analytics Zoo")

Accelerate End-to-End Data Science and AI — AI Analytics Toolkit

OpenVINO™ Toolkit

Write Once

Deploy Auto-Optimized

Anywhere

Data Analytics Scale

MODIN, SciPy, pandas, NumPy

Optimized Frameworks and Middleware

TensorFlow, PyTorch, mxnet, PaddlePaddle, scikit learn, ONNX, LightGBM, XGBoost, CatBoost

Optimize Models

Automate Model Tuning AutoML

Automate Low-Precision Optimization

SigOpt

Intel Neural Compressor

w/ Intel Optimizations

SYCLomatic | oneDAL | oneDNN | oneCCL | oneMKL | SynapseAI™

intel ATOM | intel CORE | intel XEON | intel ARC GRAPHICS | intel DATA CENTER GPU | intel habana

Note: not all components are necessarily compatible with all other components in other layers

oneDAL – Intel oneAPI Data Analytics Library, oneDNN – Intel oneAPI Deep Neural Networks Library, oneCCL – Intel oneAPI Collective Communications Library, oneMKL - Intel oneAPI Math Kernel Library
AVX – Advanced Vector Extensions, VNNI – Vector Neural Network Instructions, AMX – Advanced Matrix Extensions, XMX – Xe Matrix Extensions

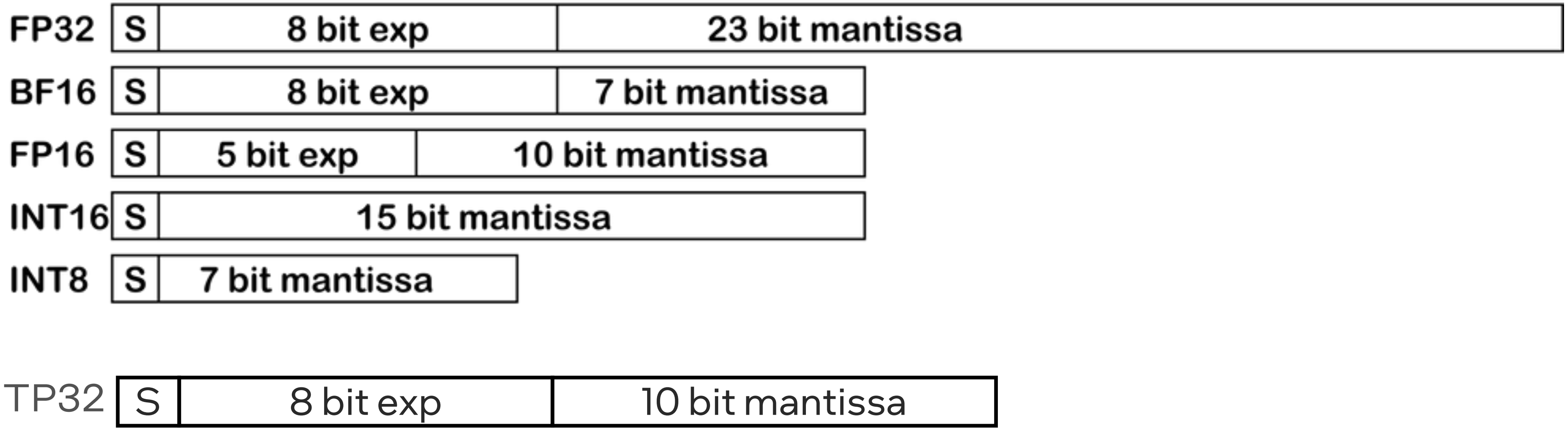
Data Precision and why it's important for Hardware optimizations

Data Precision

- Data precision:

  Number of bits used to store numerical values in memory

- Commonly found types of precision in Deep Learning:



Ref. https://www.intel.com/content/www/us/en/developer/videos/how-to-use-new-built-in-acceleration-engines.html#gs.wige3o

Lower Precision – Summary

LOWER MEMORY BANDWIDTH

LOWER STORAGE

HIGHER PERFOR-MANCE

+

MIN. ACCURACY LOSS

INT8/BF16 on Artificial intelligence/Machine Learning

- FP32 is the default datatypes used in AI/ML for inference, which has a high memory footprint and higher latency.

- Low-precision models are faster in computation. To optimize and support these:
  - HW needs special features/instructions
  - Intel provide those in the form of Intel AMX/Intel XMX.

- SYCL Joint Matrix is the coding abstraction to invoke Intel AMX/Intel XMX, which ensures portability and performance of the code

Ref. https://www.intel.com/content/www/us/en/developer/videos/how-to-use-new-built-in-acceleration-engines.html#gs.wige3o

Introduction to Intel® Advanced Matrix Extension and Intel® Xe Matrix Extensions

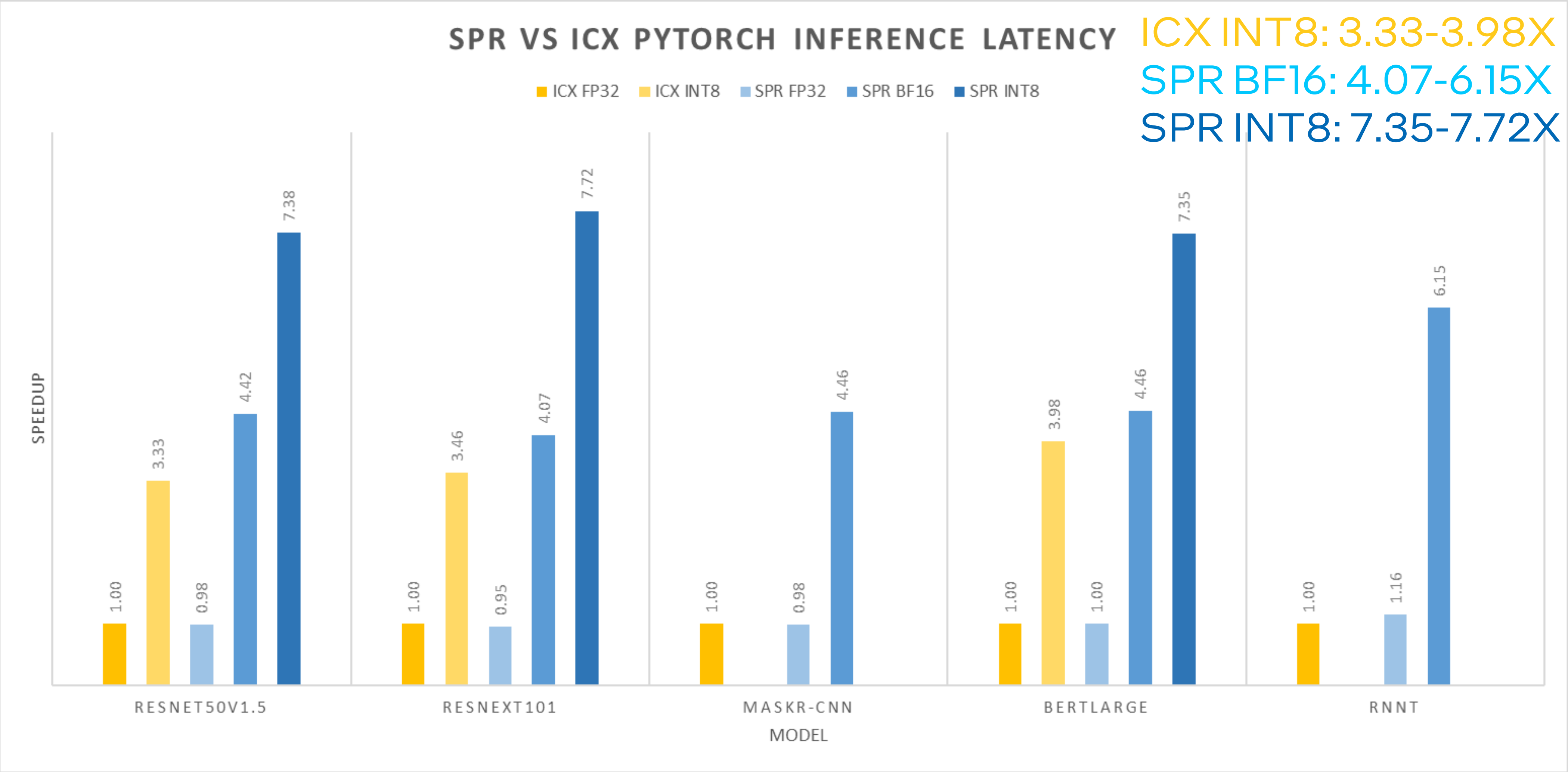
Instruction Set	**Hardware support**	Description
Intel® AMX	Intel® Xeon 4th Generation Scalable CPUs (Formerly code-named Sapphire Rapids)	Intel® Advanced Matrix Extension are extensions to the x86 instruction set architecture (ISA) for microprocessors using 2-dimensional registers called tiles upon which accelerators can perform operations. Supports INT8/BF16
Intel® XMX	Intel® Data Center GPU Max (Formerly code-named Ponte Vecchio) or Intel® Data Center GPU Flex Series	Intel® Xe Matrix Extensions also known as DPAS specializes in executing dot product and accumulate instructions on 2D systolic arrays Supports U8,S8,U4,S4,U2,S2, INT8 FP16, BF16, TF32

Both these Instruction Sets require Intel® oneAPI Base Toolkit 2023.0.0 and above for compilation

Ref. https://www.intel.com/content/www/us/en/developer/videos/how-to-use-new-built-in-acceleration-engines.html#gs.wige3o

PyTorch Benchmark: SPR vs ICX Inference (Batch Size = 1)

Inference latency speedup: the higher the better



ICX INT8: 3.33-3.98X
SPR BF16: 4.07-6.15X
SPR INT8: 7.35-7.72X

Benchmark data for the Intel® 4th Gen Xeon Scalable Processors can be found here.

Configurations slide at the end(slide 106).

Intel AI Software by Server Platform

- Intel® Xeon® Scalable Processor
- Intel® Data Center GPU
- Habana® Gaudi® Processors for DL

Category	Software	Open Source	Optimizations Upstreamed*	Intel Extension**	Intel Distribution	Intel Tool / Kit
Orchestration	Cnvrg.io	No				Xeon, Data Center GPU, Gaudi
Toolkits	AI Toolkit	Yes				Xeon, Data Center GPU
	BigDL	Yes				Xeon
	OpenVINO	Yes			Xeon, Data Center GPU	Xeon, Data Center GPU
Optimization	Neural Compressor	Yes				Xeon, Data Center GPU
	SigOpt	Yes				Xeon, Data Center GPU, Gaudi
DL Frameworks	TensorFlow	Yes	Xeon	Xeon, Data Center GPU	Xeon, Data Center GPU, Gaudi	
	PyTorch	Yes	Xeon	Xeon, Data Center GPU	Gaudi	
	ONNX	Yes	Xeon			
	PDPD	Yes	Xeon			
ML Frameworks	XGBoost	Yes	Xeon			
	Scikit-Learn	Yes		Xeon, Data Center GPU		
	CatBoost	Yes	Xeon			
	LightGBM	Yes	Xeon			
Data Preprocessing	Modin (for Pandas)	Yes	Xeon		Xeon	
	Intel® Distribution for Python	Yes			Xeon, Data Center GPU	
	Spark	Yes	Xeon	Xeon		

* Intel strives to upstream as many optimizations for as many hardware targets as soon as possible
** Access more Intel optimizations and target hardware support through API-compliant extensions

Machine Learning

Modin

Intel distribution of Modin

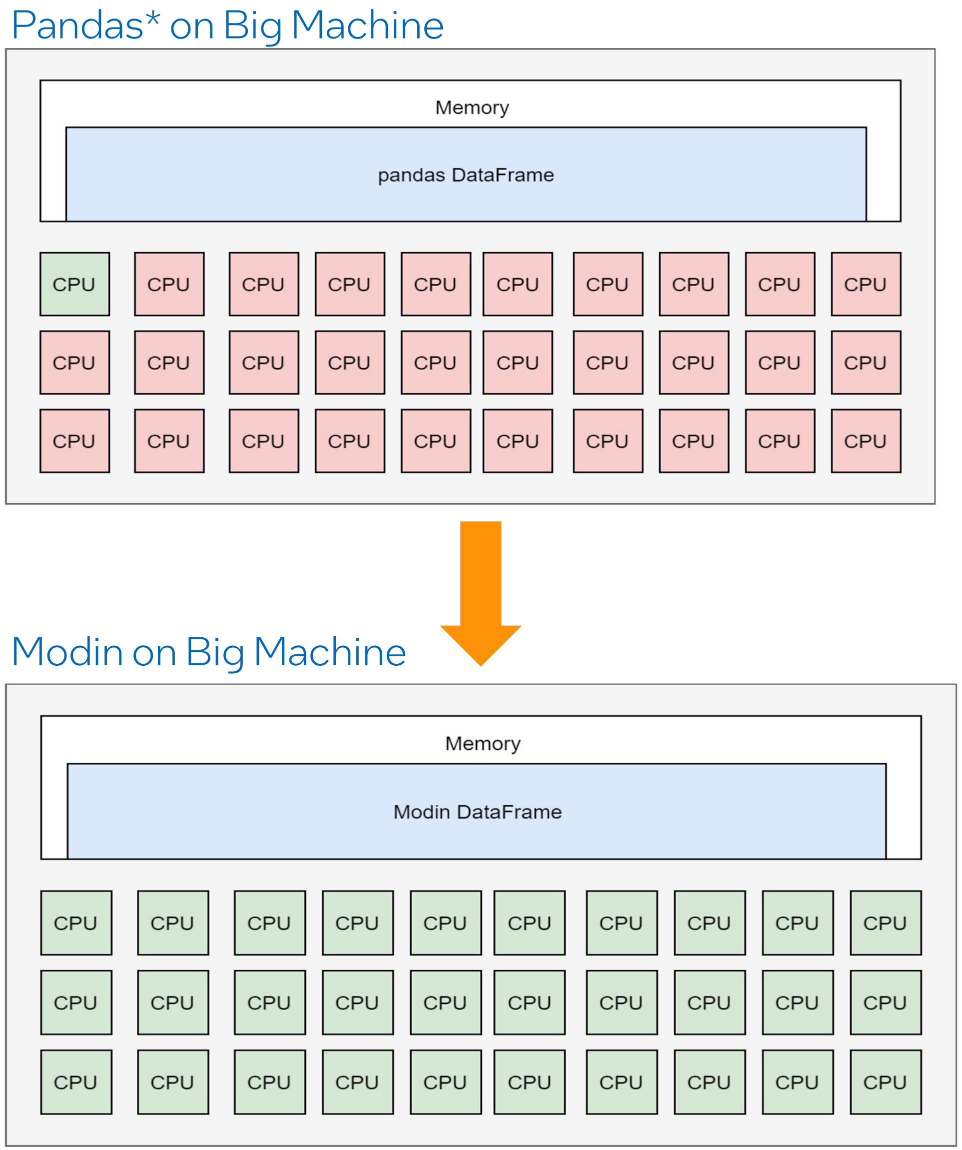
- Pandas is a Python package for data manipulation and analysis that offers data structures and operations for manipulating numerical tables and time series

- **Modin** = Pandas + Scalability
- As simple as **import modin.pandas as pd**

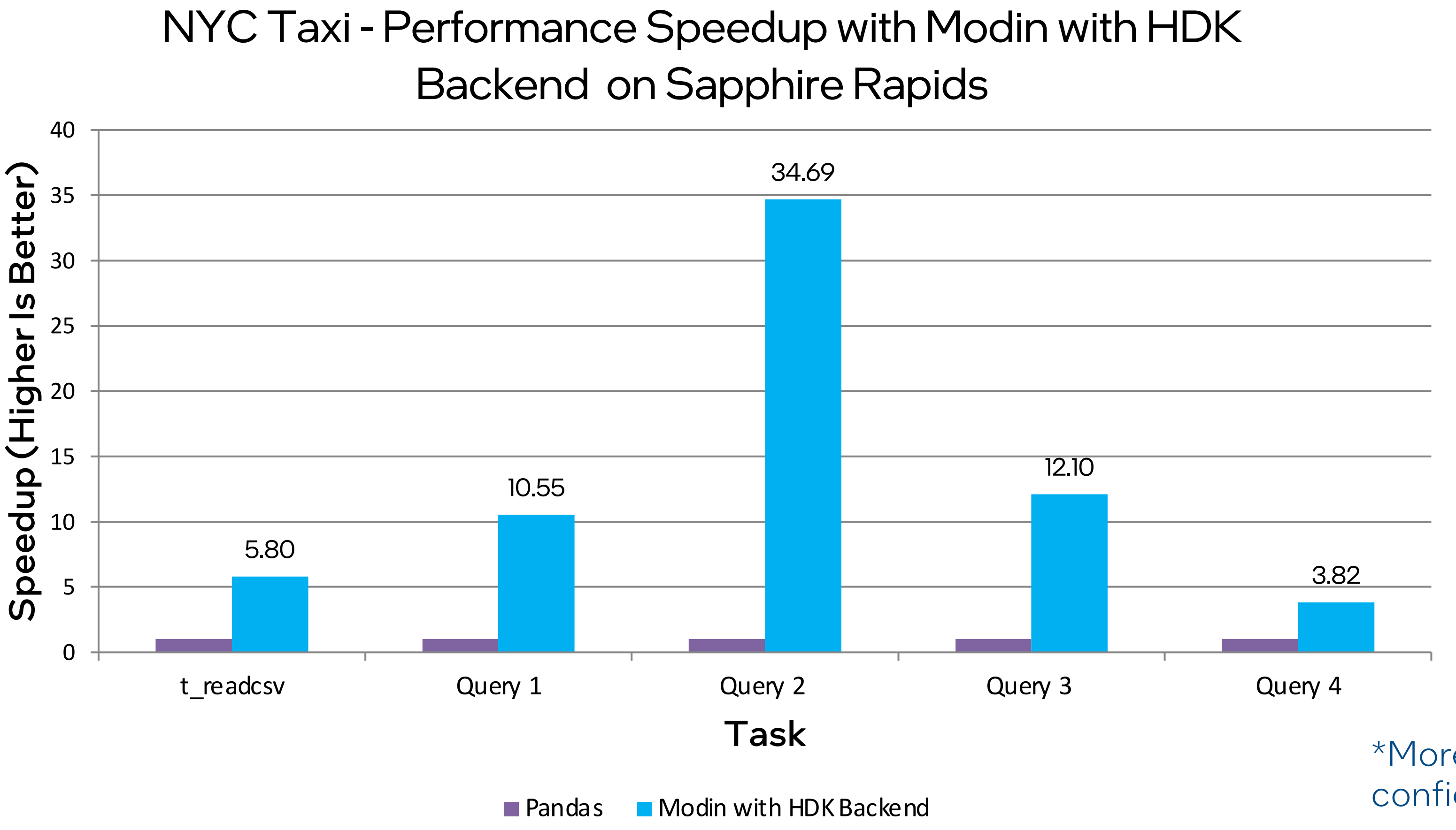
```
import pandas as pd
```

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



Intel® Distribution of Modin NYC Taxi Benchmark on SPR



*More detail on queries in configuration details

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/toddwschneider/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*

scikit learn Install User Guide API Examples More Go

scikit-learn

Machine Learning in Python

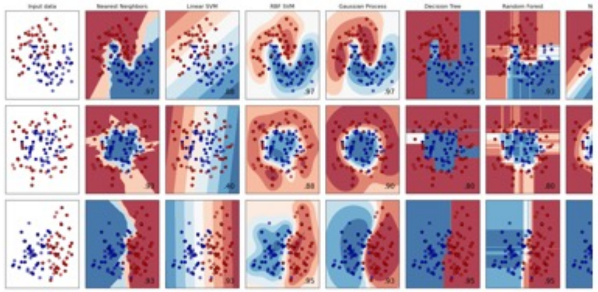
Getting Started | Release Highlights for 0.24 | GitHub

- 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.

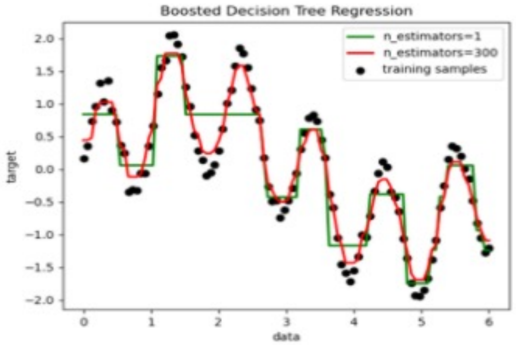
Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...



Regression

Predicting a continuous-valued attribute associated with an object.

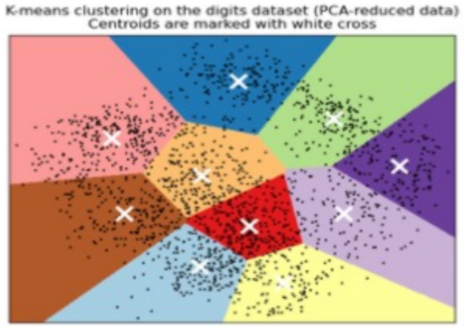
Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...



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
```

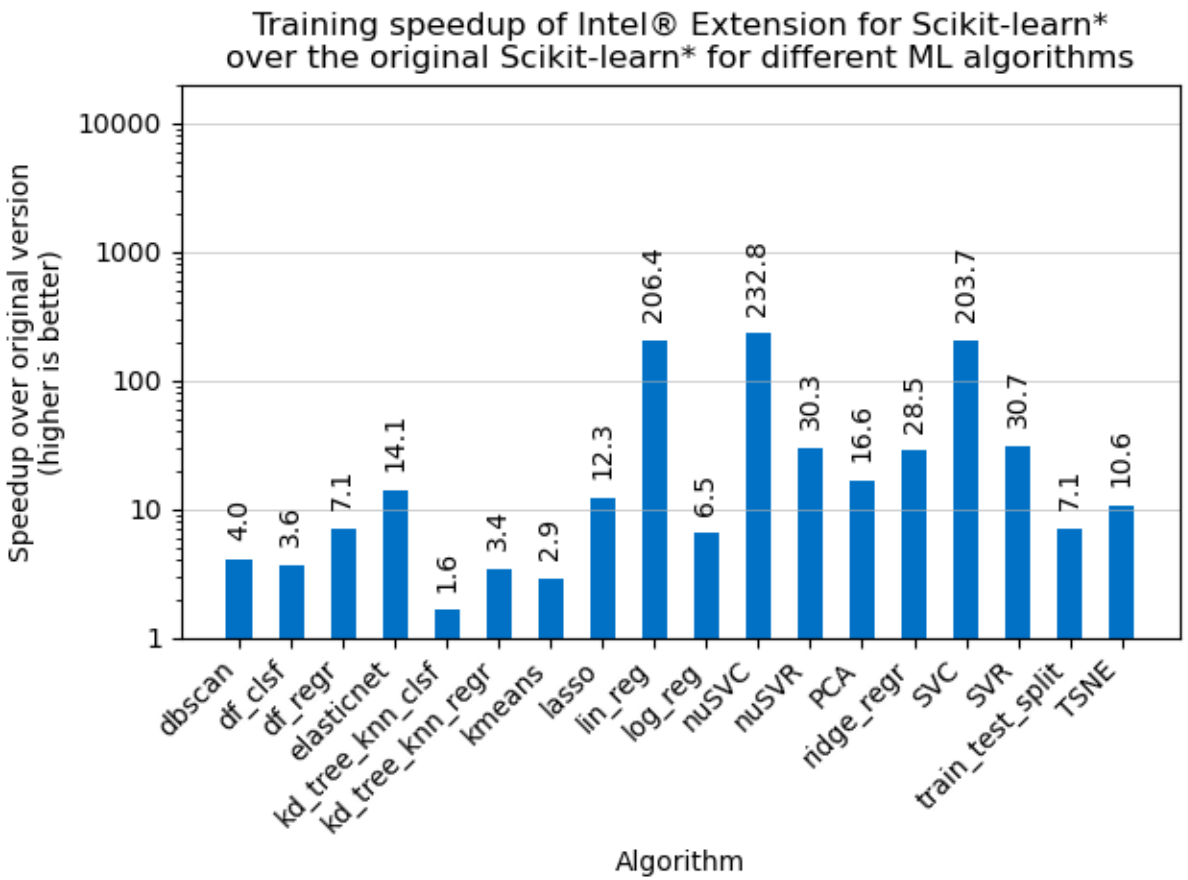
Same Code,
Same Behavior

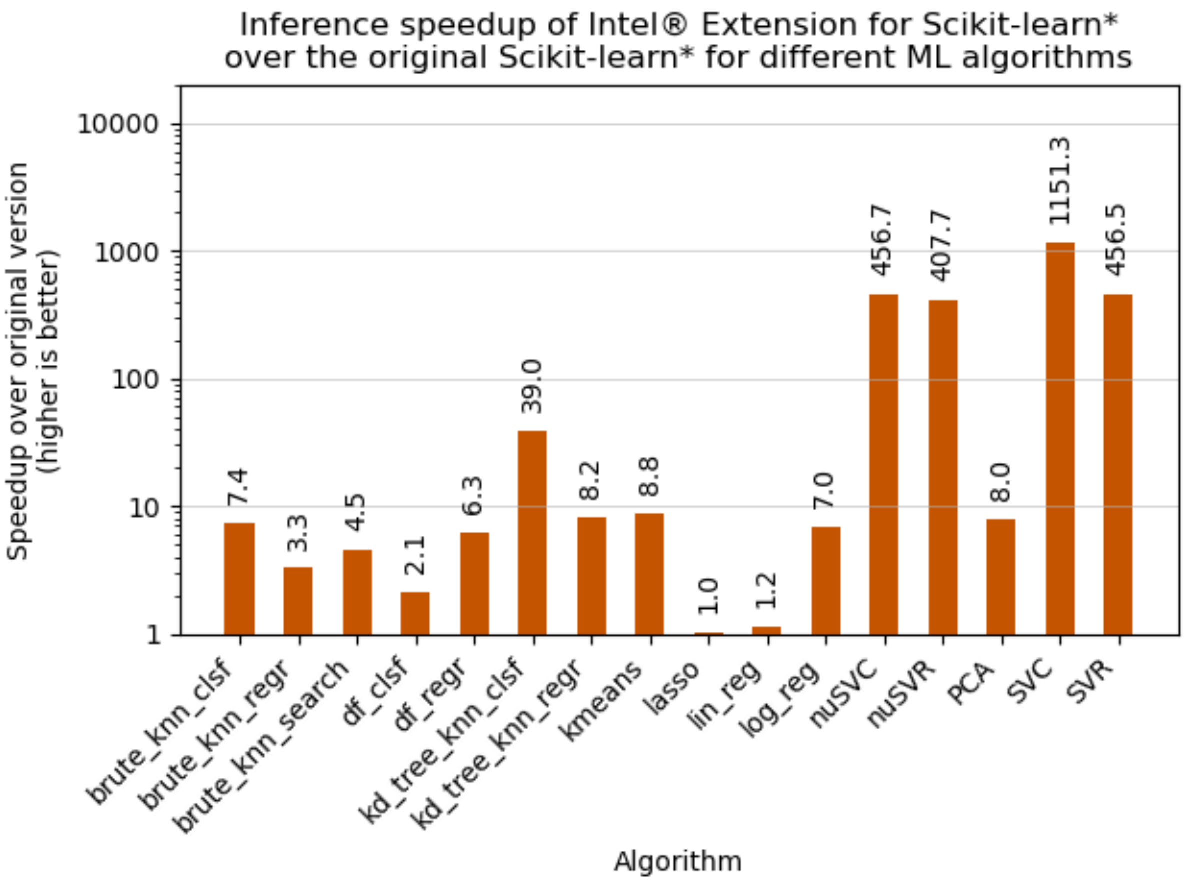
PASSED

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

Intel Confidential

Speedup summaries (float32 and float64 data combined)





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-intelex 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

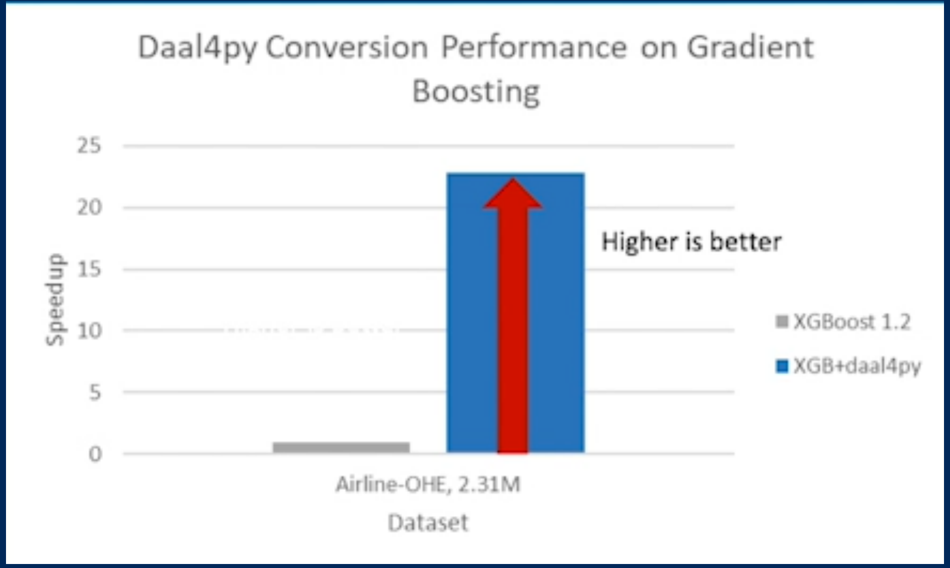
- XGBoost Training -> upstreamed
- XGBoost Inference -> have to switch to oneDAL backend with daal4py package
- Supported implementations: XGBoost, LightGBM, CatBoost

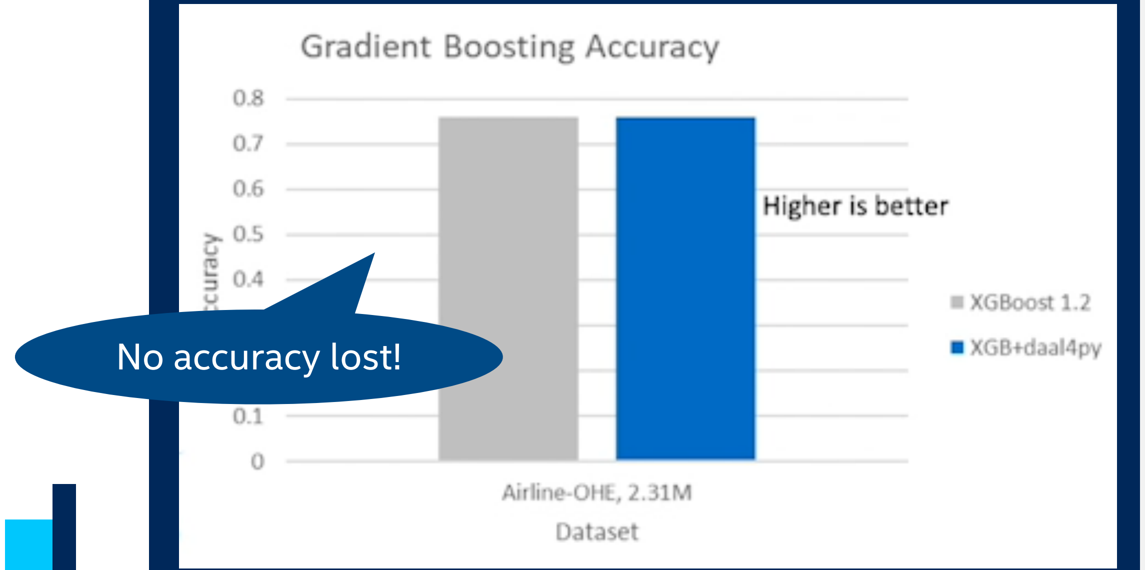
XGBoost* and LightGBM* Prediction Acceleration with Daal4Py

- 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

ECOSYSTEM

- torchvision
- TorchServe
- Hugging Face
- PyTorch Lightning
- ...

FRAMEWORKS

- PyTorch
- Intel® Extension for PyTorch*

LIBRARIES

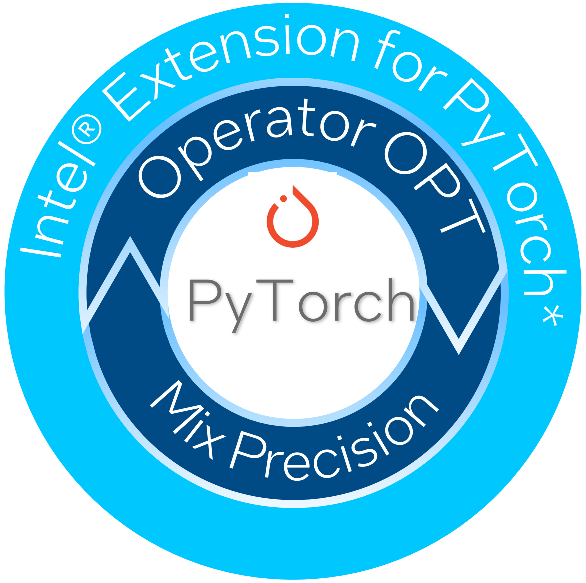
- oneAPI oneDNN
- oneAPI oneCCL

39

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

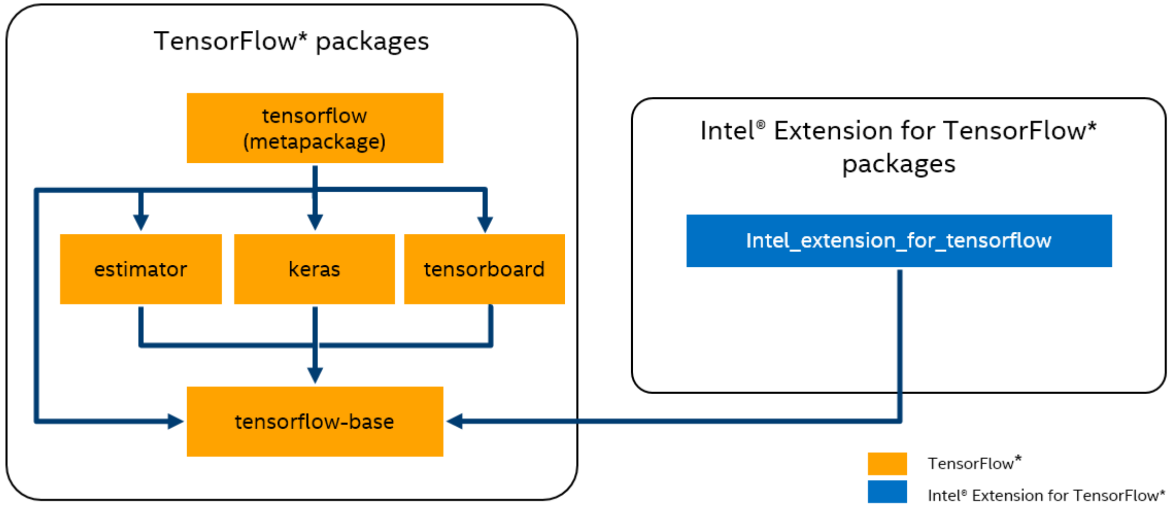
Intel® Extension for PyTorch* (IPEX)

- 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)

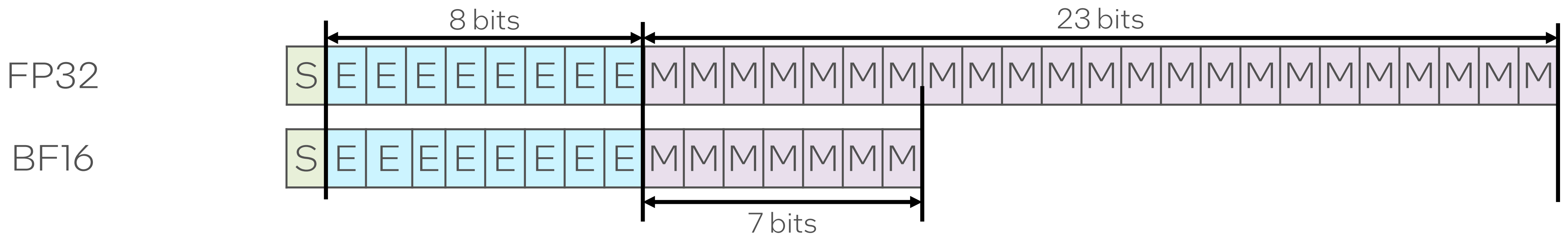
- Provide users with the up-to-date Intel software/hardware features
- Streamline the work to integrate oneDNN
- Unify user experiences on Intel CPU and GPU



Intel® Extension for TensorFlow* PyPI packages and dependencies

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.

https://www.intel.com/content/dam/develop/external/us/en/documents/bf16-hardware-numerics-definition-white-paper.pdf

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

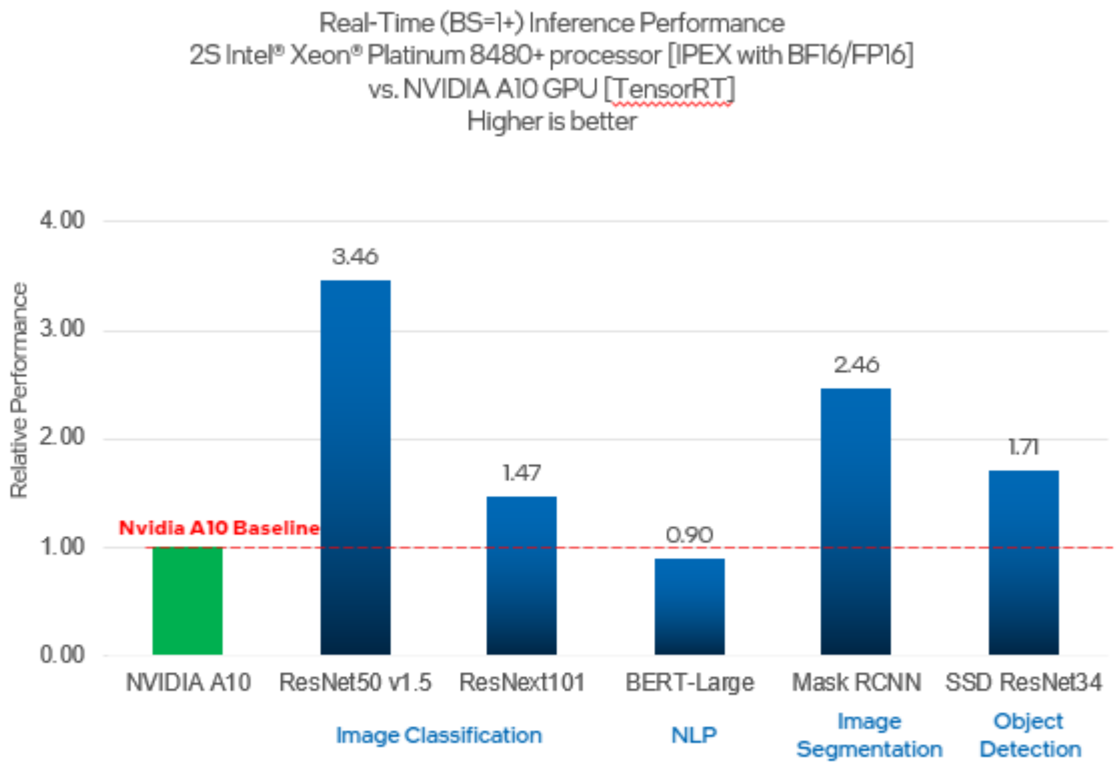
```
import torch
from transformers import BertModel
############# code changes ###############
import intel_extension_for_pytorch as ipex
############# code changes ###############

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

```python
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
)
```

```python
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 GPU[3]

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

What is Quantization?

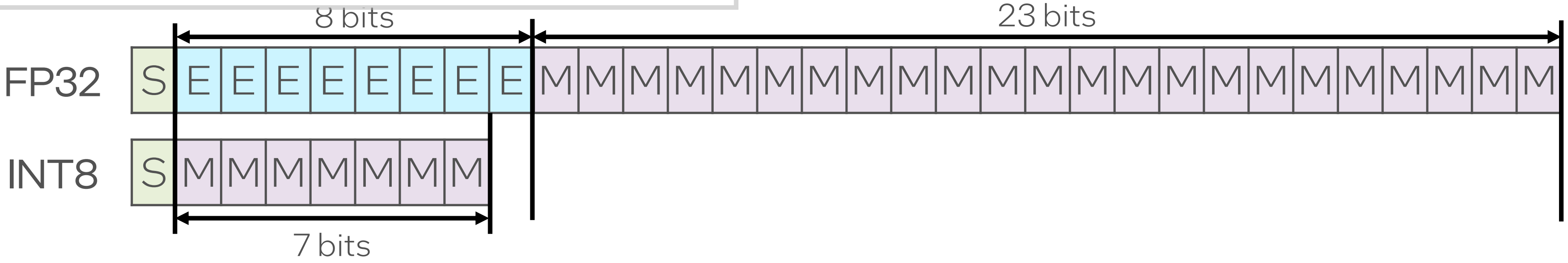
- An approximation method
- The process of mapping values from a large set (e.g., continuous, FP64/FP32) to those with smaller set (e.g., countable, BF16, INT8)

How to Quantize?

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

Why Quantization?

- 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

Resnet50

```python
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)
```

torch.compile() – in BETA

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

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

Model	TensorFlow · PyTorch · ONNX · MXNet
User-Facing APIs	Quantization, Pruning, Knowledge Distillation, Graph Optimization, …
Compressions	**Quantization**: Post training static quantization; Post training dynamic quantization; Quantization-aware training **Pruning**: magnitude pruning; Gradient sensitivity pruning **Knowledge Distillation** **Mix Precision**: FP32 -> INT8/BF16; FP32 -> BF16; FP32 -> Opt Fp32
Auto Tuning	Tuning Strategies
Backends	TensorFlow, PyTorch, ONNX Runtime, MXNet, Engine
Hardware platforms	Intel CPU · Intel GPU

Verifying That AMX Is Used

How to Check If AMX Is Enabled

- On bash terminal, enter the following command:
  - *cat /proc/cpuinfo*
- Check the “flags” section for amx_bf16, amx_int8
- Alternatively, you can use:
  - *lscpu | grep amx*
- If you do not see them, upgrade to Linux kernel 5.17 and above

```
Flags:                fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse s
se2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpu
id aperfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 s
se4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb cat_l3 cat_
l2 cdp_l3 invpcid_single intel_ppin cdp_l2 ssbd mba ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi flexpriority ept vpid ept_
ad fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm cqm rdt_a avx512f avx512dq rdseed adx smap avx512ifma clflus
hopt clwb intel_pt avx512cd sha_ni avx512bw avx512vl xsaveopt xsavec xgetbv1 xsaves cqm_llc cqm_occup_llc cqm_mbm_total cqm_
mbm_local split_lock_detect avx_vnni avx512_bf16 wbnoinvd dtherm ida arat pln pts hwp hwp_act_window hwp_epp hwp_pkg_req hfi
 avx512vbmi umip pku ospke waitpkg avx512_vbmi2 gfni vaes vpclmulqdq avx512_vnni avx512_bitalg tme avx512_vpopcntdq la57 rdp
id bus_lock_detect cldemote movdiri movdir64b enqcmd fsrm uintr avx512_vp2intersect md_clear serialize tsxldtrk pconfig arch
_lbr amx_bf16 avx512_fp16 amx_tile amx_int8 flush_l1d arch_capabilities
```

How to Check AMX Is Actually Used

- Generate oneDNN Verbose logs using guide and parser
- To enable verbosity, set environment variables:
  - export DNNL_VERBOSE=1
  - export DNNL_VERBOSE_TIMESTAMP=1
- Set a Python breakpoint RIGHT AFTER one iteration of training/inference

oneDNN Verbose Sample Output

Sample oneDNN Verbose Output

```
onednn_verbose,info,oneDNN v2.6.0 (commit 52b5f107dd9cf10910aaa19cb47f3abf9b349815)
onednn_verbose,info,cpu,runtime:OpenMP,nthr:32
onednn_verbose,info,cpu,isa:Intel AVX-512 with Intel DL Boost
onednn_verbose,info,gpu,runtime:none
onednn_verbose,info,prim_template:timestamp,operation,engine,primitive,implementation,prop_kind,memory_descriptors,attributes,auxiliary,problem_desc,exec_time
onednn_verbose,1678917979730.501953,exec,cpu,reorder,jit:uni,undef,src_f32::blocked:abcd:f0 dst_f32:p:blocked:Acdb16a:f0,attr-scratchpad:user ,,1x1x1x37,0.00292969
onednn_verbose,1678917979730.888916,exec,cpu,convolution,jit:avx512_core,forward_training,src_f32::blocked:abcd:f0 wei_f32:p:blocked:Acdb16a:f0 bia_undef::undef::f0 dst_f
onednn_verbose,1678917979732.105957,exec,cpu,reorder,jit:uni,undef,src_f32:p:blocked:aBcd16b:f0 dst_f32::blocked:abcd:f0,attr-scratchpad:user ,,1x1x1x48000,0.0649414
onednn_verbose,1678917980009.694092,exec,cpu,reorder,jit:uni,undef,src_f32::blocked:abc:f0 dst_f32::blocked:acb:f0,attr-scratchpad:user ,,1x60x305,0.00878906
onednn_verbose,1678917980011.387939,exec,cpu,convolution,brgconv:avx512_core,forward_training,src_f32::blocked:acb:f0 wei_f32::blocked:Acb32a:f0 bia_f32::blocked:a:f0 dst_
onednn_verbose,1678917980012.134033,exec,cpu,reorder,jit:uni,undef,src_f32::blocked:abc:f0 dst_f32::blocked:acb:f0,attr-scratchpad:user ,,1x1024x301,0.278076
onednn_verbose,1678917980012.912109,exec,cpu,reorder,simple:any,undef,src_f32:p:blocked:Acb48a:f0 dst_f32::blocked:Acb64a:f0,attr-scratchpad:user ,,1024x1024x1,3.31201
```

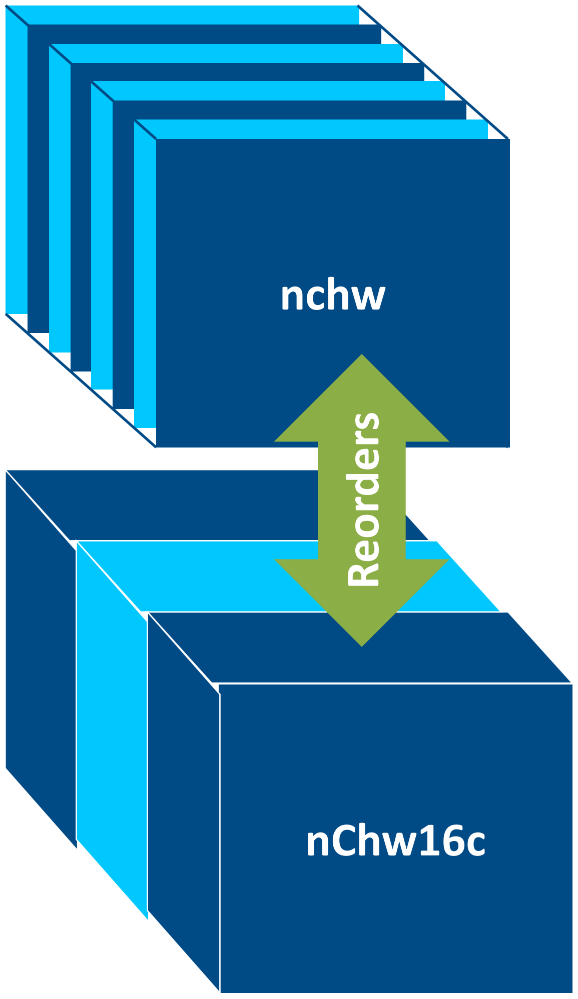
- Note the ISA. For AMX, you should see the following:
  - Intel AMX with bfloat16 and 8-bit integer support
- Check for AMX in the primitive implementation:

```
onednn_verbose,1673049613345.454102,exec,cpu,convolution,brgconv:avx512_core_amx_bf16,forward_training,src_bf16::blocked:acdb:f0 wei_
onednn_verbose,1673049613348.691895,exec,cpu,convolution,brgconv_1x1:avx512_core_amx_bf16,forward_training,src_bf16::blocked:acdb:f0
onednn_verbose,1673049613353.259033,exec,cpu,convolution,brgconv_1x1:avx512_core_amx_bf16,forward_training,src_bf16::blocked:acdb:f0
onednn_verbose,1673049613364.104980,exec,cpu,convolution,brgconv_1x1:avx512_core_amx_bf16,forward_training,src_bf16::blocked:acdb:f0
```

Under the hood of oneDNN & IPEX

Memory Layouts Optimization

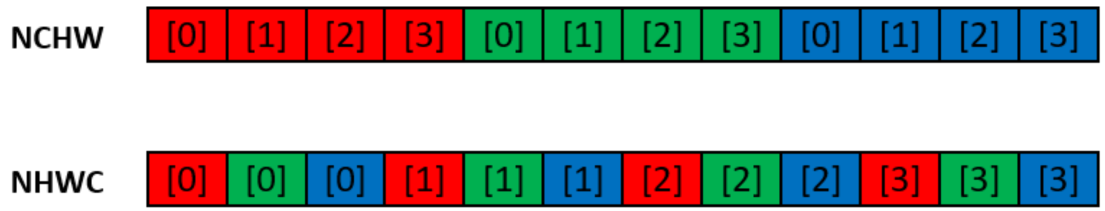
- Most popular memory layouts for image recognition are **NHWC** and **NCHW**
  - Challenging for Intel processors both for vectorization or for memory accesses
- Intel oneDNN convolutions use blocked layouts
  - Most popular oneDNN data format is nChw16c on AVX512+ systems and nChw8c on SSE4.1+ systems



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

Data Layouts in PyTorch

PyTorch

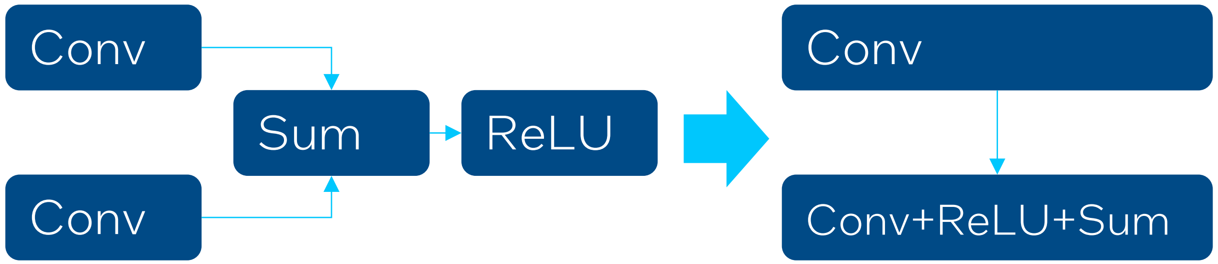


- Used in Vision workloads
- NCHW
  - Default format
  - *torch.contiguous_format*
- NHWC
  - *torch.channels_last*
  - NHWC format yields higher performance with IPEX

Channels last conversion is now applied **automatically** with IPEX
Users do not have to explicitly convert input and weight for CV models.

Fusing Computations

- On Intel processors a high percentage of time is typically spent in bandwidth-limited ops such activation functions
  - ~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 number of memory accesses
  - We fuse patterns: Conv+ReLU+Sum, BatchNorm+ReLU, etc...

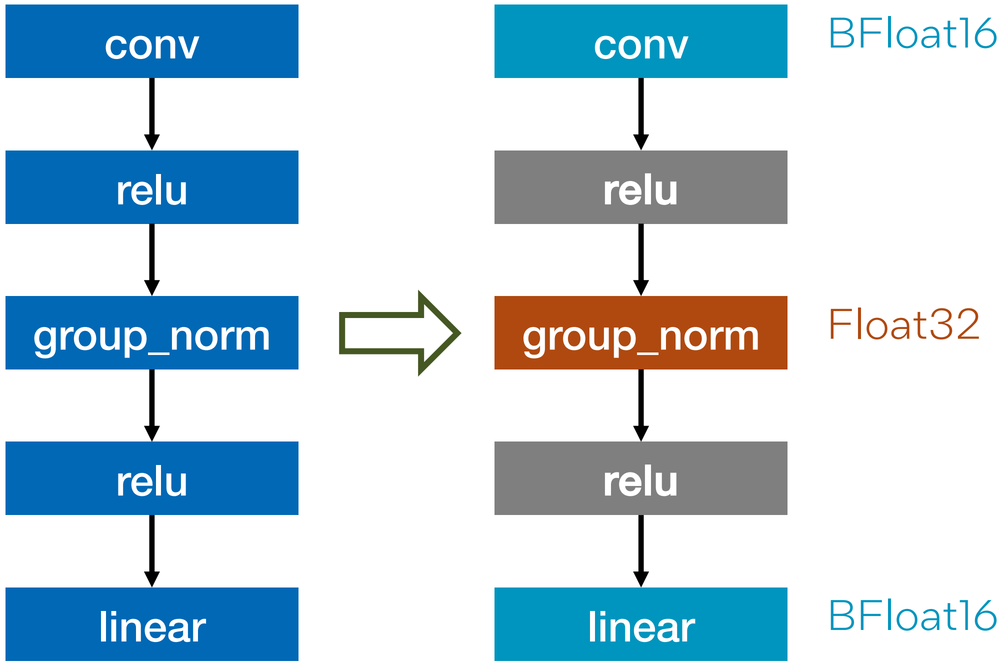


Hugging Face reports that ~70% of most popular NLP tasks in question-answering, text-classification, and token-classification can get performance benefits with such fusion patterns

Ref. https://huggingface.co/docs/transformers/perf_infer_cpu

Auto Mixed Precision (AMP)

- 3 Categories of operators
  - lower_precision_fp
    - Computation bound operators that could get performance boost with BFloat16.
    - E.g.: conv, linear
  - Fallthrough
    - Operators that runs with both Float32 and BFloat16 but might not get performance boost with BFloat16.
    - E.g.: relu, max_pool2d
  - FP32
    - Operators that are not enabled with BFloat16 support yet. Inputs of them are casted into float32 before execution.
    - E.g.: max_pool3d, group_norm



Use cases

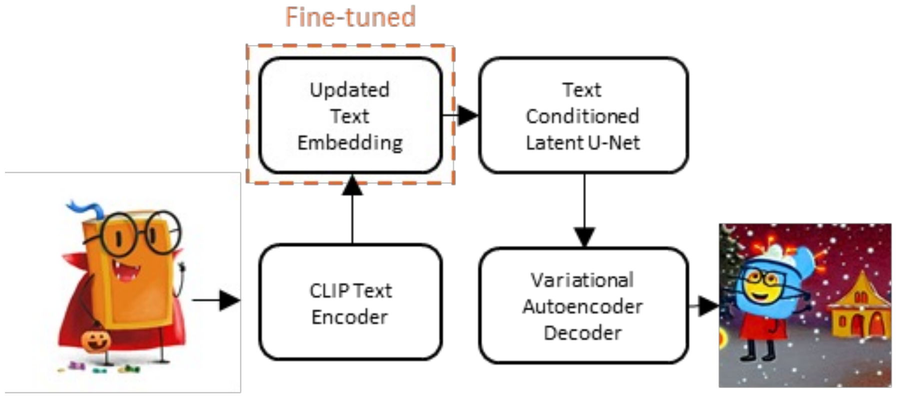
Stable Diffusion(SD) Use case

Create Your Own Stable Diffusion

Few-shot Fine-tuning

5 minutes

4 SPR nodes



object: dicoo

Accelerated Stable Diffusion Inference

Low-precision Inference

5 seconds

1 SPR chip

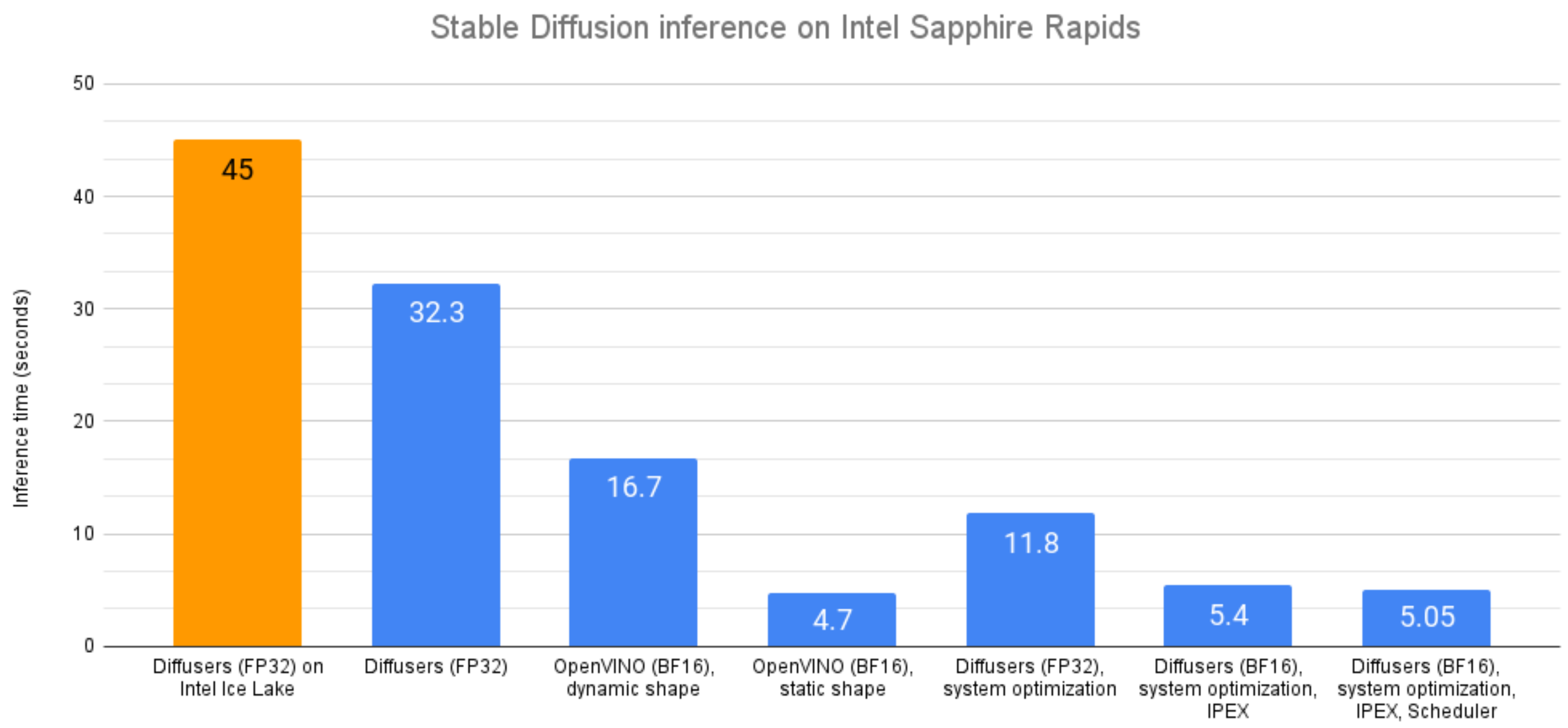
Optimizations upstreamed to Hugging Face Diffusers and Optimum-Intel

[Try SD demo here: https://huggingface.co/spaces/Intel/Stable-Diffusion-Side-by-Side](https://huggingface.co/spaces/Intel/Stable-Diffusion-Side-by-Side)

[Ref: https://venturebeat.com/ai/unlocking-generative-ai-with-ubiquitous-hardware-and-open-software/](https://venturebeat.com/ai/unlocking-generative-ai-with-ubiquitous-hardware-and-open-software/)

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!



https://huggingface.co/blog/stable-diffusion-inference-intel

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

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

Conclusion

- Intel provides a plethora of AI software tools
- 100% Python via, e.g., Docker, Conda, or Pip
- No to very minimal code changes necessary
- "Low-hanging fruit" to run AI workloads efficiently on Intel hardware

Conclusion

Data Preprocessing

MODIN

38X faster ETL compared to Pandas

Model Training

PyTorch

1.55X faster DLRM training (FP32 vs BF16)

learn

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)

learn

21X faster prediction

dmlc XGBoost

4.5X faster prediction

Thank you for your attention!

Vladimir Kilyazov

AI Software Solutions Engineer

vladimir.kilyazov@intel.com

In: @vladimirwest

Questions?

Back-Up

How to get Modin

- In the Intel AI Analytics toolkit

- Through conda wheel:

```
conda install -c intel modin-all
```

How to get Intel extension for scikit-learn

- In the Intel AI Analytics toolkit

- Through pip/conda wheel:

➢ `pip install scikit-learn-intelex`

➢ `conda install scikit-learn-intelex -c conda-forge`

Gradient Boosting Acceleration – gain sources

Pseudocode for XGBoost* (0.81) implementation

```
def ComputeHist(node):
  hist = []
  for i in samples:
    for f in features:
      bin = bin_matrix[i][f]
      hist[bin].g += g[i]
      hist[bin].h += h[i]
  return hist

def BuildLvl:
  for node in nodes:
    ComputeHist(node)

  for node in nodes:
    for f in features:
      FindBestSplit(node, f)

  for node in nodes:
    SamplePartition(node)
```

Memory prefetching to mitigate

irregular memory access

Usage uint8 instead of uint32

SIMD instructions instead of scalar code

Nested parallelism

Advanced parallelism, reducing seq loops

Usage of AVX-512, vcompress instruction (from Skylake)

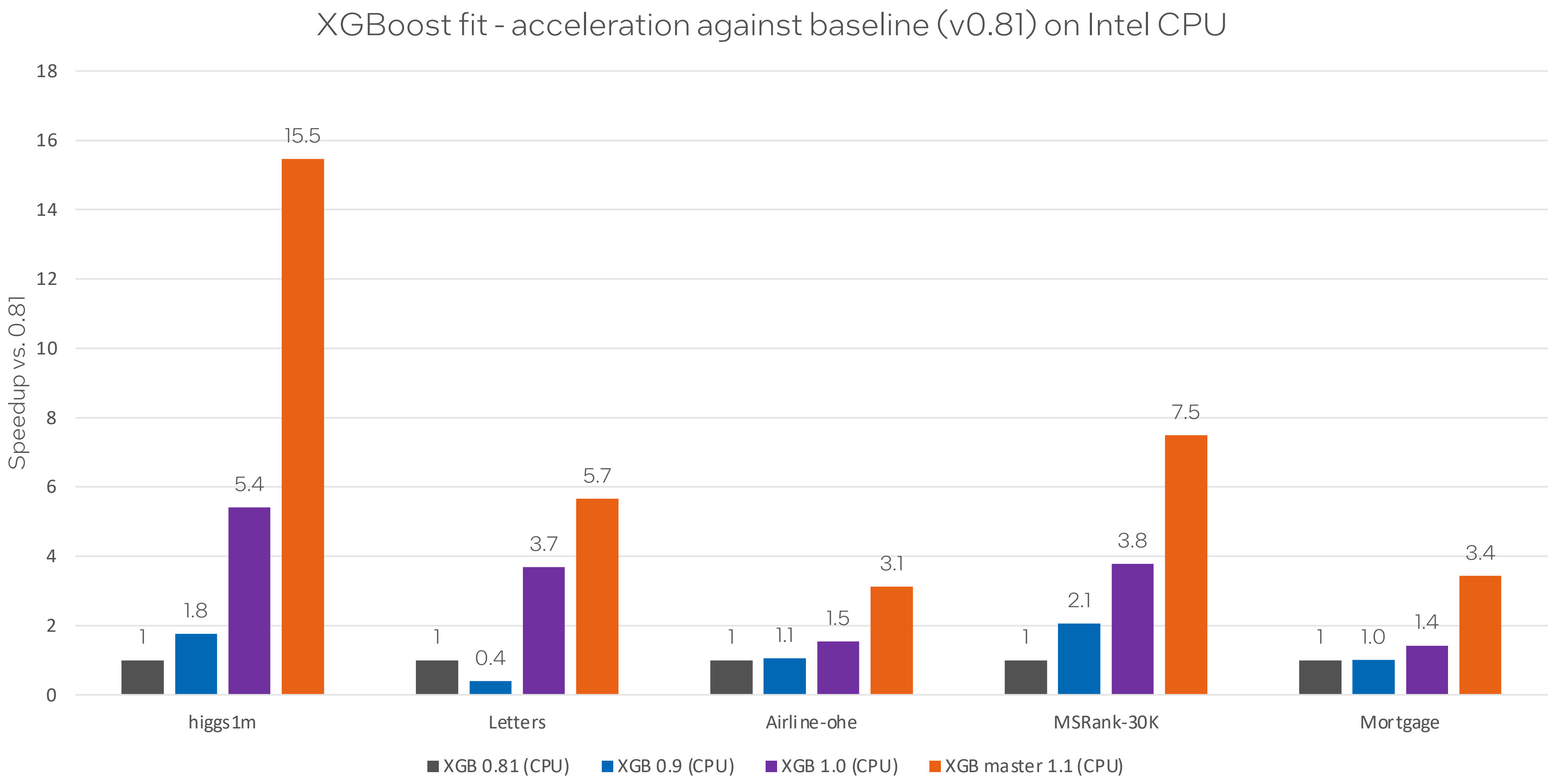
Pseudocode for Intel® oneDAL implementation

```
def ComputeHist(node):
  hist = []
  for i in samples:
    prefetch(bin_matrix[i + 10])
    for f in features:
      bin = bin_matrix[i][f]
      bin_value = load(hist[2*bin])
      bin_value = add(bin_value, gh[i])
      store(hist[2*bin], bin_value)
  return hist


def BuildLvl:
  parallel_for node in nodes:
    ComputeHist(node)

  parallel_for node in nodes:
    for f in features:
      FindBestSplit(node, f)

  parallel_for node in nodes:
    SamplePartition(node)
```

Training stage

Legend: Moved from Intel® oneDAL to XGBoost (v1.3) | Already available in Intel® DAAL, potential optimizations for XGBoost*

XGBoost* fit CPU acceleration (“hist” method)



+ Reducing memory consumption

memory, Kb	Airline	Higgs1m
Before	28311860	1907812
#5334	16218404	1155156
reduced:	1.75	1.65

CPU configuration: c5.24xlarge AWS Instance, CLX 8275 @ 3.0GHz, 2 sockets, 24 cores per socket, HT:on, DRAM (12 slots / 32GB / 2933 MHz)

Intel Confidential

How to get the optimized frameworks

- In the Intel AI Analytics toolkit

No need to call the flag for Tensorflow

- Through the framework pip/conda wheel:

PyTorch Build	Stable (1.11.0)	Preview (Nightly)	LTS (1.8.2)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python	C++ / Java		
Compute Platform	CUDA 10.2	CUDA 11.3	~~ROCm 4.5.2 (beta)~~	CPU
Run this Command:	`pip3 install torch torchvision torchaudio`			

TensorFlow > Install

Was this helpful?

Install TensorFlow with pip

TensorFlow 2 packages are available

- `tensorflow` —Latest stable release with CPU and GPU support *(Ubuntu and Windows)*
- `tf-nightly` —Preview build *(unstable)*. Ubuntu and Windows include GPU support.

How to get the Intel Neural Compressor

- In the Intel AI Analytics toolkit

- Through the pip/conda wheel

➢conda install neural-compressor -c conda-forge -c intel

➢pip install neural-compressor