第四代英特尔® 至强® 可扩展处理器人工智能调优指南

概述

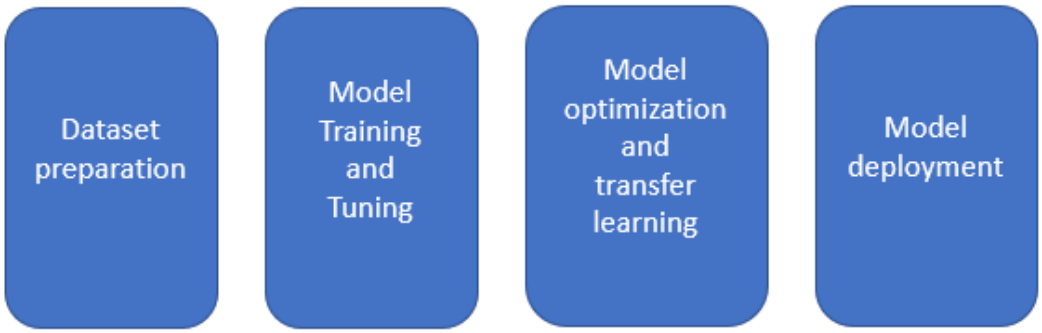
本指南面向已熟练掌握英特尔® AI 分析工具套件 (Intel® AI Analytics Toolkit) 和英特尔® 分发版 OpenVINO™ 工具包的用户，针对面向英特尔® 架构优化的 AI 工具包，提供基于第四代英特尔® 至强® 可扩展处理器平台的调优建议。所述硬件和软件配置在大多数情况下均可提供出色性能。但由于所涉及工具能够以多种方式进行部署，请用户就其特定场景对相关设置进行仔细斟酌。

第四代英特尔® 至强® 可扩展处理器平台是一个独特的可扩展平台，采用多项针对科学计算、AI、大数据、网络等工作负载的优化加速技术，具有更高性能和更优 TCO:

- 配置更多内核，每路可达 56 个内核，因此，一个 8 路平台可拥有多达 448 个内核
- 内置全新 AI 加速引擎——英特尔® 高级矩阵扩展（Intel® Advanced Matrix Extensions，英特尔® AMX），支持 BF16 和 INT8 两种数据类型，可加速包括自然语言处理 (NLP)、推荐系统、图像识别等在内的多种 AI 推理和训练工作负载
- 配备 DDR5（上一代配置 DDR4）和高带宽内存 (HBM)，为内存敏感型工作负载提供更大内存带宽和更高处理速度
- 配备全新内置存内数据库加速器 (In-Memory Database Accelerator, IAX)，提升数据分析速率
- 配备 PCIe 5.0，能够以高达 2 倍 I/O 带宽为时延敏感型工作负载提供更高吞吐量
- 内置英特尔® 动态负载均衡器（Intel® Dynamic Load Balancer，英特尔® DLB），高效处理网络数据，提升整体系统性能
- 内置英特尔® 数据保护与压缩加速技术（Intel® QuickAssist Technology，英特尔® QAT），使密码操作和数据压缩工作负载实现高达 4 倍加速

AI 应用开发阶段

典型的深度学习应用开发和部署包括以下几个阶段:

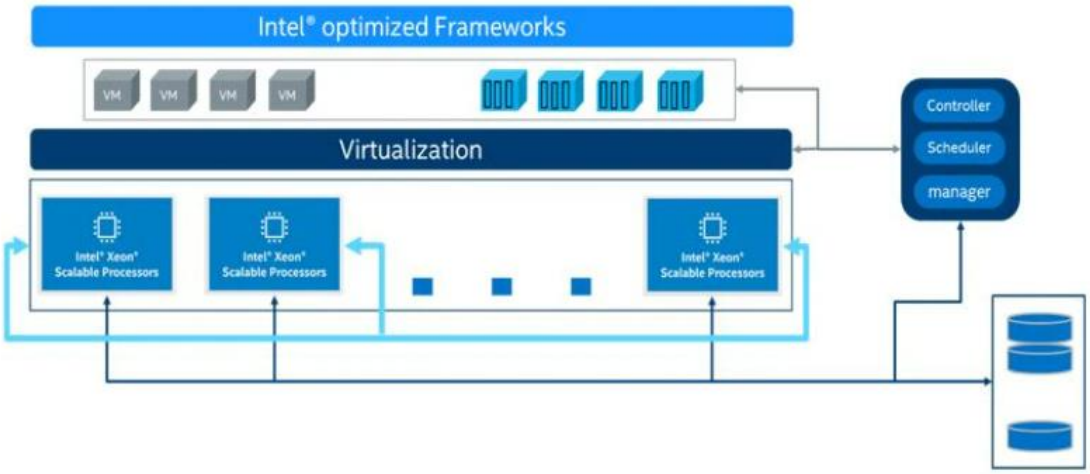


各阶段需要配置以下资源:

- 算力
- 内存
- 数据集存储设备

- 计算节点之间的通信链路
- 经优化的软件

选择合适的资源配置组合可以大幅提升 AI 业务效率。基于第四代英特尔® 至强® 可扩展处理器平台的基础设施可支持机器学习/深度学习训练和推理，因此从数据集准备、模型训练、模型优化到模型部署等各个阶段都可以在该基础设施上完成。推荐的基础设施示意图如下:



英特尔® AMX 介绍

英特尔® 高级矢量扩展 512 (Intel® Advanced Vector Extensions 512，英特尔® AVX-512) 是 x86 处理器上的一套单指令多数据 (SIMD) 指令集，用于实现通过一条指令执行多个数据操作。英特尔® AVX-512 顾名思义，使用位宽是 512 位的寄存器，可以支持 16 个 32 位单精度浮点数或 64 个 8 位整数。

英特尔® 至强® 可扩展处理器支持多种工作负载，包括复杂的 AI 工作负载。该处理器还借助英特尔® 深度学习加速技术（Intel® Deep Learning Boost，英特尔® DL Boost）进一步提升 AI 计算性能。英特尔® DL Boost 包含英特尔® AVX-512 VNNI（矢量神经网络指令）、英特尔® AVX512 BF16 和英特尔® AMX。

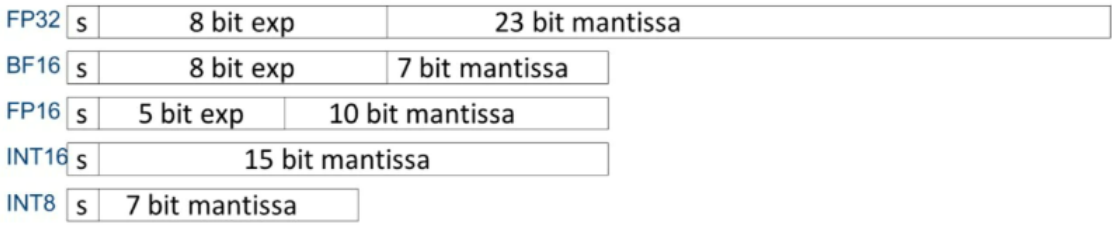
英特尔® AVX-512 VNNI 可以将三条指令（vpmaddubsw、vpmaddwd 和 vpaddd）合并成一条指令 (vpdpbusd) 执行，这进一步增强了新一代英特尔® 至强® 可扩展处理器的计算潜能，提升了 INT8 模型的推理性能。目前第二代、第三代和第四代英特尔® 至强® 可扩展处理器全部支持英特尔® VNNI。

英特尔® AVX-512 BF16 包含的 VDPBF16PS 指令可以进行 BF16 对点积运算并将结果累加成单精度 (FP32)，VCVTNE2PS2BF16 和 VCVTNEPS2BF16 指令可以将打包的单精度数据 (FP32) 转化为打包的 BF16 数据。

英特尔® AMX 是全新 64 位编程范式，由两部分构成：代表大型 2D 内存映像子阵列的一组 2D 寄存器 (TILE) 和在 TILE 执行操作的加速器，前者也称 TMUL（平铺矩阵乘法）单元。

英特尔® DL Boost

处理器	AVX-512 VNNI	AVX-512 BF16	AMX
Cascade Lake	V		
Cooper Lake	V	V	
Ice Lake	V		
Sapphire Rapids	V	V	V



深度学习 VNNI

未使用 VNNI 的平台需要运行三条指令（vpmaddubsw、vpmaddwd 和 vpaddd）才能完成 INT8 卷积运算中的乘累加：

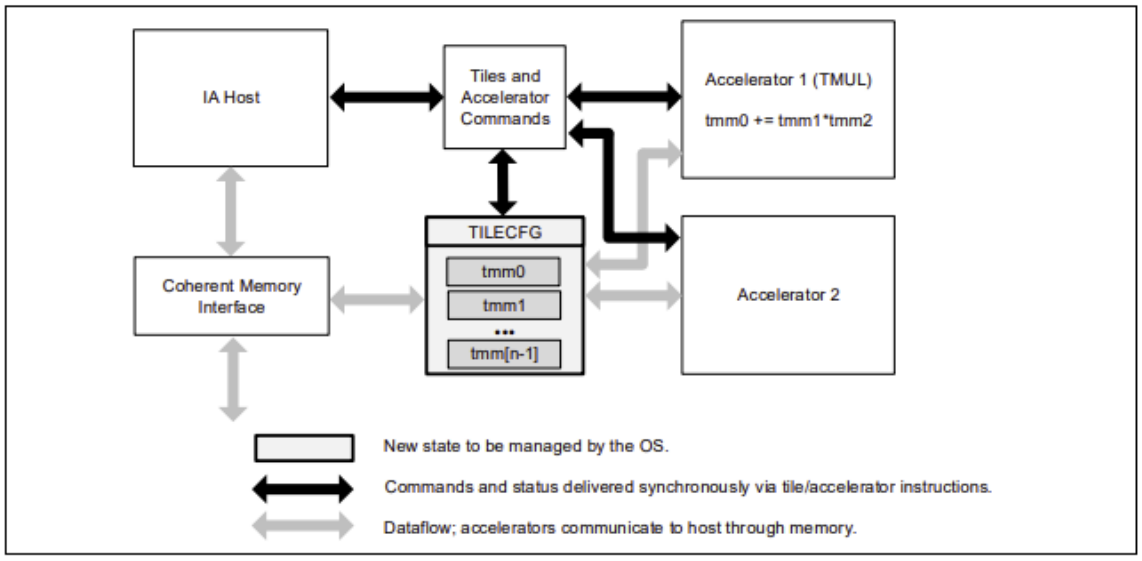
上文所述 INT8 卷积运算示意图

采用 VNNI 的平台仅需 vpdpbusd 一条指令即可完成 INT8 卷积运算：

上文所述 INT8 卷积运算示意图

英特尔® AMX

英特尔® AMX 引入一个包含 8 个名为“TILE”的二阶张量寄存器的全新矩阵寄存器文件，以及可在“TILE”上执行运算的加速器。矩阵寄存器文件包含 8 个“TILE”（按“TMM0”至“TMM7”命名），每个“TILE”最大为 16 行乘 64 字节列，总大小为 1 KiB/寄存器，整个寄存器文件总大小为 8 KiB。加载代表内存中较大映像一小部分的“TILE”，并在该“TILE”上执行运算，然后对代表该映像下一部分的下一个“TILE”重复相同操作。完成后，将生成的“TILE”存储至内存。平铺矩阵乘法（TMUL）单元为一组可在“TILE”上执行运算的融合乘加单元。



英特尔® AMX 子扩展及相关指令

英特尔® AMX 包含 3 个子扩展：AMX-TILE、AMX-INT8 和 AMX-BF16。

指令	AMX-TILE	AMX-INT8	AMX-BF16
LDTILECFG	V		

指令	AMX-TILE	AMX-INT8	AMX-BF16
STTILECFG	V		
TILELOADD	V		
TILELOADDT1	V		
TILESTORED	V		
TILERELEASE	V		
TILEZERO	V		
TDPBSSD		V	
TDPBSUD		V	
TDPBUSD		V	
TDPBUUD		V	
TDPBF16PS			V

环境

本配置的基础硬件为第四代英特尔® 至强® 可扩展处理器，服务器平台、内存、硬盘和网卡可按使用要求选用。针对本调优指南测试的硬件和软件包括：

硬件

硬件	型号
服务器平台（名称/品牌/型号）	英特尔® Eagle Stream 服务器平台
CPU	英特尔® 至强® 铂金 8480 CPU @ 2.00GHz
内存	8 x 32 GB DDR5，4800 MT/s

软件

软件	版本
操作系统	Ubuntu 22.04.1 LTS
内核	5.17.6
频率驱动程序	intel_pstate
频率调节器	性能

BIOS 设置

设置项目	建议
插槽配置 → 处理器配置	
（新 BIOS））启用 LP	单线程
（旧 BIOS））超线程 [全部]	禁用
插槽配置 → 高级电源管理配置	
CPU（P 状态）控制	将以下各项设置为“禁用”：

设置项目	建议
	SpeedStep（P 状态）、 高能效睿频、 CPU Flex Ratio Override、 Perf P-Limit
CPU C 状态（C-State）控制	将以下各项设置为“禁用”： CPU C1 自动降级、 禁止 CPU C1 自动降级、 CPU C6 报告、增强型空闲电源管理状态转换（C1E）
封装 C 状态（C-State）控制 → 封装 C 状态（C-State）	C0/C1 状态
插槽配置 → 高级电源管理配置 → 高级电源管理调优 → 能耗/性能偏差	
功耗性能调优	OS 控制 EPB

内存

使用所有可用的内存通道

CPU

第四代英特尔® 至强® 可扩展处理器采用全新英特尔® AMX 指令集。英特尔® AMX 可加速混合精度深度学习训练和推理工作负载。第四代英特尔® 至强® 可扩展处理器提供 AMX_BF16 和 AMX_INT8 两套指令集，可分别加速 bfloat16 和 int8 操作。

注：为确认 CPU 是否支持 AMX_BF16 和 AMX_INT8 指令集，请在 bash 终端输入以下命令，并在“flags”部分查找 AMX。如 AMX 指令集并不在列，请考虑将 Linux 内核升级至 5.17 或更高版本

```
$ cat /proc/cpuinfo
```

网络

如果需要搭建跨节点的训练集群，选择 25G/100G 网络等高速网络能够实现更好的可扩展性。

硬盘

为提高 I/O 效率，建议使用读写速度更快的固态盘及驱动器。

Linux 操作系统优化

调整面向并行编程的 Linux 操作系统，进一步提升处理速度。

OpenMP 参数设置

OpenMP 是一种并行编程规范。如您的应用实施基于 OpenMP 的线程，请使用下列环境变量进行测试并确定理想参数：

- OMP_NUM_THREADS =“容器中的 CPU 内核数”
- KMP_BLOCKTIME = 1 或者 0（根据模型的实际类型进行设置）
- KMP_AFFINITY=granularity=fine, verbose, compact,1,0

CPU 内核数

所用的 CPU 内核数对推理性能的影响如下：

- 批大小较小时（例如在线类服务），推理吞吐量的增幅会随着 CPU 内核数的增加而逐渐降低，实践中根据使用模型的不同，推荐 8-16 核 CPU 进行服务部署。
- 批大小较大时（例如离线类服务），推理吞吐量会随着 CPU 内核数的增加呈线性增长，实践中推荐使用 20 核以上的 CPU 进行服务部署。

```
# taskset -C xxx-xxx –p pid (limits the number of CPU cores used in service)
```

NUMA 配置

对于基于 NUMA 架构的服务器而言，在同一节点上配置 NUMA 较在不同节点上配置 NUMA 会有 5% - 10% 的性能提升。

```
#numactl -N NUMA_NODE -l command args ...(controls NUMA nodes running in service)
```

Linux 性能调节器配置

效率是关键考量因素。将 CPU 频率设置为峰值以确保最佳性能。

```
# cpupower frequency-set -g performance
```

CPUC 状态（C-States）设置

每个 CPU 都有几种功耗模式，统称为 C 状态或者 C 模式。为在 CPU 空闲时降低功耗，可命令其进入低功耗模式。禁用 C 状态可以提升性能。

```
#cpupower idle-set -d 2,3
```

利用面向英特尔® 架构优化的 TensorFlow*

自 TensorFlow 2.9 版开始，默认启用英特尔® oneAPI 深度神经网络库（Intel® oneAPI Deep Neural Network Library，英特尔® oneDNN）以提升性能。而面向英特尔® 架构优化的 TensorFlow* 添加了多项优化，可有效提升 TensorFlow 在英特尔® 硬件上运行的性能。这样可为运行在英特尔® 硬件上的 TensorFlow 提供诸如英特尔® AVX-512 VNNI 和英特尔® AMX 等新功能特性和优化技术。

面向英特尔® 架构优化的 TensorFlow* 已作为开源项目发布于 https://github.com/Intel-tensorflow/tensorflow

安装面向英特尔® 架构优化的 TensorFlow*

> **注：** 针对第四代英特尔® 至强® 可扩展处理器的优化从面向英特尔® 架构优化的 TensorFlow* 2.11 开始，并且需使用开发工具包。

```
conda create -n intel-tf python=3.8 -y

conda activate intel-tf

pip install intel-tensorflow==2.11.dev202242
```


```
conda create -n intel-tf python=3.8 -y
conda activate intel-tf
pip install intel-tensorflow==2.11.dev202242
```


基于 TensorFlow 的深度学习模型优化建议

为在英特尔® 平台上显著提升深度学习性能，面向英特尔® 架构优化的 Tensorflow* 利用了以下各项技术：

- NUMA 控制：numactl 规定了 NUMA 调度和内存布局策略
- 多线程：利用 OpenMP 在 CPU 内核间并行处理深度学习模型
- 线程亲和性：在多处理器计算机中使用 KMP_AFFINITY 将特定线程的执行限制在物理处理单元子集中

详情请见 Maximize TensorFlow* Performance on CPU（优化 CPU 上的 TensorFlow* 性能）。

启用 BF16

第四代英特尔® 至强® 可扩展处理器基于英特尔® DL Boost 和英特尔® AMX，利用低精度数据类型 BF16 和 INT8，实现 AI 推理加速。AMX_BF16、AMX_INT8、AVX512_BF16 和 AVX512_VNNI 等多种指令都可用于加速 AI 模型。

针对推理

预训练 FP32 模型（以下以来自 TensorFlow Hub 的 resnet50 为例）：

1. 除面向英特尔® 架构优化的 TensorFlow* 外，本示例还需安装 tensorflow_hub

```
pip install tensorflow_hub
```

2. 使用 export ONEDNN_VERBOSE=1 运行推理，可看到 AVX512_BF16 和 AMX_BF16 指令均已启用。
    - 将下列代码保存为 inference.py

```
import os
import tensorflow as tf
import tensorflow_hub as tf_hub

# Enable Auto Mixed Precision
tf.config.optimizer.set_experimental_options({"auto_mixed_precision_onednn_bfloat16": True})

os.environ["TFHUB_CACHE_DIR"] = 'tfhub_models'
model = tf_hub.KerasLayer('https://tfhub.dev/google/imagenet/resnet_v1_50/classification/5')
model(tf.random.uniform((1, 224, 224, 3)))
```

    - 在终端运行推理脚本

```
export ONEDNN_VERBOSE=1
python inference.py
```

    - 查看结果

```
dst_bf16::blocked:Adcb16a:f0,,,64x3x7x7,69.343

onednn_verbose,exec,cpu,convolution,jit:avx512_core_amx_bf16,forward_training,src_bf16::blocked:acdb:f0 wei_bf16::blocked:Adcb16a:f0 bia_undef::undef::f0 dst_bf16::blocked:acdb:f0,attr-scratchpad:user attr-post-ops:binary_sub:f32:2+binary_mul:f32:2+binary_mul:f32:2+binary_add:f32:2+eltwise_relu ,alg:convolution_direct,mb1_ic3oc64_ih224oh112kh7sh2dh0ph3_iw224ow112kw7sw2dw0pw3,3.29614

onednn_verbose,exec,cpu,pooling_v2,jit:avx512_core_bf16,forward_training,src_bf16::blocked:acdb:f0 dst_bf16::blocked:acdb:f0 ws_u8::blocked:acdb:f0,,alg:pooling_max,mb1ic64_ih112oh56kh3sh2dh0ph0_iw112ow56kw3sw2dw0pw0,5.21802
```

针对训练

您还可以用 TensorFlow Keras API 训练基于 BF16 的混合精度模型

- 将下列代码保存为 training.py

```
import tensorflow as tf

from keras.utils import np_utils

from tensorflow.keras import mixed_precision


# Enable Auto Mixed Precision

mixed_precision.set_global_policy('mixed_bfloat16')


# load data

cifar10 = tf.keras.datasets.cifar10

(x_train, y_train), (x_test, y_test) = cifar10.load_data()

num_classes = 10


# pre-process

x_train, x_test = x_train/255.0, x_test/255.0

y_train = np_utils.to_categorical(y_train, num_classes)

y_test = np_utils.to_categorical(y_test, num_classes)
```

```
# build model
feature_extractor_layer = tf.keras.applications.ResNet50(include_top=False, weights='imagenet')
feature_extractor_layer.trainable = False
model = tf.keras.Sequential([
  tf.keras.layers.Input(shape=(32, 32, 3)),
  feature_extractor_layer,
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(1024, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(num_classes, activation='softmax')
])
model.compile(
 optimizer=tf.keras.optimizers.Adam(),
 loss=tf.keras.losses.CategoricalCrossentropy(),
 metrics=['acc'])

# train model
model.fit(x_train, y_train,
  batch_size = 128,
  validation_data=(x_test, y_test),
  epochs=1)

model.save('resnet_bf16_model')
```

- 在终端运行训练脚本

```
export ONEDNN_VERBOSE=1
python training.py
```

- 查看结果

dst_bf16::blocked:Adcb16a:f0,,,64x3x7x7,160.375

onednn_verbose,exec,cpu,convolution,jit:avx512_core_amx_bf16,forward_training,src_bf16::blocked:acdb:f0 wei_bf16::blocked:Adcb16a:f0 bia_bf16::blocked:a:f0 dst_bf16::blocked:acdb:f0,attr-scratchpad:user ,alg:convolution_direct,mb128_ic3oc64_ih32oh16kh7sh2dh0ph3_iw32ow16kw7sw2dw0pw3,2.0481

onednn_verbose,exec,cpu,batch_normalization,bnorm_jit:avx512_core_bf16,forward_inference,data_bf16::blocked:acdb:f0
diff_undef::undef::f0,,flags:GSR,mb128ic64ih16iw16,0.783203

onednn_verbose,exec,cpu,pooling_v2,jit:avx512_core_bf16,forward_training,src_bf16::blocked:acdb:f0 dst_bf16::blocked:acdb:f0 ws_u8::blocked:acdb:f0,,alg:pooling_max,mb128ic64_ih18oh8kh3sh2dh0ph0_iw18ow8kw3sw2dw0pw0,0.908936

启用 INT8

面向英特尔® 架构优化的 TensorFlow* 协同英特尔® Neural Compressor 能以相同的用户体验提供可兼容的 TensorFlow INT8 量化解决方案支持。

使用英特尔® Extension for PyTorch* 实现优化与性能提升

英特尔® Extension for PyTorch* 添加了多项优化，可有效提升 PyTorch 在英特尔® 硬件上运行的性能。其中大多数优化将包含在未来发布的 stock PyTorch 版本中。该扩展可为运行在英特尔® 硬件上的 PyTorch 带来，诸如英特尔® AVX-512 VNNI 和英特尔® AMX 等新功能特性和优化技术。

英特尔® Extension for PyTorch* 已作为开源项目发布于 https://github.com/intel/intel-extension-for-pytorch

使用深度学习框架 PyTorch*

请确认已安装 PyTorch，以确保扩展可正常运行。有关安装的更多信息请参考以下网址：https://intel.github.io/intel-extension-for-pytorch/latest/tutorials/installation.html#

部署 PyTorch

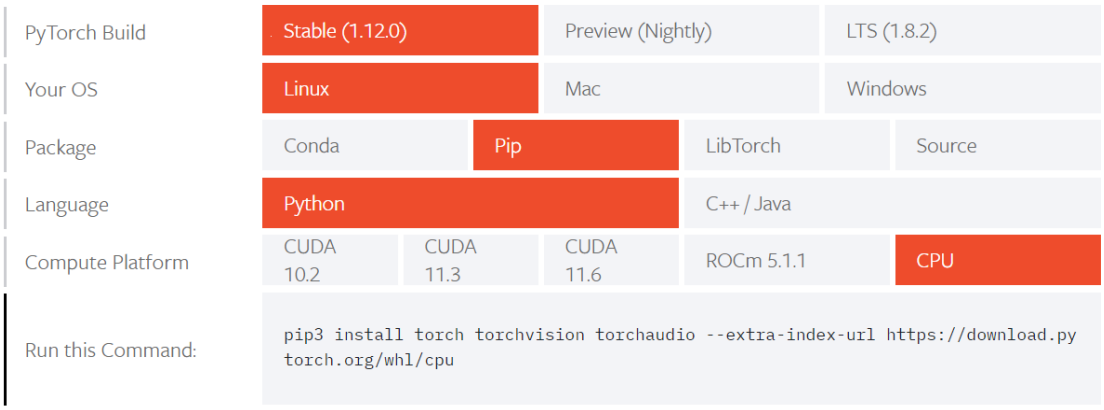
参考网址：https://intel.github.io/intel-extension-for-pytorch/latest/index.html

环境：Python 3.7 及以上

第 1 步：访问 PyTorch 官网：https://pytorch.org/

第 2 步：选择 CPU

目前，英特尔® oneDNN 已经集成至 PyTorch 的正式版本中，因此无需额外的安装步骤就可以在英特尔® 至强® 可扩展处理器平台上实现性能提升。为计算平台选择 CPU。具体见下图。



第 3 步：安装

```
pip3 install torch torchvision torchaudio --extra-index-url https://download.pytorch.org/whl/cpu
```

安装英特尔® Extension for PyTorch*

您可使用以下任意命令安装英特尔® Extension for PyTorch*：

```
python -m pip install intel_extension_for_pytorch

python -m pip install intel_extension_for_pytorch-fhttps://software.intel.com/ipex-whl-stable
```

开始

在使用英特尔® Extension for PyTorch* 开始前，用户须进行代码微调。可支持 PyTorch imperative 和 TorchScript 两种模式。

进行推理时，在模型对象中应用 ipex.optimize 函数。

进行训练时，在模型对象和优化器对象中应用 ipex.optimize 函数。

以下代码片段展示了用 BF16/FP32 数据类型进行训练的训练代码：请于 Example Page（示例页面）查看更多训练和推理示例

```
import torch

import intel_extension_for_pytorch as ipex


model = Model()
```

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

criterion = ...

optimizer = ...

model.train()


# For FP32

model, optimizer = ipex.optimize(model, optimizer=optimizer)


# For BF16

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


# Setting memory_format to torch.channels_last could improve performance with 4D input data. This is optional.

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

optimizer.zero_grad()

output = model(data)
```

针对基于 PyTorch 的深度学习模型训练和推理的优化建议

尽管 PyTorch 和英特尔® Extension for PyTorch* 的默认原语均已进行了高度优化，但依然可通过额外配置选项进一步提升性能。多数情况下，可通过能够自动设置配置选项的启动脚本（launch script）来应用，这些配置选项主要针对下列内容：

1. OpenMP 库：[英特尔® OpenMP 库（默认）| GNU OpenMP 库]
2. 内存分配器：[PyTorch 默认内存分配器 | Jemalloc | TCMalloc（默认）]
3. 实例数量：[单个实例（默认）|多个实例]

更多详情，请见 Launch Script Usage Guide（启动脚本使用指南）

除启动脚本外，还有一些其他硬件设置，包括配置英特尔® CPU 的结构和非一致性内存访问（NUMA）。也可借助软件配置，利用 Channels Last 内存格式、OpenMP 和 numactl，在英特尔® Extension for PyTorch* 的支持下充分利用 CPU 的计算资源，进而提升性能。更多详情，请见 Performance Tuning Guide（性能调优指南）

启用 BF16 推理

第四代英特尔® 至强® 可扩展处理器可基于英特尔® DL Boost 和英特尔® AMX，使用低精度数据类型 BF16 和 INT8 加速 AI 推理。AMX_BF16、AMX_INT8、AVX512_BF16 和 AVX512_VNNI 等多种指令都可用于加速 AI 模型。

自动混合精度（AMP）

Torch.cpu.ampkezai 让运行期间的数据类型自动转换更易实现。torch.float16 或 torch.bfloat16 等低精度浮点数类型可为深度学习工作负载带来诸多裨益，这是因为其计算工作负载更轻、内存使用空间更小。自动混合精度（AMP）功能自动调优各算子间的数据类型转换。

AMX_BF16 的启用步骤

请使用 lscpu 命令确认特定 CPU 设备是否支持 AMX_BF16 指令。

Torch.cpu.amp.autocast 允许各脚本域以混合精度运行。在这些域内，所有操作均能够以 autocast class 所选择的数据类型运行，进而在保证准确性的同时提高性能。下方的简单网络展示了采用混合精度可实现加速：

```
class SimpleNet(torch.nn.Module):
  def __init__(self):
    super(SimpleNet, self).__init__()
    self.conv = torch.nn.Conv2d(64, 128, (3, 3), stride=(2, 2), padding=(1, 1), bias=False)

  def forward(self, x):
    return self.conv(x)
```

Torch.cpu.amp.autocast 是一个下文管理器，可让脚本域以混合精度运行。AMX_BF16 是比 AVX512_BF16 更新的版本，具有更多高级内联函数（intrinsics），可提供更加出色的性能来支持 AI 应用。因此，对于 BFloat16 而言，AMX_BF16 拥有最高的执行优先级。经英特尔优化的 AI 框架将优先选择 AMX_BF16。如 AMX_BF16 不可用，则选择 AVX512_BF16。更多详情，请参阅 Auto Mixed Precision [自动混合精度（AMP）]

```
model = SimpleNet().eval()
x = torch.rand(64, 64, 224, 224)
with torch.cpu.amp.autocast():
  y = model(x)
```

确定是否启用 AMX_BF16，请检查

Avx512_core_amx_bf16 JIT 内核使用情况。检查设置

ONEDNN_VERBOSE=1

英特尔® Extension for PyTorch* 的 github 链接如下：Intel® Extension for PyTorch*（英特尔® Extension for PyTorch*）

AI 神经网络模型低精度优化

概述

第四代英特尔® 至强® 可扩展处理器可基于英特尔® DL Boost 和英特尔® AMX，使用低精度数据类型 BF16 和 INT8 加速 AI 推理。

1. 量化

使用 INT8 数据类型优化 AI 模型是一种量化方式，即将连续的无限值映射到更小的一组有限离散值的过程。

2. 混合精度

混合精度使用 BF16 等精度更低的数据类型，使模型在训练和推理阶段以 16 位和 32 位混合浮点数据类型运行，从而以更少的内存占用实现更快的运行速度。

指令

第四代英特尔® 至强® 可扩展处理器支持多种可加速 AI 模型的指令：

内联函数	AVX512_VNNI	AVX512_BF16	AMX_INT8	AMX_BF16
数据类型	INT8	BF16	INT8	BF16
变量类型	矢量	矢量	矩阵	矩阵

AMX_INT8/AMX_BF16 是比 AVX512_VNNI/AVX512_BF16 更新的版本，具有更多高级内联函数 (intrinsics)，可提供更加出色的性能来支持 AI 应用。AMX_INT8 拥有最高执行优先级。经英特尔优化的 AI 框架将优先选择 AMX_INT8。如 AMX_INT8 不可用，则选择 AVX512_VNNI。

步骤

1. 将 FP32 模型转换为 INT8/BF16 模型。

运行量化或混合精度流程，获得 INT8/BF16 模型。

2. 在第四代英特尔® 至强® 可扩展处理器上，借助面向英特尔® 架构优化的 AI 框架执行 INT8/BF16 模型推理。

AI 框架将调用 CPU 中自身所支持的最高级内联函数，以获得更加出色的性能。

例如：如果 AI 框架调用的是 AVX512_VNNI 而非 AMX_INT8，请检查新版本是否支持 AMX_INT8 并安装正确的版本。

AI 神经网络量化流程

神经网络的计算主要集中在卷积层和全连接层。这两层的计算均可表示为：Y = X * Weights + Bias。因此，在优化性能时重点关注矩阵乘法是十分顺其自然的事情。这种神经网络模型量化方法的出发点是以有限的精度损失换取性能提升。用低精度的整数代替 32 位浮点数做矩阵运算，不仅能够加速计算，同时也压缩了模型，从而节约了内存带宽。

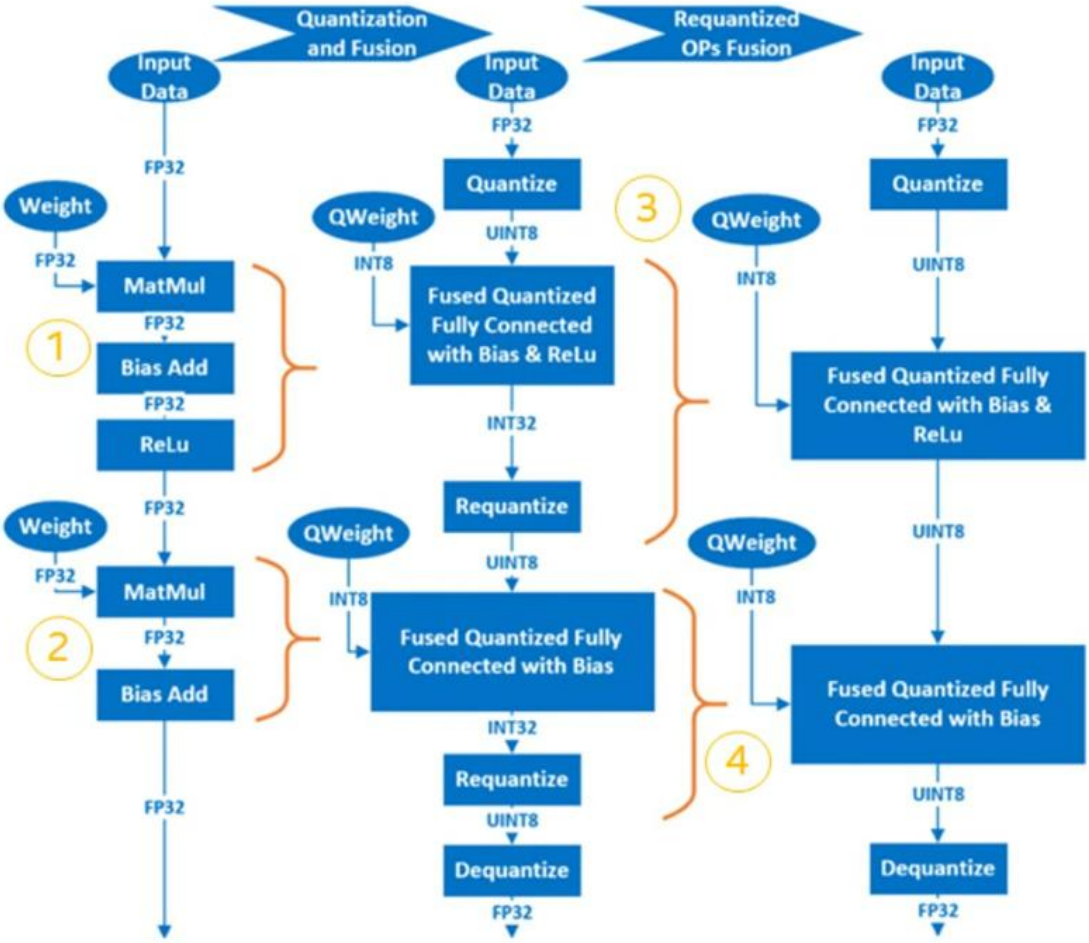
神经网络模型量化有三种方式：

- 训练后量化（Post-Training Quantization, PTQ）：大多数 AI 框架均支持。量化完成训练的 FP32 模型。执行校准流程（以小数据集进行 FP32 模型推理），并记录每层输入和输出的数据范围。根据数据范围信息量化模型。得出较好的量化结果。
- 量化感知训练（Quantization-Aware-Training, QAT）：在训练融合时将 Fake Quantization 节点嵌入 FP32 模型。增加了量化引入的噪声。在训练的反向传播阶段使模型权重落入一个有限区间，从而达到更好的量化精度。
- 动态量化（Dynamic Quantization, DQ）：这与 PTQ 非常类似，二者都是针对训练后模型使用的量化方法。不同点在于 DQ 激活层的量化因子由神经网络模型在运行时所用的数据范围动态决定；而 PTQ 则是在小规模预处理的数据集上采样，并用以获取激活层数据分布和范围信息，再将其永久记录在新生成的量化模型中。针对下文所介绍的英特尔® Neural Compressor，目前仅有 onnxruntime 在后端支持此方法。

神经网络模型训练后量化的基本流程如下：

1. 融合 FP32 OP 并转换为 INT8 OP。例如 MatMul、BiasAdd 和 ReLU 可在全连接层融合为单个量化的 OP，即 QuantizedMatMulWithBiasAndRelu。不同的神经网络框架所支持的可融合的 OP 不完全相同。针对下文所介绍的英特尔® Neural Compressor，此网站提供了 TensorFlow 所支持的可融合 OP 清单：https://github.com/intel/neural-compressor/blob/master/neural_compressor/adaptor/tensorflow.yaml#L110。有关 PyTorch 所支持的可融合 OP，请见 https://github.com/intel/neural-compressor/blob/master/neural_compressor/adaptor/pytorch_cpu.yaml#L251
2. 量化权重并保存在量化模型中。
3. 量化输入/激活层，方法是在校准数据集上采样从而获取激活层数据分布和范围信息，并将这些信息记录在新生成的量化模型中。
4. 将 Requantize 操作融合到其对应的 INT8 OP 中，以生成最终的量化模型。

以包含两层 MatMul 的简单模型为例，可看到量化过程如下：



AI 神经网络混合精度流程

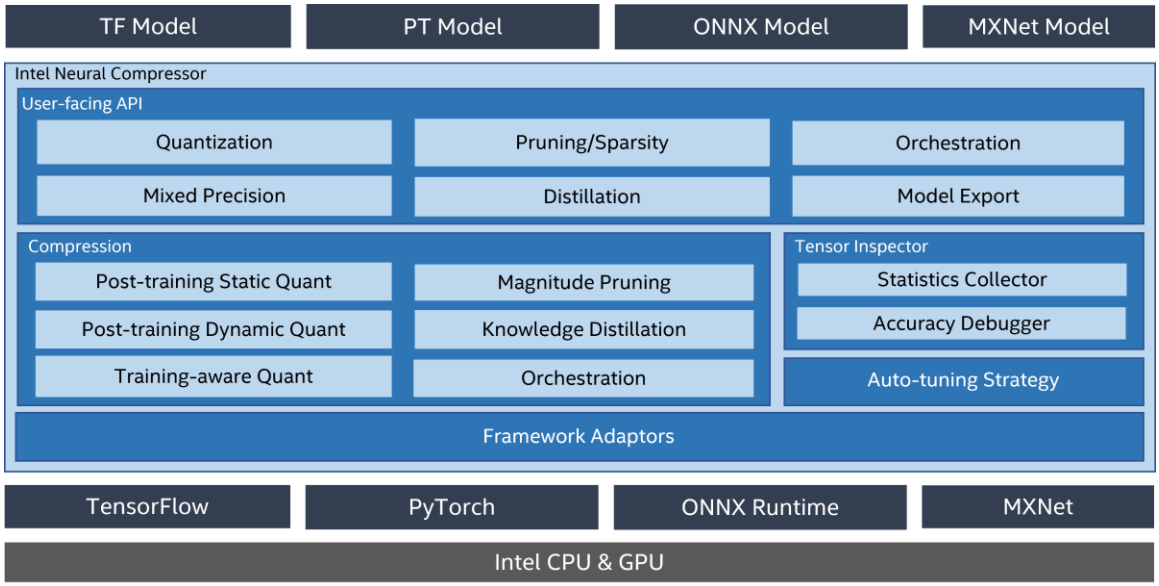
FP32 与 BF16 有相同的数据范围和不同的小数部分。将 FP32 转化为 BF16 很容易，但使用 BF16 会因截断的小数部分影响模型准确性。部分操作不会对 BF16 造成数值影响，即相比 FP32，基于 BF16 的操作不会产生明显的准确性损失。然而，部分操作则会对 BF16 造成数值影响，即相比 FP32，基于 BF16 的操作会产生明显的准确性损失。以下步骤将由 AI 框架/工具自动处理。但可对其进行配置调试，选择允许或拒绝可造成数值影响的操作。

1. 根据具体实验，设置不会造成数值影响的操作以允许或拒绝会造成数值影响的操作
2. 在“允许”设置中将操作转化为 BF16
3. 在 FP32 和 BF16 操作中插入“Cast”操作，即可实现 FP32 与 BF16 间的数据转换

英特尔® Neural Compressor

英特尔® Neural Compressor 作为英特尔® oneAPI AI 分析工具套件中的关键 AI 软件组件之一，是可在英特尔® CPU 和 GPU 上运行的开源 Python 库。针对量化、剪枝和知识蒸馏等主流网络压缩技术，该工具套件可在多个深度学习框架间构建统一的接口。它支持以准确性为准的自动调优策略，能够快速找到最佳模型。还可实施不同的权重修剪算法，生成具有预定义稀疏目标的修剪模型，并支持从教师模型到学生模型的知识蒸馏。

参考网址：https://github.com/intel/neural-compressor



目前英特尔® Neural Compressor 支持下列经英特尔优化的深度学习框架：

- Tensorflow*
- PyTorch*
- Apache* MXNet
- ONNX Runtime

当前已验证的各框架版本如下：

- OS 版本： CentOS 8.4, Ubuntu 20.04
- Python 版本： 3.7, 3.8, 3.9, 3.10

框架	TensorFlow	面向英特尔®架构优化的TensorFlow	PyTorch	IPEX	ONNX Runtime	MXNet
版本	2.9.1	2.9.1	1.12.0+cpu	1.12.0	1.11.0	1.8.0
	2.8.2	2.8.0	1.11.0+cpu	1.11.0	1.10.0	1.7.0
	2.7.3	2.7.0	1.10.0+cpu	1.10.0	1.9.0	1.6.0

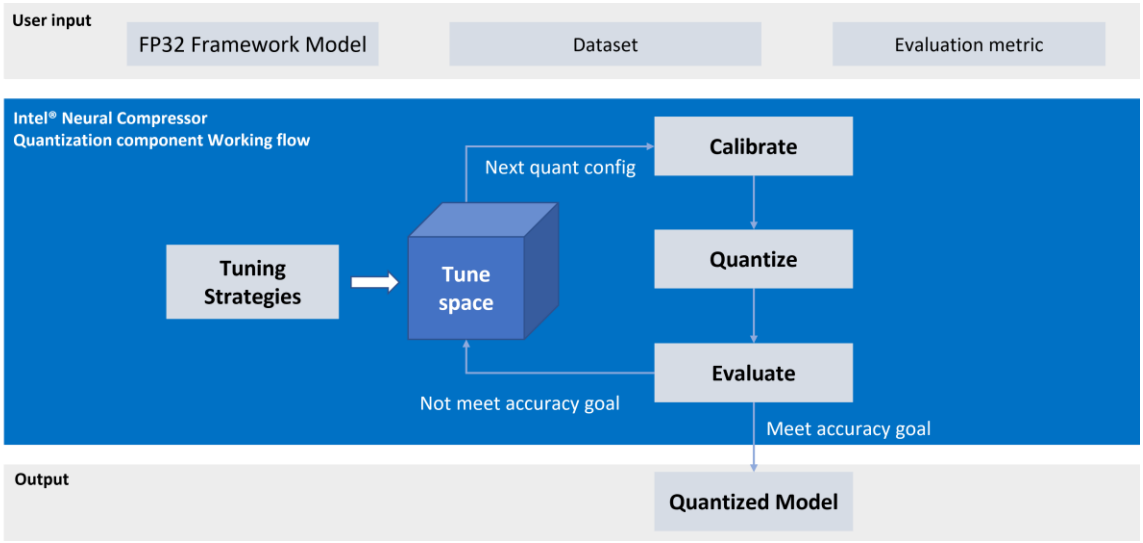
注： 如使用 TensorFlow 2.6 至 2.8 版，请设置环境变量 TF_ENABLE_ONEDNN_OPTS=1 以启用 oneDNN 优化。TensorFlow 2.9 及更高版本均已默认开启 oneDNN。

英特尔® Neural Compressor 支持的调优策略有：

- Basic
- Bayesian
- MSE
- TPE
- Exhaustive

- Random

下图所示为英特尔® Neural Compressor 工作流程。根据已设置的调优策略自动选择与准确性损失目标匹配的模型量化参数，然后生成量化模型。



安装英特尔® Neural Compressor

安装详情请参阅：https://github.com/intel/neural-compressor#installation

第 1 步：使用 Anaconda 创建名为 env_inc 的 Python3.x 虚拟环境。这里我们使用 Python 3.9 为例：

```
conda create -n env_inc python=3.9
conda activate env_inc
```

第 2 步：使用二进制文件安装英特尔® Neural Compressor：

```
# install stable basic version from pip
pip install neural-compressor
# install stable full version from pip (including GUI)
pip install neural-compressor-full
```

或

```
# install stable basic version from pip
pip install -i https://test.pypi.org/simple/ neural-compressor
# install nightly full version from pip (including GUI)
```

```
pip install -i https://test.pypi.org/simple/ neural-compressor-full
```

或

```
# install stable basic version from from conda

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

# install stable full version from from conda (including GUI)

conda install neural-compressor-full -c conda-forge -c intel
```

第 3 步：安装 AI 框架

根据将要处理的模型类型安装 Tensorflow、PyTorch、ONNX-RT、MXNet。例如安装 intel-tensorflow

```
# install intel-tensorflow from pip

pip install intel-tensorflow
```

使用英特尔® Neural Compressor

我们以 ResNet50 v1.0 为例，演示如何使用此工具进行量化和混合精度优化。

数据集准备：

第 1 步：下载 ImageNet 验证数据集并解压缩：

用户需在使用 image-net.org. 创建账户后才可下载原始图片

```
mkdir –p img_raw/val

cd img_raw

wget http://www.image-net.org/challenges/LSVRC/2012/xxxxxxxx/ILSVRC2012_img_val.tar

tar –xvf ILSVRC2012_img_val.tar -C val
```

第 2 步：将 image 文件移入按标签分类的子目录：

```
cd val

wget -qO- https://raw.githubusercontent.com/soumith/imagenetloader.torch/master/valprep.sh | bash
```

第 3 步：使用脚本 prepare_dataset.sh，将原始数据转化为 128 个 TFrecord 格式的分片：

```
git clone https://github.com/intel/neural-compressor.git

cd neural-compressor/

git checkout 6663f7b

cd examples/tensorflow/image_recognition/tensorflow_models/quantization/ptq

bash prepare_dataset.sh --output_dir=./data --raw_dir=/PATH/TO/img_raw/val/ --shards=128 --subset=validation
```

参考网址：https://github.com/intel/neural-compressor/tree/master/examples/tensorflow/image_recognition/tensorflow_models/quantization/ptq#2-prepare-dataset

预训练 FP32 模型准备：

```
wget https://storage.googleapis.com/intel-optimized-tensorflow/models/v1_6/resnet50_fp32_pretrained_model.pb
```

运行量化：

编辑文件：

examples/tensorflow/image_recognition/tensorflow_models/quantization/ptq/resnet50_v1.yaml，将量化和评估的默认数据集路径（root: /path/to/evaluation/dataset）更改为实际本地路径，用来保存之前在数据准备阶段生成的 TFrecord 格式数据集。

```
cd examples/tensorflow/image_recognition/tensorflow_models/quantization/ptq

bash run_tuning.sh --config=resnet50_v1.yaml \

  --input_model=/PATH/TO/resnet50_fp32_pretrained_model.pb \

  --output_model=./nc_resnet50_v1.pb
```

参考网址：https://github.com/intel/neural-compressor/tree/master/examples/tensorflow/image_recognition/tensorflow_models/quantization/ptq#1-resnet50-v10

运行混合精度（BF16 + FP32）：

参考 Mixed Precision（混合精度）

无准确性调优转换

该示例方法可将尽可能多的节点转换为 BF16，但可能发生的模型准确性损失不可预测。

```
python convert_bf16_without_tuning.py
```

```
from neural_compressor.experimental import MixedPrecision
converter = MixedPrecision()
converter.precisions = 'bf16'
converter.model = '/PATH/TO/resnet50_fp32_pretrained_model.pb'
optimized_model = converter()
optimized_model.save('nc_resnet50_v1_bf16.pb')
```

准确性调优转换

该方法以准确性为准，且需要评估流程。模型准确性损失可预测，但转换为 BF16 的节点比无准确性调优方法转换的要少。

1. Edit resnet50_v1.yaml

添加

```
mixed_precision:
 precisions: 'bf16'
```

示例：

```
model:
 name: resnet50_v1
 framework: tensorflow

mixed_precision:
 precisions: 'bf16'

evaluation:
 accuracy:
  dataloader:
   ...
  metric:
   ...
```

2. 运行 Python

```
python convert_bf16_with_tuning.py


from neural_compressor.experimental import MixedPrecision
converter = MixedPrecision('resnet50_v1.yaml')
converter.precisions = 'bf16'
converter.model = '/PATH/TO/resnet50_fp32_pretrained_model.pb'
optimized_model = converter()
optimized_model.save('nc_resnet50_v1_bf16.pb')
```

INT8 + BF16 + FP32

可在同一模型中混合 INT8、BF16、FP32 来优化性能。在本示例中，将混合精度应用于量化模型：nc_resnet50_v1.pb.中，由此获得 INT8 + BF16 + FP32 模型。

```
python convert_bf16_without_tuning.py


from neural_compressor.experimental import MixedPrecision
converter = MixedPrecision()
converter.precisions = 'bf16'
converter.model = '/PATH/TO/nc_resnet50_v1.pb'
optimized_model = converter()
optimized_model.save('nc_resnet50_v1_int8_bf16.pb')
```

我们可在日志中看到一项操作：MatMul 转换为 BF16。该操作可略微提升性能。如果量化模型拥有更多 FP32 操作，混合精度可实现更多向 BF16 的转换并获得更加出色的性能。

```
2022-08-04 19:50:01 [INFO] |**********Mixed Precision Statistics**********|
2022-08-04 19:50:01 [INFO] +-------------+--------+-------+-------+-------+
2022-08-04 19:50:01 [INFO] |  Op Type  | Total | INT8 | BF16 | FP32 |
2022-08-04 19:50:01 [INFO] +-------------+--------+-------+-------+-------+
2022-08-04 19:50:01 [INFO] |  MaxPool  | 1  | 1 | 0  | 0  |
2022-08-04 19:50:01 [INFO] |  AvgPool  | 1  | 1 | 0  | 0  |
```

```
2022-08-04 19:50:01 [INFO] |  Conv2D  | 53 | 53 | 0 | 0 |
2022-08-04 19:50:01 [INFO] |  MatMul  | 1 | 0 | 1 | 0 |
2022-08-04 19:50:01 [INFO] | QuantizeV2 | 1 | 1 | 0 | 0 |
2022-08-04 19:50:01 [INFO] | Dequantize | 1 | 1 | 0 | 0 |
2022-08-04 19:50:01 [INFO] |  Cast  | 2 | 0 | 1 | 1 |
2022-08-04 19:50:01 [INFO] +-------------+--------+-------+-------+-------+
```

运行基准测试：

性能测试：

```
bash run_benchmark.sh --input_model=./xxx.pb --config=resnet50_v1.yaml --mode=performance
```

性能模式基准测试结果：

```
批大小 = 1
时延：xxx
吞吐量：xxx
```

准确性测试：

```
bash run_benchmark.sh --input_model=./xxx.pb --config=resnet50_v1.yaml --mode=accuracy
```

准确性模式基准测试结果：

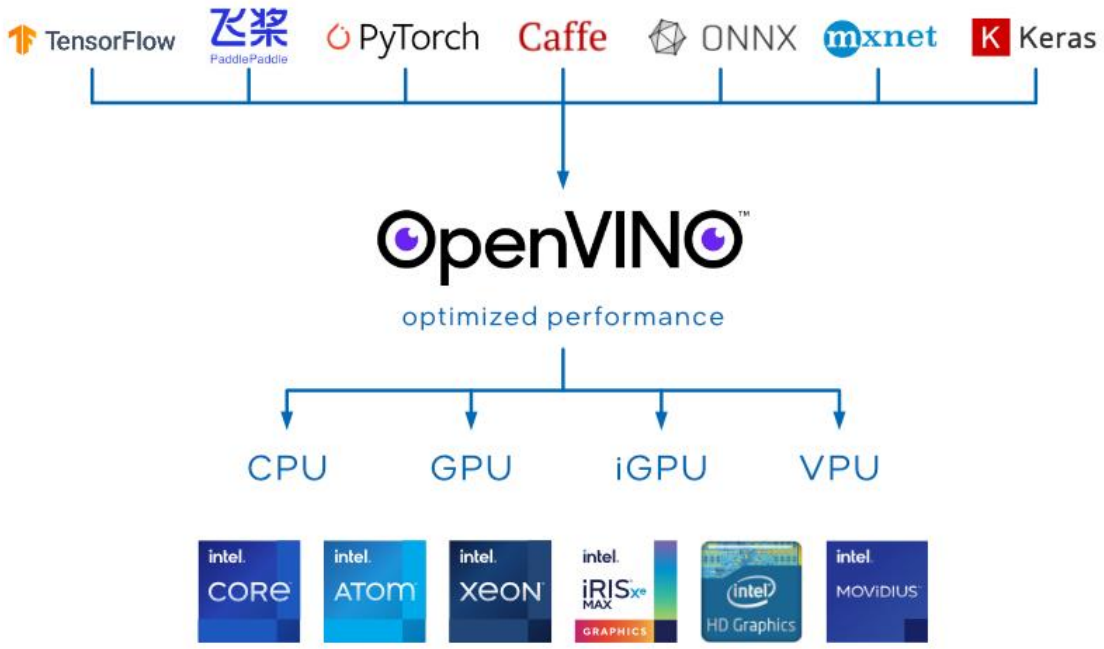
```
准确性为 x.xx
批大小 = 32
```

使用英特尔® 分发版 OpenVINO™ 工具包加速推理

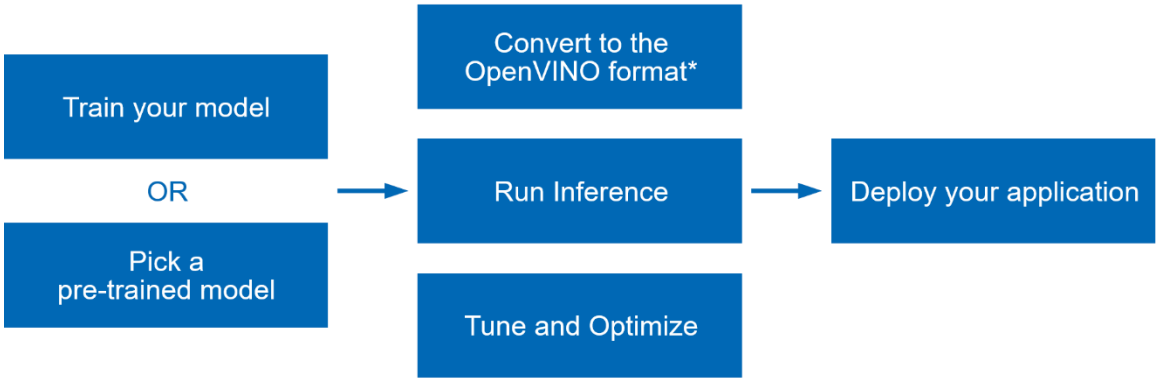
英特尔® 分发版 OpenVINO™ 工具包

OpenVINO™ 是一个用于优化和部署 AI 推理的开源工具平台。它可以：

- 提高计算机视觉、自动语音识别、自然语言处理和其他常见任务的深度学习性能
- 使用经过 TensorFlow、PyTorch 等主流框架训练的模型
- 减少资源需求，并在从边缘到云的一系列英特尔® 平台上高效部署



部署使用 OpenVINO™ 完成训练的深度学习模型时的典型工作流程如下图所示：



- 使用 TensorFlow* 和 PyTorch* 等主流框架进行模型训练，或从 Open Model Zoo 中选择预训练模型。
- 运行模型优化器执行静态模型分析，并生成该模型可用 OpenVINO™ Runtime 推理的中间表示 (Intermediate Representation, IR)。
- 使用 OpenVINO™ Runtime API 读取中间表示 (IR)、ONNX 或 PaddlePaddle（百度飞桨）模型，并在首选设备中执行。

- 采用量化、剪枝和流程优化等特殊优化方法，对整个流水线进行调优和优化，提升最终模型性能。
- 所有步骤完成后，使用 OpenVINO™ 部署您的应用。

更多详情，请参考 OpenVINO™ 在线文档：https://docs.openvino.ai/cn/latest/index.html

使用英特尔® DL Boost 启用 BF16/INT8 推理

CPU 原语的默认浮点精度为 FP32。对于可通过 AVX512_BF16 或 AMX_BF16 扩展提供原生 BF16 计算支持的平台，将自动使用 BF16 而非 FP32，进而实现更加出色的性能。有关 BF16 格式的更多详情，请参阅 BFLOAT16 – Hardware Numerics Definition white paper（BFLOAT16——硬件数值定义白皮书）。

使用 BF16 精度类型可带来以下性能增益：

- 由于 BF16 数据的尾数位数更少，两数乘积运算速度更快。
- 由于 BF16 数据的大小是 32 位浮点数数据大小的 1/2，内存占用率更低。

如需确认 CPU 设备是否支持 BF16 数据类型，请使用 openvino.runtime.Core.get_property 查询 ov::device::capabilities 属性。应如下图所示在 CPU 功能列表中包含 BF16：

```
core = Core()

cpu_optimization_capabilities = core.get_property("CPU", "OPTIMIZATION_CAPABILITIES")
```

在运行推理时，使用 benchmark_app 检查 BF16/INT8 是否启用：

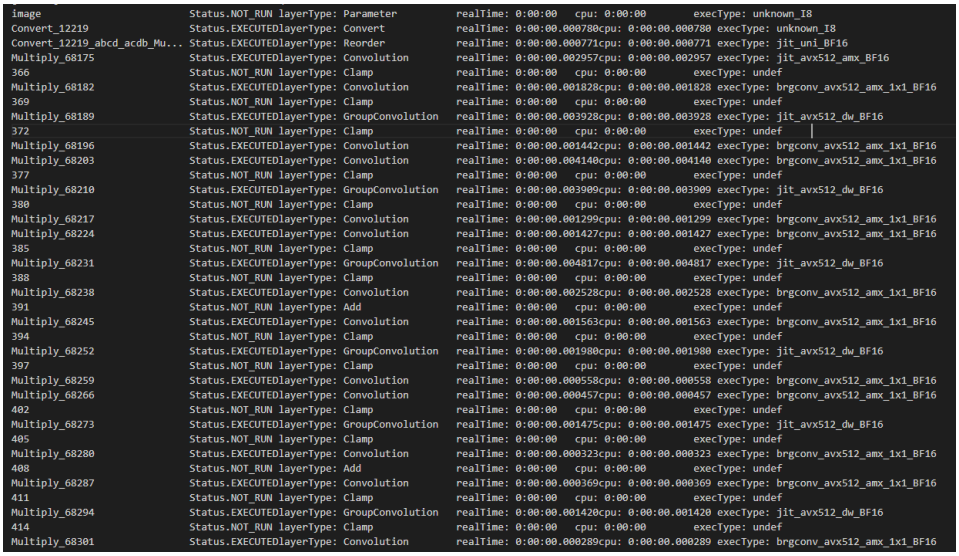
- 安装 OpenVINO™ 开发工具并安装 OpenVINO™ Runtime
- BF16
  - 在 Open Model Zoo 中下载 FP32 模型（或选择您自有的 FP32 模型），在此下载 horizontal-text-detection-0001 示例：

    ```
    omz_downloader --name horizontal-text-detection-0001 --precisions FP32 -o .
    ```

  - 使用 -pc 运行基准测试应用

    ```
    benchmark_app -m ./intel/horizontal-text-detection-0001/FP32/horizontal-text-detection-0001.xml -pc
    ```

  - 可以看到：部分内核同时使用英特尔® AVX-512 和英特尔® AMX 两种指令运行 BF16 精度数据。



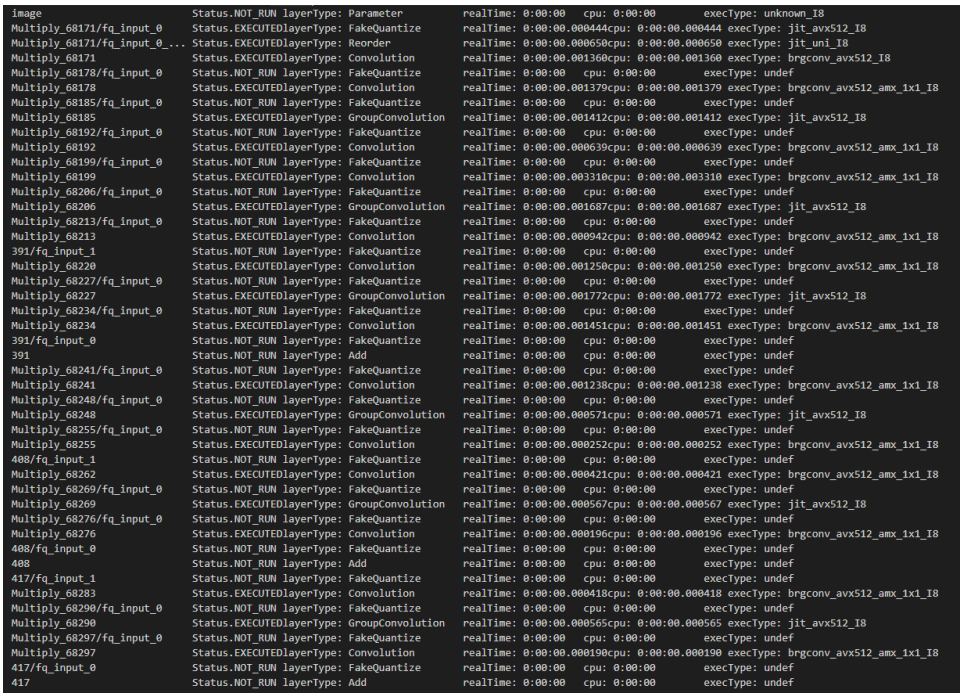
- INT8
  - 在 Open Model Zoo 中下载 INT8 模型（或选择您自有的 INT8 模型），在此下载 horizontal-text-detection-0001 示例：

```
omz_downloader --name horizontal-text-detection-0001 --precisions FP16-INT8 -o .
```

  - 使用 -pc 运行基准测试应用

```
benchmark_app -m ./intel/horizontal-text-detection-0001/FP16-INT8/horizontal-text-detection-0001.xml -pc
```

  - 可以看到：部分内核同时使用英特尔® AVX-512 和英特尔® AMX 两种指令运行 INT8 精度数据。



注:

由于 BF16 数据类型的尾数位数减少，因此 BF16 推理准确性或与 FP32 推理准确性会有差异，尤其是对于那些未利用 BF16 数据类型进行训练的模型。如无法接受 BF16 推理准确性，可转为 FP32 精度。

C++:

```
ov::Core core;
core.set_property("CPU", ov::hint::inference_precision(ov::element::f32));
```

Python:

```
core = Core()
core.set_property("CPU", {"INFERENCE_PRECISION_HINT": "f32"})
```

如使用 benchmark_app，请将 -infer_precision 设置为 f32，例如:

```
benchmark_app -m ./intel/horizontal-text-detection-0001/FP32/horizontal-text-detection-0001.xml -pc -infer_precision f32
```

数据分析和机器学习加速

作为人工智能行业的重要分支，机器学习已成为备受瞩目的焦点领域。基于机器学习的数据分析正逐渐成为主流。与其它分析方法相比，机器学习能够帮助众多企业或机构的 IT 人员、数据科学家和业务团队迅速释放和利用 AI 优势。机器学习还可提供多种全新商用开源解决方案，为开发人员提供广泛的生态系统。开发人员可以从各种各样的开源机器学习库中进行选择，如 Scikit-learn、*Cloudera* 和 Spark* MLlib。

英特尔® 分发版 Python*

英特尔® 分发版 Python* 是一个面向 AI 软件开发人员的 Python 开发工具包，可以加速 Python 在英特尔® 至强® 可扩展处理器平台上的计算速度。该工具包可通过 Anaconda* 获取，也可以与 Conda、*PIP*、APT GET、YUM、Docker* 及其他工具一起安装和使用。

英特尔® 分发版 Python* 的特点：

- 充分利用主流和快速发展的编程语言，以及面向英特尔® 架构优化的底层指令集。
- 通过加速用英特尔® 性能程序库搭建的核心 Python 数值软件包和科学软件包，实现近原生性能。
- 实现高效的多线程、矢量化和内存管理，以及集群内的科学计算扩展。
- 核心软件包涵盖 Numba、NumPy 和 SciPy 等。

参考和下载网址：https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/distribution-for-python.html

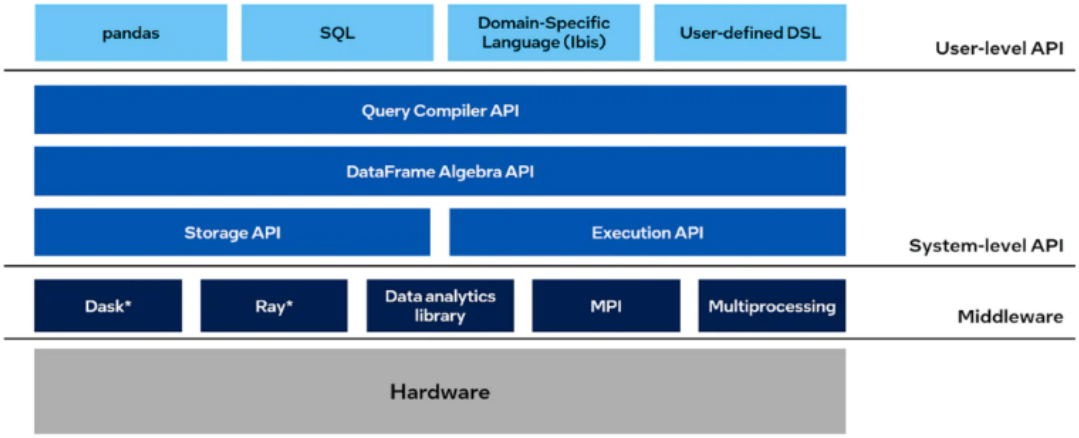
英特尔® 分发版 Modin*

Modin* 是 pandas 的嵌入式替代库，数据科学家可借此在无需更改 API 编码的前提下扩展分布式 DataFrame 处理。英特尔® 分发版 Modin* 增加了多项优化，可进一步提升 Modin 在英特尔® 硬件上的处理速度。

使用此库，您可以：

- 在单个工作站上处理 TB 级数据
- 使用相同代码从单个工作站扩展到云端

更多地关注数据分析，而非学习新的 API



英特尔® 分发版 Modin* 的特点：

- 加速 DataFrame 处理
  - 加快大型 DataFrame 的提取、转换和加载（ETL）流程
  - 自动使用您设备上可用的所有处理内核
- 面向英特尔® 硬件优化
  - 在单个数据科学工作站上使用英特尔® 傲腾™ 持久内存，实现 TB 级数据扩展
  - 使用 HEAVY.AI* 数据分析工具分析大型数据集（超过 10 亿行）
- 兼容现有 API 与引擎
  - 无论扩展规模如何，仅需调整一行代码即可使用现有的 pandas API 调用。无需执行 import pandas as pd，只需使用以下命令执行 import modin.pandas as pd：
    import modin.pandas as pd

```
# import pandas as pd
import modin.pandas as pd
```

```
df = pd.read_csv("my_dataset.csv")
```

  - 使用 Dask*、Ray 或 HEAVY.AI 计算引擎，无需编写代码即可分散数据
  - 继续使用 Python 生态系统中的其余代码，例如 NumPy、XGBoost 和 scikit-learn*
  - 使用同一记事本从本地设备扩展到云端

参考和下载网站：https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/distribution-of-modin.html

英特尔® Extension for Scikit-learn*

英特尔® Extension for Scikit-learn* 能够在单节点和多节点配置中，实现面向英特尔® CPU 和 GPU 的 scikit-learn 应用加速。该扩展包可在提升机器学习算法性能的同时，为 scikit-learn 估算器动态打补丁。

主要优势：

- 无学习全新 API 的前期成本
- 集成至 Python* 生态系统
- 与标准 scikit-learn 相比，性能和准确性提升高达 100 倍

进一步了解如何为 scikit-learn 打补丁

```
import numpy as np

# Turn on scikit-learn optimizations with these 2 simple lines:

from sklearnex import patch_sklearn

patch_sklearn()


# Import scikit-learn algorithms after the patch is enabled

from sklearn.cluster import KMeans


X = np.array([[1, 2], [1, 4], [1, 0],

 [10, 2], [10, 4], [10, 0]])

kmeans = KMeans(n_clusters=2, random_state=0).fit(X)

print(f"kmeans.labels_ = {kmeans.labels_}")
```

参考和下载网址：https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/scikit-learn.html

面向英特尔® 架构优化的 XGBoost*

XGBoost* 0.81 及更高版本已集成多项英特尔® 优化功能，可在英特尔® CPU 上实现更出色的性能。这一面向梯度提升决策树的知名机器学习包现包含面向英特尔® 架构优化的嵌入式加速技术，可显著提升模型训练速度和准确度，从而实现更好的预测结果。

参考和下载网址：https://www.intel.com/content/www/us/en/developer/articles/technical/xgboost-optimized-architecture-getting-started.html

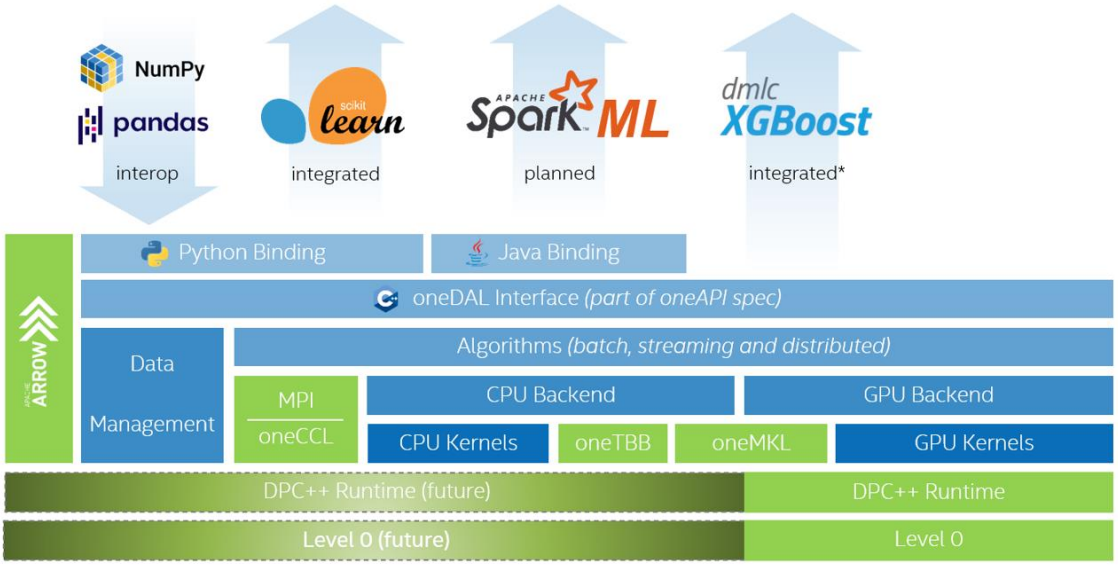
英特尔® oneAPI Data Analytics Library（英特尔® oneDAL）

英特尔® oneAPI Data Analytics Library（英特尔® oneDAL）是一个可助力加速大数据分析的库。它是通过以批量的、在线的、分布式的处理模式为数据分析的所有阶段（预处理、转换、分析、建模、验证和决策）提供高度优化的构建模块来实现这种加速的。

该库可在算法计算过程中优化数据获取，以增加吞吐量和可扩展性。它包括可以连接到 Spark* 和 Hadoop 等主流数据源的 C++ 和 Java* API 及连接器。面向英特尔® oneDAL 的 *Python* 包装器是英特尔® 分发版 Python 的组成部分。

除经典特性外，英特尔® oneDAL 还为传统 C++ 接口提供 DPC++ SYCL API 扩展，并使部分算法能够利用 GPU。

该库尤其适合分布式计算，因为其面向独立于任何通信层的分布式算法提供一整套构建模块，使用户能够使用各自偏好的通信方式，快速构建并扩展分布式应用。



参考和下载网址：https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/distribution-for-python.html

性能数据

查看第四代英特尔® 至强® 可扩展处理器和第三代英特尔® 至强® 可扩展处理器的最新性能数据（包括软硬件配置详情）。

参考资料

英特尔® AVX-512 相关信息：https://colfaxresearch.com/skl-avx512/

面向英特尔® 架构优化的 AI 框架：https://www.intel.cn/content/www/cn/zh/developer/tools/frameworks/overview.html