基于 Pytorch 训练并部署 ONNX 模型在 TDA4

Fredy Zhang *EP FAE*

摘要

TI 最新一代的汽车处理器 TDA4VM 集成了高性能计算单元 C7x DSP（Digital Signal Processor）和 Deep-learning Matrix Multiply Accelerator（MMA），TIDL 是 TI 的 Deep Learning 加速库，用于加速 TI 嵌入式设备上的深度神经网络 Deep Neural Networks (DNN)。 上一代产品 TDA2/3 系列处理器，集成了计算单元 DSP（Digital Signal Processor）和 EVE（Embedded Vision/Vector Engine），用于加速深度神经网络。相比于上一代 TDA2/TDA3 系列处理器，最新一代的 TDA4 处理器在算例上得到了大幅提高的同时，在软件方面提供了更好地支持。

基于深度神经网络 (DNN)是一种目前被广泛使用的工具，可以用于图像识别、分类，物体检测，机器翻译等。越来越多的基于 DNN 的机器学习算法被应用于 ADAS 产品中，这些神经网络通常需要大量的计算，这些计算都可以在 TI TDA4 系列处理器中的 C7x DSP 和 MMA 中高效地运行。RTOS SDK 中集成了众多的 Demo，Demo 展示 TIDL 在 TDA4 处理器上对实时的语义分割和 SSD 目标检测的能力。TIDL 当前支持的训练框架有 Tensorflow、Pytorch、Caffe 等，用户可以根据需要选择合适的训练框架进行模型训练。

PyTorch 是一个以 Python 优先的深度学习框架，能够在强大的 GPU 加速基础上实现张量和动态神经网络。Pytorch 是相当简洁且高效的框架，它让用户尽可能地专注于实现自己的想法。当前，Pytorch 广泛应用于模型的训练。

TIDL 涉及到深度学习领域和嵌入式设计领域，同时，TIDL 给大部分算法工程师的使用带来了大量困难。本文旨在通过中文的方式，快速、直接地基于 Pytorch 将训练好的模型导出 ONNX 模型，并在 TDA4 上部署模型、验证模型的正确性。

修改记录

Version	Date	Author	Notes
1.0	June 2022	Fredy Zhang	First release

目录

图

表

1. 基本介绍

1.1. TIDL 简介

TDA4 处理器集成了 TI 最新一代 C7x DSP 和 TI 的 DNN 加速器 (MMA) 用于执行 DNN。 TIDL 可用于德州仪器 (TI) 的各种嵌入式设备，是作为 TI 软件开发套件 (SDK) 的一部分发布的，同时还有其他计算机视觉功能和优化的库，包括 OpenCV。

基于深度神经网络 (DNN) 的机器学习算法用于许多行业，例如机器人、工业和汽车。越来越多的基于 DNN 的机器学习算法被应用于 ADAS 产品中，这些神经网络通常需要大量的计算，这些计算都可以在 TI TDA4 系列处理器中的 C7x DSP 和 MMA 中高效地运行。RTOS SDK 中集成了众多的 Demo 展示 TIDL 在 TDA4 处理器上对实时的语义分割和 SSD 目标检测的能力。TIDL 当前支持的训练框架有 Tensorflow、Pytorch、Caffe 等，用户可以根据需要选择合适的训练框架进行模型训练。

如图 1 所示，是 TIDL 的软件框架。在 TIDL 上，深度学习网络应用开发主要分为三个大的步骤（以 TI Jacinto7 TDA4VM 处理器为例）：

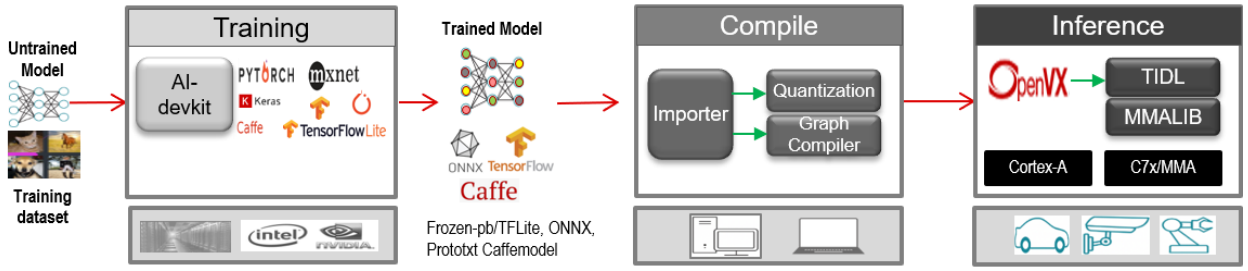

图 **1**. TIDL SW Framework

1. 基于 Tensorflow、Pytorch、Caffe 等训练框架，训练模型：选择一个训练框架，然后定义模型，最后使用训练的数据集训练出满足需求的模型。
2. 基于 TI Jacinto7 TDA4VM 处理器导入模型： 训练好的模型，需要使用 TIDL Importer 工具导入成可在 TIDL 上运行的模型。导入的主要目的是对输入的模型进行量化、优化并保存为 TIDL 能够识别的网络模型和网络参数文件。
3. 基于 TI Jacinto7 SDK 验证模型，并在应用里面部署模型：
    a. PC 上验证并部署
        i. 在 PC 上使用 TIDL 推理引擎进行模型测试。
        ii. 在 PC 上使用 OpenVX 框架开发程序，在应用上进行验证。
    b. EVM 上验证并部署
        i. 在 EVM 上使用 TIDL 推理引擎进行模型测试。
        ii. 在 EVM 上使用 OpenVX 框架开发程序，在应用上进行验证。

1.2. Pytorch 简介

2017 年 1 月，由 Facebook 人工智能研究院（FAIR）基于 Torch 推出了 PyTorch。它是一个基于 Python 的可续计算包，提供两个高级功能：1、具有强大的 GPU 加速的张量计算（如 NumPy）。2、包含自动求导系统的深度神经网络。PyTorch 的前身是 Torch，其底层和 Torch 框架一样，但是使用 Python 重新写了很多内容，不仅更加灵活，支持动态图，而且提供了 Python 接口。它是由 Torch7 团队开发，是一个以 Python 优先的深度学习框架，不仅能够实现强大的 GPU 加速，同时还支持动态神经网络。

PyTorch 是相当简洁且高效快速的框架，设计追求最少的封装并符合人类思维，它让用户尽可能地专注于实现自己的想法。因此，受广大用户的喜爱，并广泛应用于视觉感知等领域。

本问将详细介绍基于 Pytorch 导出 ONNX 模型，并基于 ONNX Runtime 验证模型的正确性。最后，介绍 TDA4 上模型 ONNX 模型部署。对于 Pytorch 的模型训练，Pytorch 社区和官方有很好的支持，本文将不做介绍。

2. Pytorch 模型导出

MobileNetV2 模型广泛应用于分类任务。本章节将以 MobileNetV2 为例介绍 PyTorch 导出 ONNX 模型和模型的正确性验证。

2.1. MobileNetV2 模型简介

MobileNetV2 是由 google 团队在 2018 年提出的，相比于 MobileNetV1 而言准确率更高，模型更小。其原始论文为 MobileNetV2: Inverted Residuals and Linear Bottlenecks。MobileNet v2 的基本单元为 Bottleneck residual block, 其模型如下图：

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

图 **2**. MobileNetV2 模型

2.2. Pytorch 导出 ONNX 模型

ONNX(Open Neural Network Exchange)，开放神经网络交换，用于在各种深度学习训练和推理框架转换的一个中间表示格式。ONNX 定义了一组和环境，平台均无关的标准格式，来增强各种 AI 模型的可交互性，开放性较强。

TIDL 对 ONNX 模型有很好地支持，因此，推荐使用使用 ONNX 模型。下面介绍 Pytorch 如何导出 ONNX 模型。

Pytorch 模型保存的格式为.pth 后缀模型，保存方式如下：

```
torch.save(mobilenetv2,'./models/mobilenetv2_test.pth')
```

Pytorch 模型导出使用自带的接口 torch.onnx.export

```
# Export the model
# torch.onnx.export(mobilenetv2,               # model being run
#                   x,                         # model input (or a tuple for
 multiple inputs)
#                   "./models/mobilenetv2.onnx",   # where to save the model
 (can be a file or file-like object)
#                   export_params=True,        # store the trained parameter
 weights inside the model file
#                   opset_version=10,          # the ONNX version to export
the model to
#                   do_constant_folding=True,  # whether to execute constant
 folding for optimization
#                   input_names = ['input'],   # the model's input names
#                   output_names = ['output'], # the model's output names
#                   dynamic_axes={'input' : {0 : 'batch_size'},    # variabl
e length axes
#                                 'output' : {0 : 'batch_size'}})
```

下面用一个例子来介绍，我们直接使用 Model Hub 下载训练好的 MobileNetV2 模型，并导出 MobileNetV2 ONNX 模型，具体的实现代码如下：

```
from torchvision import models
import torchvision
import torch
# Download the model from hub
mobilenetv2 = torch.hub.load('pytorch/vision:v0.10.0', 'mobilenet_v2', pretr
ained=True)
print(mobilenetv2)
# Input to the model
batch_size =1
new_mobilenetv2.eval()
x = torch.randn(batch_size, 3, 224, 224, requires_grad=True)
torch_out = mobilenetv2(x)
```

```
# Export the model
torch.onnx.export(mobilenetv2, x, "./models/mobilenetv2.onnx", opset_version
 = 11)
```

2.3. 验证模型正确性

在某些情况下，为了便于确定问题，ONNX 导出的模型，如果要跟 Pytorch 本身的推理进行正确性比较，可以参考如下的代码。从而确定 ONNX 模型的正确性。如果结果正确，窗口会打印“Exported model has been tested with ONNXRuntime, and the result looks good!”。

```
from torchvision import models
import torchvision
import torch
import onnx

# Download the model from hub
mobilenetv2 = torch.hub.load('pytorch/vision:v0.10.0', 'mobilenet_v2', pretr
ained=True)
# Input to the model
batch_size =1
new_mobilenetv2.eval()
x = torch.randn(batch_size, 3, 224, 224, requires_grad=True)
torch_out = mobilenetv2(x)
# Export the model
torch.onnx.export(mobilenetv2, x, "./models/mobilenetv2.onnx", opset_version
 = 11)

# ONNX Runtime
ort_session = onnxruntime.InferenceSession("./models/mobilenetv2.onnx")

def to_numpy(tensor):
    return tensor.detach().cpu().numpy() if tensor.requires_grad else tenso
r.cpu().numpy()

# compute ONNX Runtime output prediction
ort_inputs = {ort_session.get_inputs()[0].name: to_numpy(x)}
# print(ort_inputs)
ort_outs = ort_session.run(None, ort_inputs)

# compare ONNX Runtime and PyTorch results
np.testing.assert_allclose(to_numpy(torch_out), ort_outs[0],rtol=1e-03, atol
=1e-05)
```

```
print("Exported model has been tested with ONNXRuntime, and the result looks
 good!")
```

2.4. 模型推理

从网站上下载如下狗（Samoyed）的图片，分别用 Pytorch 和 ONNX 进行推理。 下载代码如下，图片如图 3 所示。

```
# Download an example image from the pytorch website
import urllib
url, filename = ("https://github.com/pytorch/hub/raw/master/images/dog.jpg",
 "dog.jpg")
try: urllib.URLopener().retrieve(url, filename)
except: urllib.request.urlretrieve(url, filename)
```

图 **3**. Samoyed

2.4.1. Pytorch 模型推理

使用图 3 的图片，利用如下代码我们进行 Pytorch 推理：

```
from PIL import Image
from torchvision import transforms
from torchvision import models
import torchvision
import torch
import onnx

input_image = Image.open(filename)
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
```

```
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.22
5]),
])

input_tensor = preprocess(input_image)
input_batch = input_tensor.unsqueeze(0) # create a mini-batch as expected by
 the model

mobilenetv2 = torch.hub.load('pytorch/vision:v0.10.0', 'mobilenet_v2', pretr
ained=True)

with torch.no_grad():
    output = mobilenetv2(input_batch)
probabilities = torch.nn.functional.softmax(output[0], dim=0)
print(probabilities)
```

输出结果如下，我们看到识别结果为：Samoyed，识别结果正确。

```
Samoyed 0.8303039073944092
Pomeranian 0.06988810002803802
keeshond 0.012964135967195034
collie 0.010797751136124134
Great Pyrenees 0.009886731393635273
```

2.4.2. ONNX 模型推理

经过 Pytorch 导出的 ONNX 模型，可以使用图 3 的图片，利用如下代码进行 PC 推理，进行结果验证：

```
from PIL import Image
import numpy
from PIL import Image
from torchvision import transforms
import torchvision
import torch
import onnx
import onnxruntime

ort_session = onnxruntime.InferenceSession("./models/mobilenetv2.onnx") ;

input_image = Image.open(filename)
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.22
5]),
```

```
])

input_tensor = preprocess(input_image)
input_batch = input_tensor.unsqueeze(0) # create a mini-batch as expected by
 the model

ort_inputs = {ort_session.get_inputs()[0].name: to_numpy(input_batch)}
ort_outs = ort_session.run(None, ort_inputs)
img_out_y = ort_outs[0]
print(img_out_y.ndim)
print(img_out_y.shape)

print("Label predict: ", img_out_y.argmax())
img_out_y=img_out_y.ravel()
img_out=torch.tensor(img_out_y)

# Read the categories
with open("imagenet_classes.txt", "r") as f:
    categories = [s.strip() for s in f.readlines()]
# Show top categories per image
top5_prob, top5_catid = torch.topk(img_out, 5)
for i in range(top5_prob.size(0)):
    print(categories[top5_catid[i]], top5_prob[i].item())
```

输出结果如下，我们看到识别结果为：Samoyed，识别结果正确。

```
Samoyed 14.355224609375
Pomeranian 11.880317687988281
keeshond 10.195608139038086
collie 10.012767791748047
Great Pyrenees 9.924626350402832
```

3. ONNX 模型 TIDL 部署

ONNX 模型并验证后，可以在 TIDL 部署。ONNX 模型，首先要经过 import 转化成能够被 TIDL 所识别的模型。如表 1 所示，TIDL 也提供了 ONNX Model Zoo，可以方便地利用模型库的模型进行部署和验证。本节将介绍如何使用上一节使用的 MobileNetV2 快速进行模型部署。ONNX 模型与 Tensorflow 的流程类似，请参考快速部署 Tensorflow 和 TFLITE 模型在 Jacinto7 Soc，因此，这里不做具体介绍，对于有差异的地方后文将给出说明，细节参考快速部署 Tensorflow 和 TFLITE 模型在 Jacinto7 Soc。

表 1. ONNX Model Zoo

Num	Network Architecture	Source
1	MobileNet-1.0 V2	Link

2	SqueezeNet 1.1	Link
3	Resnet 18 v1	Link
4	Resnet 18 v2	Link
5	ShuffleNet v1	Link
6	VGG 16	Link
7	Yolo V3	Link
8	Resnet 34 v1	Link
9	RegNetx-200mf	Link
10	RegNetx-400mf	Link
11	RegNetx-800mf	Link

3.1. ONNX 模型导入

模型导入使用 tidl_model_import.out 工具，该工具导入输入的模型（ONNX 模型），输出能够为 TIDL 所使用的网络模型和参数文件（TIDL Network File 和 TIDL IO Info File）,输入输出文件如下：

```
TF Model (Proto) File  : ../../test/testvecs/models/public/onnx/mobilenetv2/
mobilenetv2.onnx
TIDL Network File      : ../../test/testvecs/config/tidl_models/onnx/tidl_ne
t_mobilenetv2.bin
TIDL IO Info File      : ../../test/testvecs/config/tidl_models/onnx/tidl_io
_mobilenetv2
```

模型的导入需要模型导入配置文件，tidl_import_mobilenetv2.txt，需要将其放在 ${TIDL_INSTALL_PATH}/ti_dl/test/testvecs/config/import/public/onnx/路径下。导入的配置文件里面指定了模型配置参数，如果对参数有不清楚的地方参考链接 TIDL-RT Import Configuration Parameters：

```
# modelType : 2 - ONNX (.onnx files)
modelType         = 2
……
inputNetFile       = "../../test/testvecs/models/public/onnx/mobilenetv2/mobilenetv
2.onnx"
……
```

在 Ubuntu 命令行执行下面命令导入模型：

```
user@ubuntu-pc$ cd /home/fredy/startJacinto/sdks/ti-processor-sdk-rtos-j721e-evm-08_
01_00_11/tidl_j7_08_01_00_05/ti_dl/utils/tidlModelImport && \
./out/tidl_model_import.out /home/fredy/startJacinto/sdks/ti-processor-sdk-rtos-j721
e-evm-08_01_00_11/ tidl_j7_08_01_00_05/ti_dl/test/testvecs/config/import/public/onnx
/tidl_import_mobilenetv2.txt
```

Ubuntu 命令行执行上述命令输出的 log 中，结尾输出 `ALL MODEL CHECK PASSED` 说明模型导入成功。

3.2. 模型 PC 推理

这一小节，将使用上一章节生成的模型文件（TIDL Network File 和 TIDL Network File ）验证模型的正确性，模型推理的过程如图 4 所示。

TIDL Network File : ../../test/testvecs/config/tidl_models/onnx/tidl_net_mobilenetv2.bin
TIDL IO Info File : ../../test/testvecs/config/tidl_models/onnx/tidl_io_mobilenetv2

图 **4**. 模型推理

部署的模型 MobileNet V2 用来进行图像分类，其训练数据集来自于 ImageNet, 其对应的标签可以在这里查询，如表 2 所示，从左到右是输入图像、推理结果、结果对应的标签：

表 2. Input Image

Input Image	Inference Result	Lable
testvecs/input/airshow.jpg	896	'warplane, military plane'
testvecs/input/ti_lindau_I00000.jpg	558	'flagpole, flagstaff'
testvecs/input/ti_lindau_000020.jpg	443	'bell cote, bell cot'
testvecs/input/0000000271.png	499	'cinema, movie theater, movie theatre, movie house, picture palace'

3.2.1. PC 模型推理

经过 TIDL tidl_model_import.out 工具导入 tensorflow 模型，我们可以快速利用 Ubuntu 的工具 PC_dsp_test_dl_algo.out 进行推理验证结果。 PC_dsp_test_dl_algo.out 需要配置文件 testvecs/config/infer/public/onnx/tidl_infer_mobilenetv2.txt（相关参数的解释，请参考这里）

3.2.2. PC 模型部署

TDA4x 使用了 OpenVX 框架，可支持 PC 仿真。因此，应用可以基于 PC 验证。如图 9，是基于 OPENVX 的图像分类示例的框图。

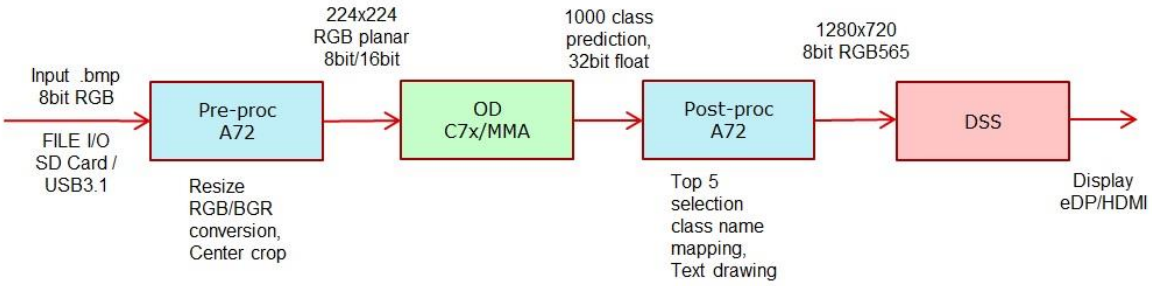

图 **5**. Image Classification Application

3.3. 模型 EVM 推理

模型在 PC 验证后，最终可以部署在 EVM 上进行性能和结果测试。如果觉得 PC 验证会浪费时间，可以省略 PC 验证的步骤，完成模型的导入后，就可以直接在 EVM 上验证。

经过 TIDL tidl_model_import.out 工具导入的 ONNX 模型，在 EVM 上，使用 TI_DEVICE_a72_test_dl_algo_host_rt.out 工具进行推理测试。该工具不仅可以测试使用模型的正确性，还可以测试真实的帧率。 经 TI_DEVICE_a72_test_dl_algo_host_rt.out 处理后，输出图像存储在 /opt/tidl_test/testvecs/output,可将数据导入到 PC 确认其结果。

基于 EVM 进行推理确认其正确性后，模型可以部署到应用。SDK 中提供了非常多的示例，拿图 5 所示，是基于 OpenVX 的 Image Classification Application 示例。

4. FAQ

4.1. SDK 中 TIDL 支持哪些算子？

TIDL 支持的算子请参考使用版本的 UserGuide，参考链接。

4.2. SDK 中 ONNX 支持和验证的模型有哪些？

TIDL 提供了支持的模型库，用户可以参考模型库中的模型快速进行模型进行验证，并参考模型进行新的模型设计。模型库参考链接。

4.3. 如果遇到 performance 问题，如何进行调试？

Performance 的问题请参考链接的建议进行调试。

4.4. TIDL Importer 工具导入 ONNX 模型需要注意哪些方面？

ONNX 模型可以直接导入 TIDL，TIDL Importer 工具输入配置文件路径在：TIDL_PATH /ti_dl/test/testvecs/config/import/public/tensorflow/tidl_import_xxx.txt。对于 import 配置文件注意检查如下内容：

1. 导入的 modelType： ONNX 模型配置为 2 (.onnx files).
2. inputNetfile/outputNetFile: 检查输入/输出模型文件的路径。
3. inData： 输入数据的配置，通常自己的模型，需要调整输入图片。
4. inWidth/inHeight/resizeWidth/resezeHeight ： 配置图片输入的 size 及调整后的 size。
5. inNumChannels： 输入图片通道数。

4.5. TIDL 如何打印 log 信息？

不论是 Import 模型的时候，还是 inference 的时候，难免会遇到问题，当遇到问题的时候，怎么样才能输出更多的调试信息呢？ TIDL 提供了两个标志变量 writeTraceLevel 和 debugTraceLevel 用来获取更多的信息。

debugTraceLevel 可以打印更多的输出信息，便于追踪 import 和 inference 的过程。默认配置是 0，可支持 1、2 配置。数字越大，表明输出的信息越多。

writeTraceLevel 可以输出每一层的信息到文件，便于模型逐层比较。默认配置是 0，没有输出，可支持 1、2、3 配置：1- Fixed Point , 2- Padded Fixed Point, 3 - Floating point。

5. 参考

1. https://software-dl.ti.com/jacinto7/esd/processor-sdk-rtos-jacinto7/08_01_00_11/exports/docs/tidl_j7_08_01_00_05/ti_dl/docs/user_guide_html/usergroup0.html
2. https://pytorch.org/tutorials/
3. https://pytorch.org/docs/stable/index.html
4. https://openaccess.thecvf.com/content_cvpr_2018/papers/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.pdf
5. https://www.ti.com.cn/cn/lit/an/zhcab78/zhcab78.pdf

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265