



TENSORRT SAMPLES

SWE-SWDOCTRT-001-SAMG_vTensorRT 7.0.0 | December 2019

Support Guide



TABLE OF CONTENTS

Chapter 1. Introduction.....	1
1.1. C++ Samples.....	4
1.2. Python Samples.....	5
Chapter 2. Application Areas.....	7
Chapter 3. Cross Compiling Samples For AArch64 Users.....	10
3.1. Prerequisites.....	10
3.2. Building Samples For QNX AArch64.....	11
3.3. Building Samples For Linux AArch64.....	11
3.4. Building Samples For Android AArch64.....	11
Chapter 4. “Hello World” For TensorRT.....	12
Chapter 5. Building A Simple MNIST Network Layer By Layer.....	13
Chapter 6. Importing The TensorFlow Model And Running Inference.....	14
Chapter 7. “Hello World” For TensorRT From ONNX.....	15
Chapter 8. Building And Running GoogleNet In TensorRT.....	16
Chapter 9. Building An RNN Network Layer By Layer.....	17
Chapter 10. Performing Inference In INT8 Using Custom Calibration.....	18
Chapter 11. Performing Inference In INT8 Precision.....	19
Chapter 12. Adding A Custom Layer To Your Network In TensorRT.....	20
Chapter 13. Object Detection With Faster R-CNN.....	21
Chapter 14. Object Detection With A TensorFlow SSD Network.....	22
Chapter 15. Movie Recommendation Using Neural Collaborative Filter (NCF).....	23
Chapter 16. Movie Recommendation Using MPS (Multi-Process Service).....	24
Chapter 17. Object Detection With SSD.....	25
Chapter 18. “Hello World” For Multilayer Perceptron (MLP).....	26
Chapter 19. Specifying I/O Formats Using The Reformat Free I/O APIs.....	27
Chapter 20. Adding A Custom Layer That Supports INT8 I/O To Your Network In TensorRT....	28
Chapter 21. Digit Recognition With Dynamic Shapes In TensorRT.....	29
Chapter 22. Neural Machine Translation (NMT) Using A Sequence To Sequence (seq2seq) Model.....	30
Chapter 23. Object Detection And Instance Segmentation With A TensorFlow Mask R-CNN Network.....	31
Chapter 24. Object Detection With A TensorFlow Faster R-CNN Network.....	33
Chapter 25. Introduction To Importing Caffe, TensorFlow And ONNX Models Into TensorRT Using Python.....	35
Chapter 26. “Hello World” For TensorRT Using TensorFlow And Python.....	36
Chapter 27. “Hello World” For TensorRT Using PyTorch And Python.....	37
Chapter 28. Adding A Custom Layer To Your Caffe Network In TensorRT In Python.....	38
Chapter 29. Adding A Custom Layer To Your TensorFlow Network In TensorRT In Python.....	39
Chapter 30. Object Detection With The ONNX TensorRT Backend In Python.....	40
Chapter 31. Object Detection With SSD In Python.....	41

Chapter 32. INT8 Calibration In Python.....	42
Chapter 33. Refitting An Engine In Python.....	43

Chapter 1.

INTRODUCTION

The following samples show how to use TensorRT in numerous use cases while highlighting different capabilities of the interface.

Title	TensorRT Sample Name	Description
trtexec	trtexec	A tool to quickly utilize TensorRT without having to develop your own application.
"Hello World" For TensorRT	sampleMNIST	Performs the basic setup and initialization of TensorRT using the Caffe parser.
Building A Simple MNIST Network Layer By Layer	sampleMNISTAPI	Uses the TensorRT API to build an MNIST (handwritten digit recognition) layer by layer, sets up weights and inputs/outputs and then performs inference.
Importing The TensorFlow Model And Running Inference	sampleUffMNIST	Imports a TensorFlow model trained on the MNIST dataset.
"Hello World" For TensorRT From ONNX	sampleOnnxMNIST	Converts a model trained on the MNIST dataset in ONNX format to a TensorRT network.
Building And Running GoogleNet In TensorRT	sampleGoogleNet	Shows how to import a model trained with Caffe into TensorRT using GoogleNet as an example.
Building An RNN Network Layer By Layer	sampleCharRNN	Uses the TensorRT API to build an RNN network layer by layer, sets up weights and inputs/outputs and then performs inference.
Performing Inference In INT8 Using Custom Calibration	sampleINT8	Performs INT8 calibration and inference. Calibrates a network for execution in INT8.
Performing Inference In INT8 Precision	sampleINT8API	Sets per tensor dynamic range and computation precision of a layer.

Title	TensorRT Sample Name	Description
Adding A Custom Layer To Your Network In TensorRT	samplePlugin	Defines a custom layer that supports multiple data formats that can be serialized and deserialized. Enables a custom layer in NvCaffeParser.
Object Detection With Faster R-CNN	sampleFasterRCNN	Uses TensorRT plugins, performs inference and implements a fused custom layer for end-to-end inferencing of a Faster R-CNN model.
Object Detection With A TensorFlow SSD Network	sampleUffSSD	Preprocess the TensorFlow SSD network, performs inference on the SSD network in TensorRT and uses TensorRT plugins to speed up inference.
Movie Recommendation Using Neural Collaborative Filter (NCF)	sampleMovieLens	An end-to-end sample that imports a trained TensorFlow model and predicts the highest-rated movie for each user.
Movie Recommendation Using MPS (Multi-Process Service)	sampleMovieLensMPS	An end-to-end sample that imports a trained TensorFlow model and predicts the highest-rated movie for each user using MPS (Multi-Process Service).
Object Detection With SSD	sampleSSD	Preprocess the input to the SSD network, performs inference on the SSD network in TensorRT, uses TensorRT plugins to speed up inference, and performs INT8 calibration on an SSD network.
“Hello World” For Multilayer Perceptron (MLP)	sampleMLP	Shows how to create a network that triggers the multi-layer perceptron (MLP) optimizer.
Specifying I/O Formats Using The Reformat Free I/O APIs	sampleReformatFreeIO	Uses a Caffe model that was trained on the MNIST dataset and performs engine building and inference using TensorRT. The correctness of outputs is then compared to the golden reference.
Adding A Custom Layer That Supports INT8 I/O To Your Network In TensorRT	sampleUffPluginV2Ext	Demonstrates how to extend INT8 I/O for a plugin that is introduced in TensorRT 6.x.x.
Digit Recognition With Dynamic Shapes In TensorRT	sampleDynamicReshape	Demonstrates how to use dynamic input dimensions in TensorRT by creating an engine for resizing dynamically shaped inputs to the correct size for an ONNX MNIST model.

Title	TensorRT Sample Name	Description
Neural Machine Translation (NMT) Using A Sequence To Sequence (seq2seq) Model	sampleNMT	Demonstrates the implementation of Neural Machine Translation (NMT) based on a TensorFlow seq2seq model using the TensorRT API.
Object Detection And Instance Segmentation With A TensorFlow Mask R-CNN Network	sampleUffMaskRCNN	Performs inference on the Mask R-CNN network in TensorRT. Mask R-CNN is based on the Mask R-CNN paper which performs the task of object detection and object mask predictions on a target image.
Object Detection With A TensorFlow Faster R-CNN Network	sampleUffFasterRCNN	Serves as a demo of how to use a pre-trained Faster-RCNN model in Transfer Learning Toolkit to do inference with TensorRT.
Introduction To Importing Caffe, TensorFlow And ONNX Models Into TensorRT Using Python	introductory_parser_samples	Uses TensorRT and its included suite of parsers (the UFF, Caffe and ONNX parsers), to perform inference with ResNet-50 models trained with various different frameworks.
“Hello World” For TensorRT Using TensorFlow And Python	end_to_end_tensorflow_mnist	An end-to-end sample that trains a model in TensorFlow and Keras, freezes the model and writes it to a protobuf file, converts it to UFF, and finally runs inference using TensorRT.
“Hello World” For TensorRT Using PyTorch And Python	network_api_pytorch_mnist	An end-to-end sample that trains a model in PyTorch, recreates the network in TensorRT, imports weights from the trained model, and finally runs inference with a TensorRT engine.
Adding A Custom Layer To Your Caffe Network In TensorRT In Python	fc_plugin_caffe_mnist	Implements a FullyConnected layer using cuBLAS and cuDNN, wraps the implementation in a TensorRT plugin (with a corresponding plugin factory) and generates Python bindings for it using <code>pybind11</code> . These bindings are then used to register the plugin factory with the CaffeParser.
Adding A Custom Layer To Your TensorFlow Network In TensorRT In Python	uff_custom_plugin	Implements a clip layer (as a CUDA kernel) wraps the implementation in a TensorRT plugin (with a corresponding plugin creator), and generates a shared library module containing its code.

Title	TensorRT Sample Name	Description
Object Detection With The ONNX TensorRT Backend In Python	yolov3_onnx	Implements a full ONNX-based pipeline for performing inference with the YOLOv3-608 network, including pre and post-processing.
Object Detection With SSD In Python	uff_ssd	Implements a full UFF-based pipeline for performing inference with an SSD (InceptionV2 feature extractor) network. The sample downloads a trained <code>ssd_inception_v2_coco_2017_11_17</code> model and uses it to perform inference. Additionally, it superimposes bounding boxes on the input image as a post-processing step.
INT8 Calibration In Python	int8_caffe_mnist	Demonstrates how to calibrate an engine to run in INT8 mode.
Refitting An Engine In Python	engine_refit_mnist	Trains an MNIST model in PyTorch, recreates the network in TensorRT with dummy weights, and finally refits the TensorRT engine with weights from the model.

1.1. C++ Samples

You can find the C++ samples in the `/usr/src/tensorrt/samples` package directory as well as on GitHub. The following C++ samples are shipped with TensorRT.

- ▶ “Hello World” For TensorRT
- ▶ Building A Simple MNIST Network Layer By Layer
- ▶ Importing The TensorFlow Model And Running Inference
- ▶ “Hello World” For TensorRT From ONNX
- ▶ Building And Running GoogleNet In TensorRT
- ▶ Building An RNN Network Layer By Layer
- ▶ Performing Inference In INT8 Using Custom Calibration
- ▶ Performing Inference In INT8 Precision
- ▶ Adding A Custom Layer To Your Network In TensorRT
- ▶ Object Detection With Faster R-CNN
- ▶ Object Detection With A TensorFlow SSD Network
- ▶ Movie Recommendation Using Neural Collaborative Filter (NCF)
- ▶ Movie Recommendation Using MPS (Multi-Process Service)
- ▶ Object Detection With SSD
- ▶ “Hello World” For Multilayer Perceptron (MLP)

- ▶ [Specifying I/O Formats Using The Reformat Free I/O APIs](#)
- ▶ [Adding A Custom Layer That Supports INT8 I/O To Your Network In TensorRT](#)
- ▶ [Digit Recognition With Dynamic Shapes In TensorRT](#)
- ▶ [Neural Machine Translation \(NMT\) Using A Sequence To Sequence \(seq2seq\) Model](#)
- ▶ [Object Detection And Instance Segmentation With A TensorFlow Mask R-CNN Network¹](#)
- ▶ [Object Detection With A TensorFlow Faster R-CNN Network²](#)

Getting Started With C++ Samples

Every C++ sample includes a **README.md** file in [GitHub](#) that provides detailed information about how the sample works, sample code, and step-by-step instructions on how to run and verify its output.

Running C++ Samples on Linux

If you installed TensorRT using the Debian files, copy `/usr/src/tensorrt` to a new directory first before building the C++ samples. If you installed TensorRT using the tar file, then the samples are located in `{TAR_EXTRACT_PATH}/samples`. To build all the samples and then run one of the samples, use the following commands:

```
$ cd <samples_dir>
$ make -j4
$ cd ../bin
$ ./<sample_bin>
```

Running C++ Samples on Windows

All of the C++ samples on Windows are provided as Visual Studio Solution files. To build a sample, open its corresponding Visual Studio Solution file and build the solution. The output executable will be generated in `(ZIP_EXTRACT_PATH)\bin`. You can then run the executable directly or through Visual Studio.

1.2. Python Samples

You can find the Python samples in the `/usr/src/tensorrt/samples/python` package directory. The following Python samples are shipped with TensorRT.

- ▶ [Introduction To Importing Caffe, TensorFlow And ONNX Models Into TensorRT Using Python](#)
- ▶ ["Hello World" For TensorRT Using TensorFlow And Python](#)
- ▶ ["Hello World" For TensorRT Using PyTorch And Python](#)
- ▶ [Adding A Custom Layer To Your Caffe Network In TensorRT In Python](#)
- ▶ [Adding A Custom Layer To Your TensorFlow Network In TensorRT In Python](#)

¹ This sample is located in GitHub only; this is not part of the product package.

² 1

- ▶ [Object Detection With The ONNX TensorRT Backend In Python](#)
- ▶ [Object Detection With SSD In Python](#)
- ▶ [INT8 Calibration In Python](#)
- ▶ [Refitting An Engine In Python](#)

Getting Started With Python Samples

Every Python sample includes a **README.md** file. Refer to the `/usr/src/tensorrt/samples/python/<sample-name>/README.md` file for detailed information about how the sample works, sample code, and step-by-step instructions on how to run and verify its output.

Running Python Samples

To run one of the Python samples, the process typically involves two steps:

1. Install the sample requirements:

```
python<x> -m pip install -r requirements.txt
```

where `python<x>` is either `python2` or `python3`.

2. Run the sample code with the `data` directory provided if the TensorRT sample data is not in the default location. For example:

```
python<x> sample.py [-d DATA_DIR]
```

For more information on running samples, see the **README.md** file included with the sample.

Chapter 2.

APPLICATION AREAS

The TensorRT samples focus on the following application areas.

Recommenders

Recommender systems are used to provide product or media recommendations to users of social networking, media content consumption, and e-commerce platforms. MLP-based Neural Collaborative Filter (NCF) recommenders employ a stack of fully-connected or matrix multiplication layers to generate recommendations.

Some examples of TensorRT recommenders samples include the following:

- ▶ [Movie Recommendation Using Neural Collaborative Filter \(NCF\)](#)
- ▶ [Movie Recommendation Using MPS \(Multi-Process Service\)](#)
- ▶ [“Hello World” For Multilayer Perceptron \(MLP\)](#)

Machine translation

Machine translation systems are used to translate text from one language to another language. Recurrent neural networks (RNN) are one of the most popular deep learning solutions for machine translation.

Some examples of TensorRT machine translation samples include the following:

- ▶ [Neural Machine Translation \(NMT\) Using A Sequence To Sequence \(seq2seq\) Model](#)
- ▶ [Building An RNN Network Layer By Layer](#)

Character recognition

Character recognition, especially on the MNIST dataset, is a classic machine learning problem. The MNIST problem involves recognizing the digit that is present in an image of a handwritten digit.

Some examples of TensorRT character recognition samples include the following:

- ▶ [“Hello World” For TensorRT](#)

- ▶ [Building A Simple MNIST Network Layer By Layer](#)
- ▶ [Importing The TensorFlow Model And Running Inference](#)
- ▶ [“Hello World” For TensorRT From ONNX](#)
- ▶ [Performing Inference In INT8 Using Custom Calibration](#)
- ▶ [Adding A Custom Layer To Your Network In TensorRT](#)
- ▶ [Digit Recognition With Dynamic Shapes In TensorRT](#)
- ▶ [Specifying I/O Formats Using The Reformat Free I/O APIs](#)
- ▶ [Adding A Custom Layer That Supports INT8 I/O To Your Network In TensorRT](#)
- ▶ [“Hello World” For TensorRT Using TensorFlow And Python](#)
- ▶ [Refitting An Engine In Python](#)
- ▶ [Adding A Custom Layer To Your Caffe Network In TensorRT In Python](#)
- ▶ [INT8 Calibration In Python](#)
- ▶ [“Hello World” For TensorRT Using PyTorch And Python](#)
- ▶ [Adding A Custom Layer To Your TensorFlow Network In TensorRT In Python](#)

Image classification

Image classification is the problem of identifying one or more objects present in an image. Convolutional neural networks (CNN) are a popular choice for solving this problem. They are typically composed of convolution and pooling layers.

Some examples of TensorRT image classification samples include the following:

- ▶ [Building And Running GoogleNet In TensorRT](#)
- ▶ [Performing Inference In INT8 Precision](#)
- ▶ [Introduction To Importing Caffe, TensorFlow And ONNX Models Into TensorRT Using Python](#)

Object detection

Object detection is one of the classic computer vision problems. The task, for a given image, is to detect, classify and localize all objects of interest. For example, imagine that you are developing a self-driving car and you need to do pedestrian detection - the object detection algorithm would then, for a given image, return bounding box coordinates for each pedestrian in an image.

There have been many advances in recent years in designing models for object detection.

Some examples of TensorRT object detection samples include the following:

- ▶ [Object Detection With SSD In Python](#)
- ▶ [Object Detection With The ONNX TensorRT Backend In Python](#)
- ▶ [Object Detection With A TensorFlow SSD Network](#)
- ▶ [Object Detection With Faster R-CNN](#)
- ▶ [Object Detection With SSD](#)

- ▶ Object Detection And Instance Segmentation With A TensorFlow Mask R-CNN Network
- ▶ Object Detection With A TensorFlow Faster R-CNN Network

Chapter 3.

CROSS COMPILING SAMPLES FOR AARCH64 USERS

The following sections show how to cross-compile TensorRT samples for AArch64 QNX, Linux and Android platforms under x86_64 Linux.

3.1. Prerequisites

This section provides step-by-step instructions to ensure you meet the minimum requirements to cross-compile.

1. Install the CUDA cross-platform toolkit for the corresponding target and set the environment variable `CUDA_INSTALL_DIR`.

```
$ export CUDA_INSTALL_DIR="your cuda install dir"
```

Where `CUDA_INSTALL_DIR` is set to `/usr/local/cuda` by default.

2. Install the cuDNN cross-platform libraries for the corresponding target and set the environment variable `CUDNN_INSTALL_DIR`.

```
$ export CUDNN_INSTALL_DIR="your cudnn install dir"
```

Where `CUDNN_INSTALL_DIR` is set to `CUDA_INSTALL_DIR` by default.

3. Install the TensorRT cross-compilation Debian packages for the corresponding target.



If you are using the tar file release for the target platform, then you can safely skip this step. The tar file release already includes the cross-compile libraries so no additional packages are required.

QNX AArch64

```
libnvinfer-dev-cross-qnx, libnvinfer5-cross-qnx
```

Linux AArch64

```
libnvinfer-dev-cross-aarch64, libnvinfer5-cross-aarch64
```

Android AArch64

No debian packages are available.

3.2. Building Samples For QNX AArch64

This section provides step-by-step instructions to build samples for QNX users.

1. Download the QNX tool-chain and export the following environment variables.

```
$ export QNX_HOST=/path/to/your/qnx/toolchains/host/linux/x86_64
$ export QNX_TARGET=/path/to/your/qnx/toolchain/target/qnx7
```

2. Build the samples by issuing:

```
$ cd /path/to/TensorRT/samples
$ make TARGET=qnx
```

3.3. Building Samples For Linux AArch64

This section provides step-by-step instructions to build samples for Linux users.

1. Install the corresponding GCC compiler, `aarch64-linux-gnu-g++`. In Ubuntu, this can be installed via:

```
$ sudo apt-get install g++-aarch64-linux-gnu
```

2. Build the samples by issuing:

```
$ cd /path/to/TensorRT/samples
$ make TARGET=aarch64
```

3.4. Building Samples For Android AArch64

This section provides step-by-step instructions to build samples for Android users.

1. Download the Android NDK (r16b) from <https://developer.android.com/ndk/>.
2. Create a standalone tool-chain, for example:

```
$ $NDK/build/tools/make_standalone_toolchain.py \
  --arch arm64 \
  --api 26 \
  --install-dir=/path/to/my-toolchain
```

You can find more information by visiting: https://developer.android.com/ndk/guides/standalone_toolchain

3. Build the samples by issuing:

```
$ cd /path/to/TensorRT/samples
$ make TARGET=android64 ANDROID_CC=/path/to/my-toolchain/bin/aarch64-linux-
android-clang++
```

Chapter 4.

“HELLO WORLD” FOR TENSORRT

This sample, `sampleMNIST`, is a simple hello world example that performs the basic setup and initialization of TensorRT using the Caffe parser.

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleMNIST` directory in the [GitHub: sampleMNIST](#) repository.

How do I get started?

Refer to the [GitHub: sampleMNIST/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 5.

BUILDING A SIMPLE MNIST NETWORK LAYER BY LAYER

This sample, `sampleMNISTAPI`, uses the TensorRT API to build an engine for a model trained on the MNIST dataset.

What does this sample do?

Specifically, it creates the network layer by layer, sets up weights and inputs/outputs, and then performs inference. This sample is similar to `sampleMNIST`. Both of these samples use the same model weights, handle the same input, and expect similar output.

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleMNISTAPI` directory in the [GitHub: sampleMNISTAPI](#) repository.

How do I get started?

Refer to the [GitHub: sampleMNISTAPI/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 6.

IMPORTING THE TENSORFLOW MODEL AND RUNNING INFERENCE

This sample, `sampleUffMNIST`, imports a TensorFlow model trained on the MNIST dataset.

What does this sample do?

The MNIST TensorFlow model has been converted to UFF (Universal Framework Format) using the explanation described in [Working With TensorFlow](#).

The UFF is designed to store neural networks as a graph. The `NvUffParser` that we use in this sample parses the UFF file in order to create an inference engine based on that neural network.

With TensorRT, you can take a TensorFlow trained model, export it into a UFF protobuf file (`.uff`) using the [UFF converter](#), and import it using the UFF parser.

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleUffMNIST` directory in the [GitHub: sampleUffMNIST](#) repository.

How do I get started?

Refer to the [GitHub: sampleUffMNIST/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 7.

“HELLO WORLD” FOR TENSORRT FROM ONNX

This sample, `sampleOnnxMNIST`, converts a model trained on the MNIST in Open Neural Network Exchange (ONNX) format to a TensorRT network and runs inference on the network. ONNX is a standard for representing deep learning models that enables models to be transferred between frameworks.

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleOnnxMNIST` directory in the [GitHub: sampleOnnxMNIST](#) repository.

How do I get started?

Refer to the [GitHub: sampleOnnxMNIST/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 8.

BUILDING AND RUNNING GOOGLNET IN TENSORRT

This sample, `sampleGoogleNet`, demonstrates how to import a model trained with Caffe into TensorRT using GoogleNet as an example.

What does this sample do?

Specifically, this sample builds a TensorRT engine from the saved Caffe model, sets input values to the engine, and runs it.

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleGoogleNet` directory in the [GitHub: sampleGoogleNet](#) repository.

How do I get started?

Refer to the [GitHub: sampleGoogleNet/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 9.

BUILDING AN RNN NETWORK LAYER BY LAYER

This sample, `sampleCharRNN`, uses the TensorRT API to build an RNN network layer by layer, sets up weights and inputs/outputs and then performs inference.

What does this sample do?

Specifically, this sample creates a CharRNN network that has been trained on the [Tiny Shakespeare](#) dataset. For more information about character level modeling, see [char-rnn](#).

TensorFlow has a useful [RNN Tutorial](#) which can be used to train a word-level model. Word level models learn a probability distribution over a set of all possible word sequences. Since our goal is to train a char level model, which learns a probability distribution over a set of all possible characters, a few modifications will need to be made to get the TensorFlow sample to work. These modifications can be seen [here](#).

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleCharRNN` directory in the [GitHub: sampleCharRNN](#) repository.

How do I get started?

Refer to the [GitHub: sampleCharRNN/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 10.

PERFORMING INFERENCE IN INT8 USING CUSTOM CALIBRATION

This sample, `sampleINT8`, performs INT8 calibration and inference.

What does this sample do?

Specifically, this sample demonstrates how to perform inference in an 8-bit integer (INT8). INT8 inference is available only on GPUs with compute capability 6.1 or 7.x. After the network is calibrated for execution in INT8, the output of the calibration is cached to avoid repeating the process. You can then reproduce your own experiments with Caffe in order to validate your results on ImageNet networks.

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleINT8` directory in the [GitHub: sampleINT8](#) repository.

How do I get started?

Refer to the [GitHub: sampleINT8/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 11.

PERFORMING INFERENCE IN INT8 PRECISION

This sample, `sampleINT8API`, performs INT8 inference without using the INT8 calibrator; using the user-provided per activation tensor dynamic range. INT8 inference is available only on GPUs with compute capability 6.1 or 7.x and supports Image Classification ONNX models such as ResNet-50, VGG19, and MobileNet.

What does this sample do?

Specifically, this sample demonstrates how to:

- ▶ Use `nvinfer1::ITensor::setDynamicRange` to set per tensor dynamic range
- ▶ Use `nvinfer1::ILayer::setPrecision` to set computation precision of a layer
- ▶ Use `nvinfer1::ILayer::setOutputType` to set output tensor data type of a layer
- ▶ Perform INT8 inference without using INT8 calibration

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleINT8API` directory in the [GitHub: sampleINT8API](#) repository.

How do I get started?

Refer to the [GitHub: sampleINT8API/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 12.

ADDING A CUSTOM LAYER TO YOUR NETWORK IN TENSORRT

This sample, `samplePlugin`, defines a custom layer that supports multiple data formats and demonstrates how to serialize/deserialize plugin layers. This sample also demonstrates how to use a fully connected plugin (**FCPlugin**) as a custom layer and the integration with `NvCaffeParser`.

Where is this sample located?

This sample is maintained under the `samples/opensource/samplePlugin` directory in the [GitHub: samplePlugin](#) repository.

How do I get started?

Refer to the [GitHub: samplePlugin/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 13.

OBJECT DETECTION WITH FASTER R-CNN

This sample, `sampleFasterRCNN`, uses TensorRT plugins, performs inference, and implements a fused custom layer for end-to-end inferencing of a Faster R-CNN model.

What does this sample do?

Specifically, this sample demonstrates the implementation of a Faster R-CNN network in TensorRT, performs a quick performance test in TensorRT, implements a fused custom layer, and constructs the basis for further optimization, for example using INT8 calibration, user trained network, etc. The Faster R-CNN network is based on the paper [Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks](#).

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleFasterRCNN` directory in the [GitHub: sampleFasterRCNN](#) repository.

How do I get started?

Refer to the [GitHub: sampleFasterRCNN/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 14.

OBJECT DETECTION WITH A TENSORFLOW SSD NETWORK

This sample, `sampleUffSSD`, preprocesses a TensorFlow SSD network, performs inference on the SSD network in TensorRT, using TensorRT plugins to speed up inference.

What does this sample do?

This sample is based on the [SSD: Single Shot MultiBox Detector](#) paper. The SSD network performs the task of object detection and localization in a single forward pass of the network.

The SSD network used in this sample is based on the TensorFlow implementation of SSD, which actually differs from the original paper, in that it has an `inception_v2` backbone. For more information about the actual model, download [ssd_inception_v2_coco](#). The TensorFlow SSD network was trained on the InceptionV2 architecture using the [MSCOCO dataset](#) which has 91 classes (including the background class). The config details of the network can be found [here](#).

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleUffSSD` directory in the [GitHub: sampleUffSSD](#) repository.

How do I get started?

Refer to the [GitHub: sampleUffSSD/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 15.

MOVIE RECOMMENDATION USING NEURAL COLLABORATIVE FILTER (NCF)

This sample, `sampleMovieLens`, is an end-to-end sample that imports a trained TensorFlow model and predicts the highest-rated movie for each user. This sample demonstrates a simple movie recommender system using a multi-layer perceptron (MLP) based Neural Collaborative Filter (NCF) recommender.

What does this sample do?

Specifically, this sample demonstrates how to generate weights for a MovieLens dataset that TensorRT can then accelerate.

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleMovieLens` directory in the [GitHub: sampleMovieLens](#) repository.

How do I get started?

Refer to the [GitHub: sampleMovieLens/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 16.

MOVIE RECOMMENDATION USING MPS (MULTI-PROCESS SERVICE)

This sample, `sampleMovieLensMPS`, is an end-to-end sample that imports a trained TensorFlow model and predicts the highest-rated movie for each user using MPS (Multi-Process Service).

What does this sample do?

MPS allows multiple CUDA processes to share a single GPU context. With MPS, multiple overlapping kernel execution and `memcpy` operations from different processes can be scheduled concurrently to achieve maximum utilization. This can be especially effective in increasing parallelism for small networks with low resource utilization such as those primarily consisting of a series of small MLPs.

This sample is identical to [Movie Recommendation Using Neural Collaborative Filter \(NCF\)](#) in terms of functionality but is modified to support concurrent execution in multiple processes. Specifically, this sample demonstrates how to generate weights for a MovieLens dataset that TensorRT can then accelerate.



Currently, `sampleMovieLensMPS` supports only Linux x86-64 (includes Ubuntu and RedHat) desktop users.

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleMovieLensMPS` directory in the [GitHub: sampleMovieLensMPS](#) repository.

How do I get started?

Refer to the [GitHub: sampleMovieLensMPS/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 17.

OBJECT DETECTION WITH SSD

This sample, sampleSSD, performs the task of object detection and localization in a single forward pass of the network.

What does this sample do?

This sample is based on the [SSD: Single Shot MultiBox Detector](#) paper. This network is built using the VGG network as a backbone and trained using [PASCAL VOC 2007+ 2012](#) datasets.

Unlike Faster R-CNN, SSD completely eliminates the proposal generation and subsequent pixel or feature resampling stages and encapsulates all computation in a single network. This makes SSD straightforward to integrate into systems that require a detection component.

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleSSD` directory in the [GitHub: sampleSSD](#) repository.

How do I get started?

Refer to the [GitHub: sampleSSD/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 18.

“HELLO WORLD” FOR MULTILAYER PERCEPTRON (MLP)

This sample, `sampleMLP`, is a simple hello world example that shows how to create a network that triggers the multilayer perceptron (MLP) optimizer. The generated MLP optimizer can then accelerate TensorRT.

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleMLP` directory in the [GitHub: sampleMLP](#) repository.

How do I get started?

Refer to the [GitHub: sampleMLP/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 19.

SPECIFYING I/O FORMATS USING THE REFORMAT FREE I/O APIS

This sample, `sampleReformatFreeIO`, uses a Caffe model that was trained on the MNIST dataset and performs engine building and inference using TensorRT. The correctness of outputs is then compared to the golden reference.

What does this sample do?

Specifically, this sample shows how to use reformat free I/O APIs to explicitly specify I/O formats to `TensorFormat::kLINEAR`, `TensorFormat::kCHW2` and `TensorFormat::kHWC8` for Float16 and INT8 precision.

`ITensor::setAllowedFormats` is invoked to specify which format is expected to be supported so that the unnecessary reformatting will not be inserted to convert from/to FP32 formats for I/O tensors. `BuilderFlag::kSTRICT_TYPES` is also assigned to the builder configuration to let the builder choose a reformat free path rather than the fastest path.

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleReformatFreeIO` directory in the [GitHub: sampleReformatFreeIO](#) repository.

How do I get started?

Refer to the [GitHub: sampleReformatFreeIO/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 20.

ADDING A CUSTOM LAYER THAT SUPPORTS INT8 I/O TO YOUR NETWORK IN TENSORRT

This sample, `sampleUffPluginV2Ext`, implements the custom pooling layer for the MNIST model (`data/samples/lenet5_custom_pool.uff`).

What does this sample do?

Since cuDNN function `cudaPoolingForward` with float precision is used to simulate an INT8 kernel, the performance for INT8 precision does not speed up. Nevertheless, the main purpose of this sample is to demonstrate how to extend INT8 I/O for a plugin that is introduced in TensorRT 6.0. This requires the interface replacement from `IPlugin/IPluginV2/IPluginV2Ext` to `IPluginV2IOExt` (or `IPluginV2DynamicExt` if dynamic shape is required).

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleUffPluginV2Ext` directory in the [GitHub: sampleUffPluginV2Ext](#) repository.

How do I get started?

Refer to the [GitHub: /sampleUffPluginV2Ext/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 21.

DIGIT RECOGNITION WITH DYNAMIC SHAPES IN TENSORRT

This sample, `sampleDynamicReshape`, demonstrates how to use dynamic input dimensions in TensorRT by creating an engine for resizing dynamically shaped inputs to the correct size for an ONNX MNIST model.

What does this sample do?

This sample creates an engine for resizing an input with dynamic dimensions to a size that an ONNX MNIST model can consume.

Specifically, this sample demonstrates how to:

- ▶ Create a network with dynamic input dimensions to act as a preprocessor for the model
- ▶ Parse an ONNX MNIST model to create a second network
- ▶ Build engines for both networks
- ▶ Run inference using both engines

For more information, see [Working With Dynamic Shapes](#) in the TensorRT Developer Guide.

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleDynamicReshape` directory in the [GitHub: sampleDynamicReshape](#) repository.

How do I get started?

Refer to the [GitHub: sampleDynamicReshape/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 22.

NEURAL MACHINE TRANSLATION (NMT) USING A SEQUENCE TO SEQUENCE (SEQ2SEQ) MODEL

This sample, `sampleNMT`, demonstrates the implementation of Neural Machine Translation (NMT) based on a TensorFlow `seq2seq` model using the TensorRT API. The TensorFlow `seq2seq` model is an open-sourced NMT project that uses deep neural networks to translate text from one language to another language.

What does this sample do?

Specifically, this sample is an end-to-end sample that takes a TensorFlow model, builds an engine, and runs inference using the generated network. The sample is intended to be modular so it can be used as a starting point for your machine translation application.

This sample implements German to English translation using the data that is provided by and trained from the [TensorFlow NMT \(seq2seq\) Tutorial](#).

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleNMT` directory in the [GitHub: sampleNMT](#) repository.

How do I get started?

Refer to the [GitHub: sampleNMT/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 23.

OBJECT DETECTION AND INSTANCE SEGMENTATION WITH A TENSORFLOW MASK R-CNN NETWORK

This sample, `sampleUffMaskRCNN`, performs inference on the Mask R-CNN network in TensorRT.

What does this sample do?

Mask R-CNN is based on the [Mask R-CNN](#) paper which performs the task of object detection and object mask predictions on a target image.

This sample's model is based on the Keras implementation of Mask R-CNN and its training framework can be found in the [Mask R-CNN Github repository](#). We have verified that the pre-trained Keras model (with backbone ResNet101 + FPN and dataset coco) provided in the [v2.0](#) release can be converted to UFF and consumed by this sample. And, it is also feasible to deploy your customized Mask R-CNN model trained with specific backbone and datasets.



This sample is available only in GitHub and is not packaged with the product.

This sample makes use of TensorRT plugins to run the Mask R-CNN model. To use these plugins, the Keras model should be converted to Tensorflow `.pb` model. Then this `.pb` model needs to be preprocessed and converted to the UFF model with the help of GraphSurgeon and the UFF utility.

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleUffMaskRCNN` directory in the [GitHub: sampleUffMaskRCNN](#) repository.

How do I get started?

Refer to the [GitHub: sampleUffMaskRCNN/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 24.

OBJECT DETECTION WITH A TENSORFLOW FASTER R-CNN NETWORK

This sample, `sampleUffFasterRCNN`, serves as a demo of how to use the pre-trained Faster-RCNN model in Transfer Learning Toolkit to do inference with TensorRT.

What does this sample do?

This sample is a UFF TensorRT sample for Faster-RCNN in [NVIDIA Transfer Learning Toolkit SDK](#). Besides the sample itself, it also provides two TensorRT plugins: [Proposal](#) and [CropAndResize](#) to implement the proposal layer and ROI Pooling layer as custom layers in the model since TensorRT has no native support for them.



This sample is available only in GitHub and is not packaged with the product.

In this sample, we provide a UFF model as a demo. While in the Transfer Learning Toolkit workflow, we can't provide the UFF model. Instead, we can only get the `.tlt` model during training and the `.etlt` model after `tlt-export`. Both of them are encrypted models and the Transfer Learning Toolkit user will use `tlt-converter` to decrypt the `.etlt` model and generate a TensorRT engine file in a single step. Therefore, in the Transfer Learning Toolkit workflow, we will consume the TensorRT engine instead of a UFF model. However, this sample can still serve as a demo on how to use the UFF Faster R-CNN model regardless of its format.

Where is this sample located?

This sample is maintained under the `samples/opensource/sampleUffFasterRCNN` directory in the [GitHub: sampleUffFasterRCNN](#) repository.

How do I get started?

Refer to the [GitHub: sampleUffFasterRCNN/README.md](#) file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

Chapter 25.

INTRODUCTION TO IMPORTING CAFFE, TENSORFLOW AND ONNX MODELS INTO TENSORRT USING PYTHON

This sample, `introductory_parser_samples`, is a Python sample that uses TensorRT and its included suite of parsers (UFF, Caffe and ONNX parsers), to perform inference with ResNet-50 models trained with various different frameworks.

Where Is This Sample Located?

This sample is installed in the `/usr/src/tensorrt/samples/python/introductory_parser_samples` directory.

Getting Started:

Refer to the `/usr/src/tensorrt/samples/python/introductory_parser_samples/README.md` file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

A summary of the `README.md` file is included in this section for your reference, however, you should always refer to the `README.md` within the package for the most recent documentation updates.

Chapter 26.

“HELLO WORLD” FOR TENSORRT USING TENSORFLOW AND PYTHON

This sample, `end_to_end_tensorflow_mnist`, trains a small, fully-connected model on the MNIST dataset and runs inference using TensorRT.

Where Is This Sample Located?

This sample is installed in the `/usr/src/tensorrt/samples/python/end_to_end_tensorflow_mnist` directory.

Getting Started:

Refer to the `/usr/src/tensorrt/samples/python/end_to_end_tensorflow_mnist/README.md` file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

A summary of the `README.md` file is included in this section for your reference, however, you should always refer to the `README.md` within the package for the most recent documentation updates.

Chapter 27.

“HELLO WORLD” FOR TENSORRT USING PYTORCH AND PYTHON

This sample, `network_api_pytorch_mnist`, trains a convolutional model on the MNIST dataset and runs inference with a TensorRT engine.

Where Is This Sample Located?

This sample is installed in the `/usr/src/tensorrt/samples/python/network_api_pytorch_mnist` directory.

Getting Started:

Refer to the `/usr/src/tensorrt/samples/python/network_api_pytorch_mnist/README.md` file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

A summary of the `README.md` file is included in this section for your reference, however, you should always refer to the `README.md` within the package for the most recent documentation updates.

Chapter 28.

ADDING A CUSTOM LAYER TO YOUR CAFFE NETWORK IN TENSORRT IN PYTHON

This sample, `fc_plugin_caffe_mnist`, demonstrates how to implement a custom FullyConnected layer using cuBLAS and cuDNN, wraps the implementation in a TensorRT plugin (with a corresponding plugin factory) and generates Python bindings for it using `pybind11`. These bindings are then used to register the plugin factory with the CaffeParser.



The Caffe InnerProduct/FullyConnected layer is normally handled natively in TensorRT using the `IFullyConnected` layer. However, in this sample, we use a plugin implementation for instructive purposes.

Where Is This Sample Located?

This sample is installed in the `/usr/src/tensorrt/samples/python/fc_plugin_caffe_mnist` directory.

Getting started:

Refer to the `/usr/src/tensorrt/samples/python/fc_plugin_caffe_mnist/README.md` file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

A summary of the `README.md` file is included in this section for your reference, however, you should always refer to the `README.md` within the package for the most recent documentation updates.

Chapter 29.

ADDING A CUSTOM LAYER TO YOUR TENSORFLOW NETWORK IN TENSORRT IN PYTHON

This sample, `uff_custom_plugin`, demonstrates how to use plugins written in C++ with the TensorRT Python bindings and UFF Parser. This sample uses the MNIST dataset.

Where Is This Sample Located?

This sample is installed in the `/usr/src/tensorrt/samples/python/uff_custom_plugin` directory.

Getting Started:

Refer to the `/usr/src/tensorrt/samples/python/uff_custom_plugin/README.md` file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

A summary of the `README.md` file is included in this section for your reference, however, you should always refer to the `README.md` within the package for the most recent documentation updates.

Chapter 30.

OBJECT DETECTION WITH THE ONNX TENSORRT BACKEND IN PYTHON

This sample, `yolov3_onnx`, implements a full ONNX-based pipeline for performing inference with the YOLOv3 network, with an input size of 608x608 pixels, including pre and post-processing.

What Does This Sample Do?

This sample is based on the [YOLOv3-608](#) paper.



This sample is not supported on Ubuntu 14.04 and older. Additionally, the `yolov3_to_onnx.py` script does not support Python 3.

Where Is This Sample Located?

This sample is installed in the `/usr/src/tensorrt/samples/python/yolov3_onnx` directory.

Getting Started:

Refer to the `/usr/src/tensorrt/samples/python/yolov3_onnx/README.md` file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

A summary of the `README.md` file is included in this section for your reference, however, you should always refer to the `README.md` within the package for the most recent documentation updates.

Chapter 31.

OBJECT DETECTION WITH SSD IN PYTHON

This sample, `uff_ssd`, implements a full UFF-based pipeline for performing inference with an SSD (InceptionV2 feature extractor) network.

What Does This Sample Do?

This sample is based on the [SSD: Single Shot MultiBox Detector](#) paper. The SSD network, built on the VGG-16 network, performs the task of object detection and localization in a single forward pass of the network. This approach discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple features with different resolutions to naturally handle objects of various sizes.

This sample is based on the TensorFlow implementation of SSD. For more information, download [ssd_inception_v2_coco](#). Unlike the paper, the TensorFlow SSD network was trained on the InceptionV2 architecture using the MSCOCO dataset which has 91 classes (including the background class). The config details of the network can be found [here](#).

Where Is This Sample Located?

This sample is installed in the `/usr/src/tensorrt/samples/python/uff_ssd` directory.

Getting Started:

Refer to the `/usr/src/tensorrt/samples/python/uff_ssd/README.md` file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

A summary of the `README.md` file is included in this section for your reference, however, you should always refer to the `README.md` within the package for the most recent documentation updates.

Chapter 32.

INT8 CALIBRATION IN PYTHON

This sample, `int8_caffe_mnist`, demonstrates how to create an INT8 calibrator, build and calibrate an engine for INT8 mode, and finally run inference in INT8 mode.

Where Is This Sample Located?

This sample is installed in the `/usr/src/tensorrt/samples/python/int8_caffe_mnist` directory.

Getting Started:

Refer to the `/usr/src/tensorrt/samples/python/int8_caffe_mnist/README.md` file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

A summary of the `README.md` file is included in this section for your reference, however, you should always refer to the `README.md` within the package for the most recent documentation updates.

Chapter 33.

REFITTING AN ENGINE IN PYTHON

This sample, `engine_refit_mnist`, trains an MNIST model in PyTorch, recreates the network in TensorRT with dummy weights, and finally refits the TensorRT engine with weights from the model. Refitting allows us to quickly modify the weights in a TensorRT engine without needing to rebuild.

Where Is This Sample Located?

This sample is installed in the `/usr/src/tensorrt/samples/python/engine_refit_mnist` directory.

Getting Started:

Refer to the `/usr/src/tensorrt/samples/python/engine_refit_mnist/README.md` file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.

A summary of the `README.md` file is included in this section for your reference, however, you should always refer to the `README.md` within the package for the most recent documentation updates.

Notice

THE INFORMATION IN THIS GUIDE AND ALL OTHER INFORMATION CONTAINED IN NVIDIA DOCUMENTATION REFERENCED IN THIS GUIDE IS PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE INFORMATION FOR THE PRODUCT, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the product described in this guide shall be limited in accordance with the NVIDIA terms and conditions of sale for the product.

THE NVIDIA PRODUCT DESCRIBED IN THIS GUIDE IS NOT FAULT TOLERANT AND IS NOT DESIGNED, MANUFACTURED OR INTENDED FOR USE IN CONNECTION WITH THE DESIGN, CONSTRUCTION, MAINTENANCE, AND/OR OPERATION OF ANY SYSTEM WHERE THE USE OR A FAILURE OF SUCH SYSTEM COULD RESULT IN A SITUATION THAT THREATENS THE SAFETY OF HUMAN LIFE OR SEVERE PHYSICAL HARM OR PROPERTY DAMAGE (INCLUDING, FOR EXAMPLE, USE IN CONNECTION WITH ANY NUCLEAR, AVIONICS, LIFE SUPPORT OR OTHER LIFE CRITICAL APPLICATION). NVIDIA EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR SUCH HIGH RISK USES. NVIDIA SHALL NOT BE LIABLE TO CUSTOMER OR ANY THIRD PARTY, IN WHOLE OR IN PART, FOR ANY CLAIMS OR DAMAGES ARISING FROM SUCH HIGH RISK USES.

NVIDIA makes no representation or warranty that the product described in this guide will be suitable for any specified use without further testing or modification. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to ensure the product is suitable and fit for the application planned by customer and to do the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this guide. NVIDIA does not accept any liability related to any default, damage, costs or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this guide, or (ii) customer product designs.

Other than the right for customer to use the information in this guide with the product, no other license, either expressed or implied, is hereby granted by NVIDIA under this guide. Reproduction of information in this guide is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

Trademarks

NVIDIA, the NVIDIA logo, and cuBLAS, CUDA, cuDNN, DALI, DIGITS, DGX, DGX-1, DGX-2, DGX Station, DLProf, Jetson, Kepler, Maxwell, NCCL, Nsight Compute, Nsight Systems, NvCaffe, PerfWorks, Pascal, SDK Manager, Tegra, TensorRT, TensorRT Inference Server, Tesla, TF-TRT, and Volta are trademarks and/or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2019 NVIDIA Corporation. All rights reserved.

www.nvidia.com

