

インテル® Tiber™ AI クラウドで AI に PyTorch* と TensorFlow* を活用する

この記事は、インテルのブログで公開されている「[Harness PyTorch* and TensorFlow* for AI on Intel® Tiber™ Developer Cloud](#)」の日本語参考訳です。原文は更新される可能性があります。原文と翻訳文の内容が異なる場合は原文を優先してください。



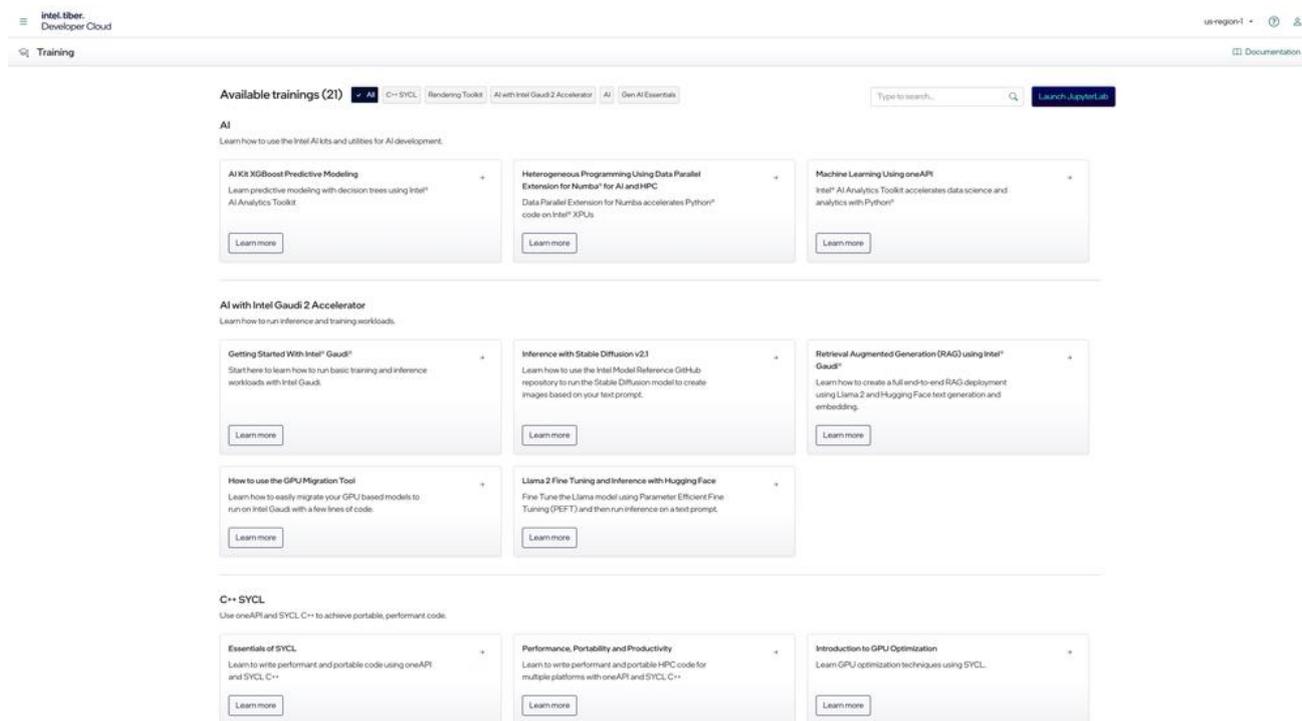
ディープラーニング (DL) は、人工知能 (AI) の一分野であり、ニューラル・ネットワークを使用してコンピューターが人間の神経プロセスを模倣し、複雑な問題や情報を処理できるようにします。PyTorch* と TensorFlow* は、大量のデータを処理する最も一般的な DL [フレームワーク](#) (英語) ですが、コードの実行方法が異なります。

インテルは、開発者が最新のインテル® ハードウェア上でインテルの最適化されたソフトウェアを使用して [AI 開発](#)を行い高速化できるインテル® Tiber™ AI クラウド (旧称: インテル® Tiber™ デベロッパー・クラウド) を提供しています。インテルはまた、[oneAPI](#) ライブラリーを使用して主要なマシンラーニング (ML) フレームワークと DL フレームワークを最適化して、インテル® アーキテクチャー全体で最高のパフォーマンスを発揮できるようにしています。これらのソフトウェア最適化により、ユーザーは同じフレームワークの標準実装よりも優れたパフォーマンスを実現できます。インテル® Tiber™ AI クラウドは、ML フレームワークを利用する AI アプリケーションとソリューションに、[インテル® Gaudi® 2 AI アクセラレーター](#)や[インテル® Xeon® スケーラブル・プロセッサ](#)を含む幅広いハードウェアへのアクセスを提供します。開発者は任意の CPU や GPU 上でワークロードを学習、プロトタイプ化、テスト、実行することができ、[無料の Jupyter* Notebook](#)と[チュートリアル](#) (英語) を通じてプラットフォームとソフトウェアの最適化をテストできます。

この記事では、インテル® Tiber™ AI クラウド上で PyTorch* と TensorFlow* を使用して DL ワークロードを構築および開発する方法を説明します。この記事の手順を実行する前に、インテル® Tiber™ AI クラウドを使い始める[ガイド](#)をお読みになることを推奨します。

インテル® Tiber™ AI クラウドの一般的な使用手順:

1. cloud.intel.com (英語) にサインインします。まだアカウントをお持ちでない場合は、[こちらの手順](#)に従ってアカウント登録を行ってください。
2. サインインしたら、コンソールの左上のメニューアイコンをクリックして **Training** (英語) を選択します。
3. 受講するトレーニングの **[Launch]** ボタンをクリックします。



JupyterLab* では、開発者のニーズに応じていくつかのカーネルタイプが用意されています。カーネルは事前にインストールされた Python* 環境であり、ユーザーが新しいノートブックを開くと、JupyterHub* は指定された環境に対応するパッケージのインストールを検出します。ほとんどの場合、以下のサンプルコードの実行に必要なパッケージは、ベースカーネルに含まれています。

インテル® Tiber™ AI クラウドでディープラーニングを始める

PyTorch* 向けインテル® エクステンション: PyTorch* は、ディープラーニングと AI 向けの Python* ベースのオープンソース・ライブラリです。[PyTorch* 向けインテル® エクステンション](#) (英語) は、最新の機能最適化で PyTorch* を拡張して、インテル® プロセッサ上の DL トレーニングと推論のパフォーマンスを最適化します。PyTorch* 向けインテル® エクステンションには、さまざまな生成 AI (GenAI) アプリケーション向けの大規模言語モデル (LLM) の特定の最適化も含まれています。

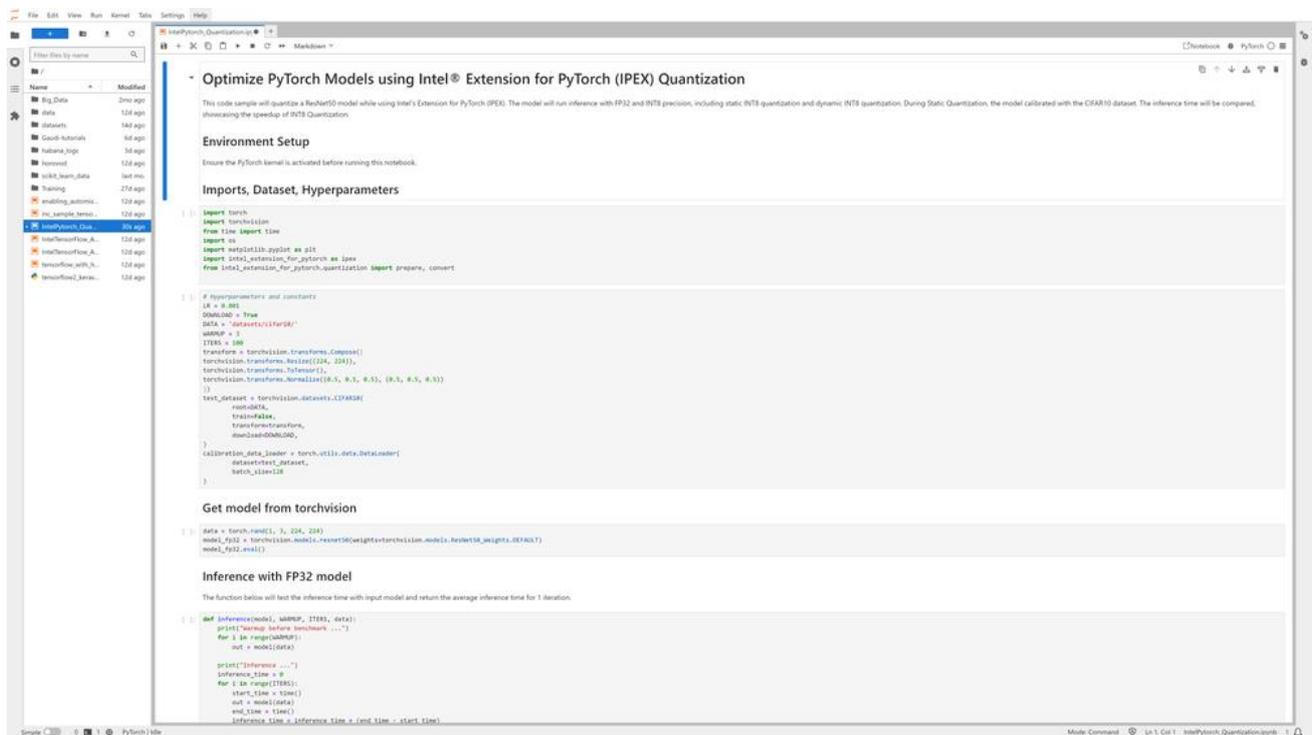
以下は、インテル® Tiber™ AI クラウドで PyTorch* 向けインテル® エクステンションの量子化サンプル (英語) を実行する手順です。

1. JupyterLab* を起動します。
2. ダッシュボードのメニューから [File] > [Open from URL...] を選択し、以下の URL をコピーしてペーストします。

https://raw.githubusercontent.com/oneapi-src/oneAPI-samples/master/AI-and-Analytics/Features-and-Functionality/IntelPytorch_Quantization/IntelPytorch_Quantization.ipynb

3. [Kernel] > [Change Kernel] > [PyTorch] を選択して、PyTorch カーネルに変更します。
4. サンプルコードのすべてのセルを実行し、出力を確認します。

このサンプルコードは、PyTorch* 向けインテル® エクステンション (IPEX) を利用して、ResNet50 モデルの量子化を行います。モデルは、静的 INT8 量子化と動的 INT8 量子化を含む FP32 および INT8 精度で推論を実行します。この例では、推論時間を比較し、INT8 量子化によるスピードアップを示します。



```
Optimize PyTorch Models using Intel® Extension for PyTorch (IPEX) Quantization

This code sample will quantize a ResNet50 model while using Intel's Extension for PyTorch (IPEX). The model will run inference with FP32 and INT8 precision, including static INT8 quantization and dynamic INT8 quantization. During Static Quantization, the model calibrated with the CIFAR10 dataset. The inference time will be compared, showcasing the speedup of INT8 Quantization.

Environment Setup
Ensure the PyTorch kernel is activated before running this notebook.

Imports, Dataset, Hyperparameters
import torch
import torchvision
from time import time
import os
import matplotlib.pyplot as plt
import intel_extension_for_pytorch as ipex
from intel_extension_for_pytorch.quantization import prepare, convert

# Hyperparameters and constants
it = 0.001
DOWNLOAD = True
DATA = "dataset/cifar10"
width = 3
ITER = 100
transform = torchvision.transforms.Compose([
    torchvision.transforms.Resize((224, 224)),
    torchvision.transforms.RandomCrop(1),
    torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
test_dataset = torchvision.datasets.CIFAR10(
    root=DATA,
    train=False,
    transform=transform,
    download=DOWNLOAD,
)
validation_data_loader = torch.utils.data.DataLoader(
    dataset=test_dataset,
    batch_size=128
)

Get model from torchvision
data = torch.randn(1, 3, 224, 224)
model_fp32 = torchvision.models.resnet50(torchvision.models.resnet50_weights.DEFAULT)
model_int8 = ipex.quantization.prepare(model_fp32)

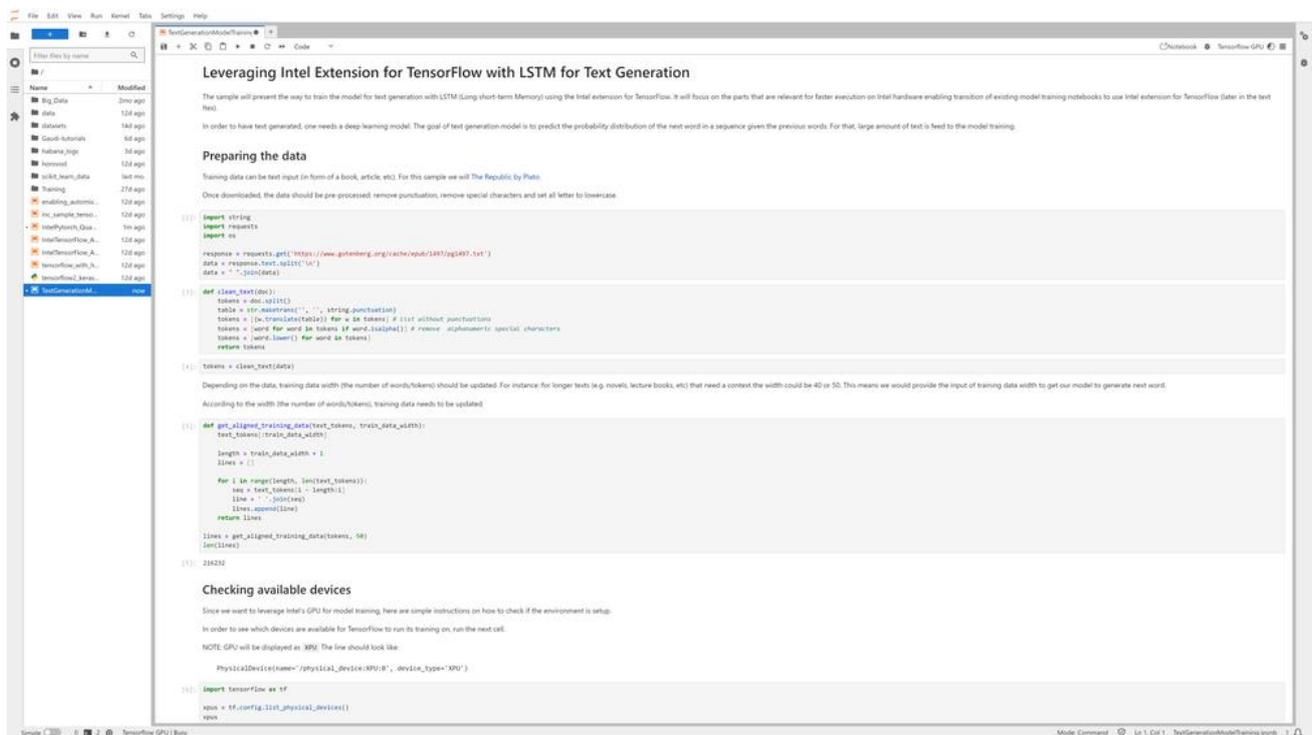
Inference with FP32 model
The function below will test the inference time with input model and return the average inference time for 1 iteration.
def inference(model, width, iter, data):
    print("Warmup before benchmark ...")
    for i in range(width):
        out = model(data)
    print("Inference ...")
    inference_time = 0
    for i in range(iter):
        start_time = time()
        out = model(data)
        end_time = time()
        inference_time = inference_time + (end_time - start_time)
```

TensorFlow* 向けインテル® エクステンション: TensorFlow* は、さまざまなアプリケーションで DL フレームワークを作成してデプロイするためのオープンソース・フレームワークです。**TensorFlow* 向けインテル® エクステンション** (英語) は、OpenMP* を利用して、CPU コア間でディープラーニングの実行を並列化します。この拡張機能により、ユーザーはオンデマンドで XPU を TensorFlow* に柔軟にプラグインし、インテル® ハードウェア内の計算能力を利用できます。

以下の手順に従って、インテル® Tiber™ AI クラウド上で TensorFlow* 向けインテル® エクステンションと LSTM を利用してテキスト生成サンプル (英語) を実行できます。

1. JupyterLab* を起動します。
2. ダッシュボードのメニューから **[File] > [Open from URL...]** を選択し、以下の URL をコピーしてペーストします。
https://raw.githubusercontent.com/oneapi-src/oneAPI-samples/master/AI-and-Analytics/Features-and-Functionality/IntelTensorFlow_TextGeneration_with_LSTM/TextGenerationModelTraining.ipynb
3. **[Kernel] > [Change Kernel] > [TensorFlow GPU]** を選択して、TensorFlow GPU カーネルに変更します。
4. サンプルコードのすべてのセルを実行し、出力を確認します。

このサンプルコードは、インテル® プロセッサ上で LSTM と TensorFlow* 向けインテル® エクステンションを利用してテキスト生成モデルをトレーニングします。テキスト生成モデルの主な目的は、与えられた入力に基づいてシーケンスの次の単語の確率分布を予測することです (より良い結果を得るには、テキスト文からの入力を提供してください)。TensorFlow* 向けインテル® エクステンションを使用することで、トレーニング時間が短縮され、GPU メモリーの消費量が少なくなります。



```
File Edit View Run Kernel Help
TextGenerationModelTraining.ipynb
Leveraging Intel Extension for TensorFlow with LSTM for Text Generation
The sample will present the way to train the model for text generation with LSTM (Long short-term Memory) using the Intel extension for TensorFlow. It will focus on the parts that are relevant for faster execution on Intel hardware enabling transition of existing model training notebooks to use Intel extension for TensorFlow (later in the text).
In order to have text generated, one needs a deep learning model. The goal of text generation model is to predict the probability distribution of the next word in a sequence given the previous words. For that, large amount of text is feed to the model training.
Preparing the data
Training data can be text input (in form of a book, article, etc). For this sample we will use The Republic by Plato.
Once downloaded, the data should be pre-processed remove punctuation, remove special characters and set all letter to lowercase.
%%writefile data.txt
import requests
import os
response = requests.get('https://www.gutenberg.org/cache/epub/1487/pg1487.txt')
data = response.text.split('\n')
data = "\n".join(data)
def clean_text(text):
    tokens = doc.split()
    table = str.maketrans('', '', string.punctuation)
    tokens = [w.translate(table) for w in tokens] # list without punctuations
    tokens = [word for word in tokens if word.isalpha()] # remove alpha numeric special characters
    return tokens
tokens = clean_text(data)
Depending on the data, training data width (the number of words/tokens) should be updated. For instance, for longer texts (e.g. novels, lecture books, etc) that need a context width could be 40 or 50. This means we would provide the input of training data width to get our model to generate next word.
According to the width (the number of words/tokens), training data needs to be updated.
def get_aligned_training_data(text_tokens, train_data_width):
    text_tokens = train_data_width
    length = len(text_tokens) - 1
    lines = []
    for i in range(length, len(text_tokens)):
        seq = text_tokens[i - length:i]
        lines.append(seq)
    return lines
lines = get_aligned_training_data(tokens, 50)
len(lines)
214232
Checking available devices
Since we want to leverage Intel's GPU for model training, here are simple instructions on how to check if the environment is setup.
In order to see which devices are available for TensorFlow to run its training on, run the next cell.
NOTE: GPU will be displayed as XPU. The line should look like:
PhysicalDeviceName="/physical_device:XPU:0", device_type="XPU"
import tensorflow as tf
xpus = tf.config.list_physical_devices('XPU')
```

PyTorch* 向けインテル® エクステンションの量子化サンプルと TensorFlow* 向けインテル® エクステンションは、AI ツールの一部として入手できます。また、インテル® Tiber™ AI クラウド上で scikit-learn*、Modin、XGBoost などの一般的な ML フレームワークを実行するためのガイド (英語) もあります。

インテル® Tiber™ AI クラウドで最新のシリコン・ハードウェアと最適化されたソフトウェアにアクセスし、革新的な AI プロジェクトを開発および強化してみてください。[インテルの AI ツールとフレームワークの最適化](#) (英語) をチェックし、[インテルの AI ソフトウェア・ポートフォリオ](#) (英語) の基盤となる統一されたオープンな標準ベースの [oneAPI](#) プログラミング・モデルについて学びましょう。また、業界をリードする独立系ソフトウェア・ベンダー (ISV)、システム・インテグレーター (Sler)、OEM、エンタープライズ・ユーザーとの [コラボレーション](#) (英語) が、AI の導入をどのように加速させるかをご覧ください。

エクセルソフトでは、インテル® Tiber™ AI クラウドの評価用クーポンを配布しています。ご興味のある方は、[こちらのフォーム](#)よりお問い合わせください。

関連資料

- [インテルの AI 開発ツールとリソース](#) (英語)
- [oneAPI 統一プログラミング・モデル](#)
- [インテル® Tiber™ AI クラウド](#)
- [インテルの AI ツール](#)
- [記事: インテル® Tiber™ AI クラウド上での ML ワークロードの構築と開発](#) (英語)
- [記事: PyTorch* 向けインテル® エクステンションの GPU 入門](#) (英語)
- [記事: TensorFlow* 向けインテル® エクステンション入門](#) (英語)
- [記事: インテル® Tiber™ AI クラウドで AI へのアクセスが容易に](#)
- [記事: インテル® Tiber™ AI クラウドで生成 AI と Jupyter* Notebooks を試してみる](#) (英語)