恩智浦半导体
用户指南

文档编号：IMXMLUG
第 LF5.10.72_2.2.0 版，2021 年 12 月 17 日

i.MX 机器学习用户指南

目录

第 1 章 软件栈简介 5

第 2 章 eIQ Inference Runtime 概述 7

第 3 章 TensorFlow Lite 9
3.1 TensorFlow Lite 软件栈 9
3.2 计算后端和代理 10
3.2.1 内置内核 10
3.2.2 XNNPACK Delegate 11
3.2.3 NNAPI Delegate 11
3.2.4 VX Delegate 11
3.3 交付包 11
3.4 构建详情 12
3.5 应用程序开发 12
3.5.1 创建使用 TensorFlow Lite 的 CMake 项目 13
3.5.2 使用 Yocto SDK 预编译库 13
3.6 运行图像分类示例 14
3.7 运行基准测试应用程序 16
3.8 使用 TensorFlow Lite 转换器进行训练后量化 18

第 4 章 Arm 计算库 21
4.1 使用随机权重和输入运行 DNN 21
4.1.1 使用图形 API 运行 AlexNet 21

第 5 章 Arm NN 23
5.1 Arm NN 软件栈 23
5.2 计算后端 24
5.3 运行 Arm NN 测试 25
5.3.1 TensorFlow Lite 测试 25
5.3.2 ONNX 测试 26
5.4 在自定义 C/C++应用程序中使用 Arm NN 27
5.5 对接 Arm NN（PyArmNN）的 Python 接口 28
5.5.1 开始 28
5.5.2 运行示例 29
5.6 面向 TensorFlow Lite 的 Arm NN Delegate 29
5.6.1 ARM NN Delegate C++项目集成 29

第 6 章 ONNX Runtime 32
6.1 ONNX Runtime 软件栈 32
6.2 执行提供程序 33
6.2.1 ONNX 模型测试 34
6.2.2 C API. 34
6.2.2.1 启用执行提供程序 34
6.2.3 ONNX 性能测试 35

第 7 章 PyTorch 36
7.1 运行图像分类示例 36
7.2 构建和安装 wheel 包....36
7.2.1 如何构建....37
7.2.2 如何安装....37

第 8 章 OpenCV 机器学习演示 38
8.1 下载 OpenCV 演示....38
8.2 OpenCV DNN 演示 38
8.2.1 图像分类演示 39
8.2.2 YOLO 目标检测示例 40
8.2.3 图像分割演示 41
8.2.4 图像着色演示 42
8.2.5 人体姿势检测演示 43
8.2.6 目标检测示例 43
8.2.7 CNN 图像分类示例....44
8.2.8 文本检测....45
8.3 OpenCV 经典机器学习演示 46
8.3.1 SVM 简介 46
8.3.2 非线性可分离数据的 SVM....47
8.3.3 主成分分析（PCA）介绍....48
8.3.4 逻辑回归....49

第 9 章 DeepViewRT 51
9.1 DeepViewRT 软件栈....51
9.2 交付包 52
9.3 示例应用程序....52
9.3.1 图像标记应用 53
9.3.2 目标检测应用程序 54
9.3.3 Labelcam-gst 示例应用程序 54
9.3.4 Ssdcam-gst 示例应用 55
9.4 ModelRunner 55
9.4.1 DeepViewRT 55
9.4.2 OpenVX....56
9.4.3 TensorFlow Lite 56
9.4.4 Arm NN....56
9.4.5 ONNX Runtime 56

第 10 章 TVM 58
10.1 TVM 软件工作流程 58
10.2 开始....58
10.2.1 通过 RPC 验证运行示例....58
10.2.2 在设备上单独运行示例....59
10.3 如何在主机上构建 TVM 堆栈....59
10.4 支持的模型 60

第 11 章 在硬件加速器上执行 NN 62
11.1 硬件加速器介绍....62
11.2 硬件加速器上的分析....62
11.3 硬件加速器预热时间....63
11.4 GPU 和 NPU 之间的切换....64

第 12 章 eIQ 演示 65
12.1 GStreamer......65
12.1.1 Gstreamer 软件工作流程 65
12.1.2 开始......66
12.1.2.1 使用视频流进行目标检测 66
12.1.2.2 使用摄像头流进行目标检测......66
12.1.2.3 使用视频流运行姿势估计 66
12.1.2.4 使用摄像头流运行姿势估计......66
12.1.2.5 Pipeline 演示命令......67
12.2 NNStreamer 67
12.2.1 目标检测 pipeline 示例......69
12.2.2 Pipeline 分析 70
12.2.2.1 启用 NNShark 分析 71
12.2.2.2 向 NNShark 添加功率测量......72
12.2.2.3 已知问题和限制 72
12.3 AWS 端到端 SageMaker 演示 73
12.3.1 SageMaker 演示工作流程 73
12.3.2 开始......74
12.3.2.1 构建 BSP 镜像......75
12.3.2.2 在器件上运行演示脚本......75
12.3.2.3 查看推理结果 76
12.3.2.4 清理云环境......76
12.3.3 其他资源......76

第 13 章修订历史 78

附录 A 版本说明......79

附录 B 使用的变量列表 82

附录 C 神经网络 API 参考......83

附录 D GPU 支持 OVXLIB 操作......89

附录 E NPU 支持 OVXLIB 操作 105

第 1 章
软件栈简介

恩智浦 eIQ 机器学习软件开发环境（以下简称“恩智浦 eIQ”）为针对恩智浦微控制器和应用处理器的机器学习应用提供了多个库和开发工具。恩智浦 eIQ 包含在 *meta-imx/meta-ml* Yocto layer 中。如需了解更多信息，请参见《*i.MX Yocto Project 用户指南*》（IMXLXYOCTOUG）。

恩智浦 eIQ 软件栈目前支持以下 6 个推理引擎：Arm NN、TensorFlow Lite、ONNX Runtime、PyTorch、OpenCV 和 DeepView™RT。下图展示了不同计算单元支持的 eIQ 推理引擎。

NXP eIQ Inference Engines & Libraries	eIQ Inference Engine Deployment													
	PyTorch	arm NN	ONNX RUNTIME	TensorFlow Lite	OpenCV	DeepViewRT	arm NN	ONNX RUNTIME	TensorFlow Lite	DeepViewRT	arm NN	ONNX RUNTIME	TensorFlow Lite	DeepViewRT
Compute Engines	Cortex-A						GPU				NPU			
i.MX 8M Plus	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
i.MX 8QuadMax	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	NA	NA	NA	NA
i.MX 8QuadXPlus	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	NA	NA	NA	NA
i.MX 8M Quad, Nano	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	NA	NA	NA	NA
i.MX 8M Mini, 8ULP	✓	✓	✓	✓	✓	✓	NA	NA	NA	NA	NA	NA	NA	NA

✓ Supported　　NA (Not Applicable)

图 1. 恩智浦 eIQ 支持的计算引擎与推理引擎

恩智浦 eIQ 推理引擎支持 Cortex-A 内核多线程执行。此外，Arm NN、ONNX Runtime、TensorFlow Lite 和 DeepViewRT 还支持通过 Neural Network Runtime（NNRT）在 GPU 或 NPU 上进行加速。另请参见《eIQ Inference Runtime 概述》。

通常，恩智浦 eIQ 支持以下关键应用领域：

- 视觉
  - 多摄像头观察
  - 主动物体识别
  - 手势控制
- 语音
  - 语音处理
  - 家庭娱乐

- 声音
  - 智能感知和控制
  - 目视检查
  - 声音监测

第 2 章
eIQ Inference Runtime 概述

本章概述了与恩智浦神经网络加速器 IP（GPU 或 NPU）一起使用的恩智浦 eIQ 软件栈。下图展示了每个元件之间的数据流。下图包含两个关键部分：

- Neural Network Runtime (NNRT)，它是一种连接各种推理框架和 NN 加速器驱动程序的中间件。
- TIM-VX，它是一个软件集成模块，用于在支持 OpenVX 的机器学习加速器上加快部署神经网络。

ModelRunner for DeepViewRT 是一个服务器应用程序，能够通过 HTTP REST API、Python API 或 UNIX RPC 服务接收请求，并将这些请求直接委托给不同的推理引擎或 NN 加速器驱动程序。如需了解更多详细信息，请参见《ModelRunner》。

NNRT 为 Android NN HAL、Arm NN、ONNX 和 TensorFlow Lite 提供后端，允许快速部署应用程序。NNRT 还支持一个面向应用程序的框架，用于 i.MX8 处理器。Android NN、TensorFlow Lite 和 Arm NN 等应用框架可直接通过 NNRT 内置的后端插件实现加速。还可以实施额外的后端，扩展对其他框架的支持。

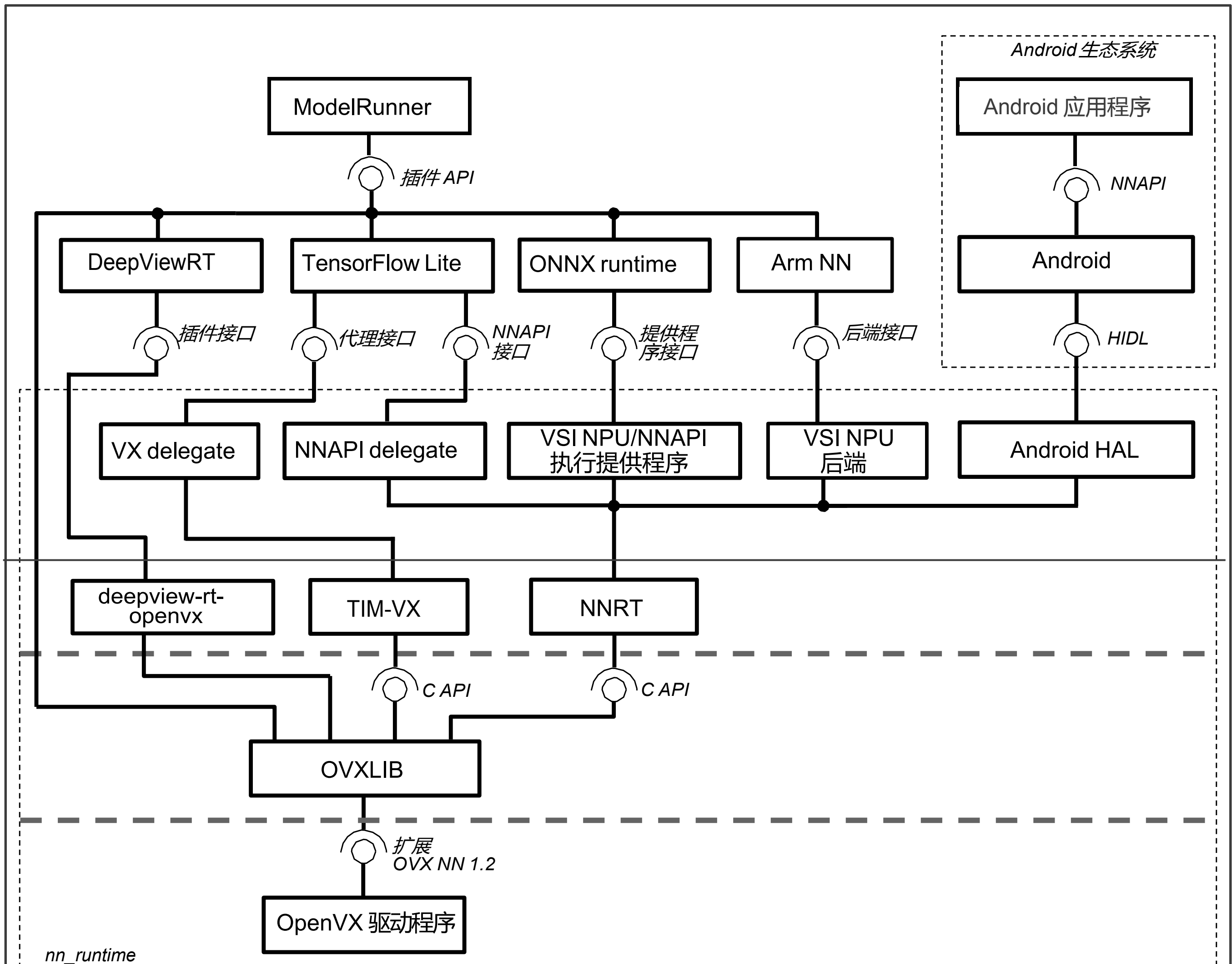

图 2. eIQ 推理软件架构

NNRT 将自己注册为计算后端，可支持不同的机器学习框架。由于每个框架都定义了不同的后端 API，NNRT 根据每个框架设计了不同的轻量级后端层：

- 对于 Android NN，NNRT 遵循 Android HIDL 定义。它兼容 v1.2 HAL 接口
- 对于 TensorFlow Lite，NNRT 支持 NNAPI Delegate。它支持《Android NNAPI v1.2》中的大部分操作
- 对于 Arm NN，NNRT 将自己注册为计算后端
- 对于 ONNX Runtime，NNRT 将自己注册为执行提供程序

这样，NNRT 弥合了应用程序框架的差异，并为驱动程序栈提供了通用运行时接口。同时，NNRT 还充当异构计算平台，用于在 i.MX8 计算器件（如 NPU、GPU 和 CPU）上高效地分配工作负载。

注意

OpenCV 和 PyTorch 推理引擎目前不支持在恩智浦 NN 加速器上运行。因此，上述 NXP-NN 架构框图中未包含这两个框架。

第 3 章
TensorFlow Lite

TensorFlow Lite 是一个开源软件库，专注于在移动和嵌入式设备上运行机器学习模型（请访问 http://www.tensorflow.org/lite）。它能够以低延迟和较小的二进制大小实现在设备上进行机器学习推理。TensorFlow Lite 还支持在各种 i.MX 8 平台（在恩智浦 eIQ 中）上使用 VX Delegate 或 Android OS Neural Networks API（NNAPI）进行硬件加速。

针对 Yocto Linux 的 TensorFlow Lite 发布包位于 lf-5.10.72_2.2.0 分支。该存储库是主线（https://github.com/tensorflow/tensorflow）的一个分支，并针对恩智浦 i.MX8 平台进行了优化。

特性：

- 在 Cortex-A 内核上使用 Arm Neon SIMD 指令加速多线程计算
- 使用 GPU/NPU 硬件加速的并行计算（在 shader 或卷积单元上）
- C++ 和 Python API（Python 版本 3）
- 支持按张量和按通道量化模型

3.1 TensorFlow Lite 软件栈

TensorFlow Lite 软件栈如下图所示。TensorFlow Lite 支持在以下硬件单元上进行计算：

- CPU Arm Cortex-A 内核
- 使用 Android NNAPI 驱动程序或 VX Delegate 的 GPU/NPU 硬件加速器

如需了解在不同硬件平台上支持 GPU/NPU 硬件加速器计算的详细信息，请参见“软件栈简介”章节。

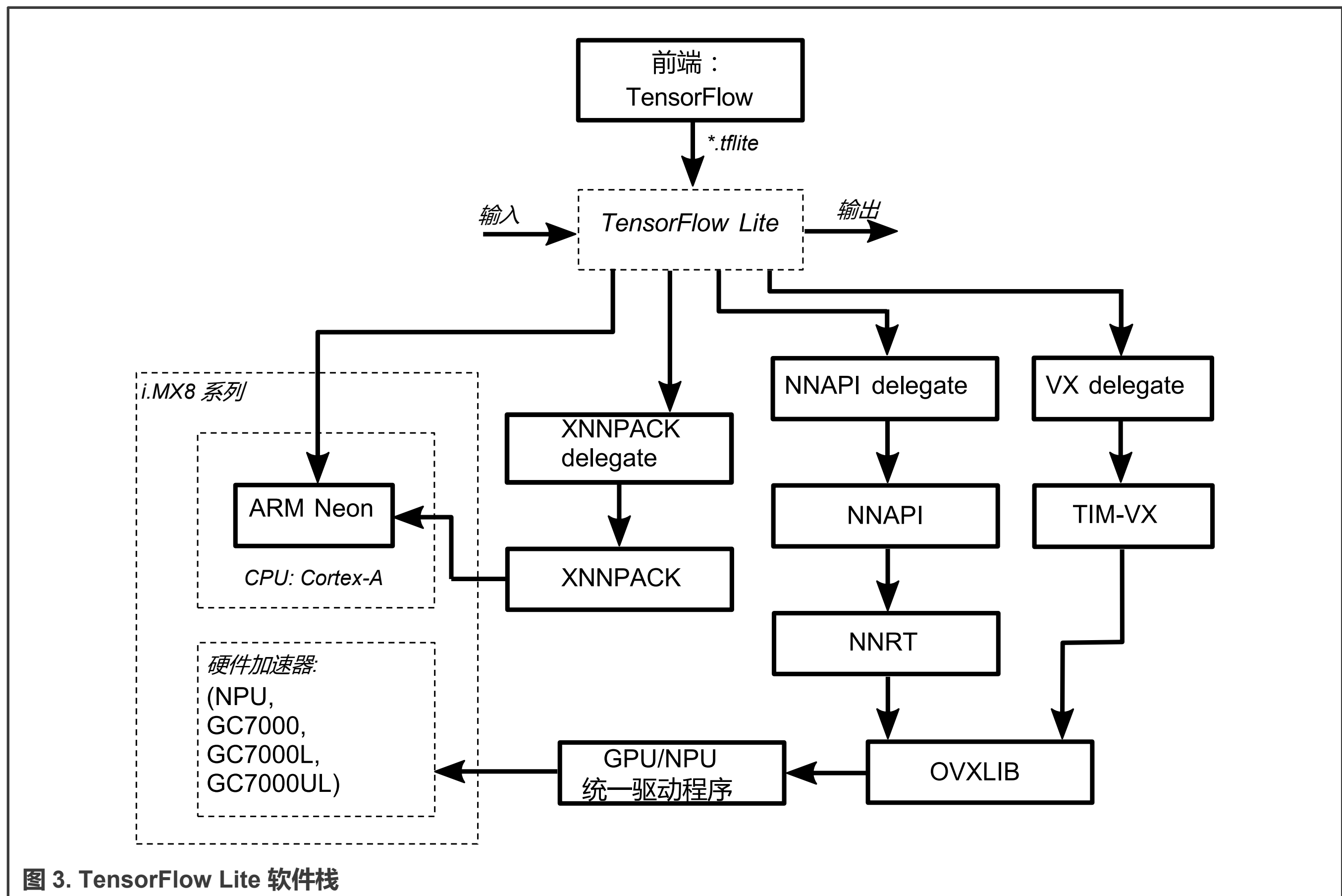

图 3. TensorFlow Lite 软件栈

注意

由于 GPU/NPU 驱动程序初始化计算图需要较长的时间，使用 NNAPI 或 VX Delegate 首次执行模型推理将花费更长的时间。图形初始化之后的迭代将执行得更快。请注意，计算图展示了操作及其依赖关系，以执行模型指定的计算。计算图是在模型解析阶段构建的。

NNAPI 和 VX Delegate 实施使用 OpenVX™ 库在 GPU/NPU 硬件加速器上执行计算图。因此，所选设备必须支持 OpenVX 库才能使用加速。如需了解 OpenVX 库可用性的更多详细信息，请参见《*i.MX 图形用户指南*》（IMXGRAPHICUG）。

GPU/NPU 硬件加速器驱动程序支持按张量和按通道量化模型。GPU/NPU 硬件加速器针对按张量量化模型进行了优化。如使用按通道量化模型，性能可能会更低。实际影响取决于所使用的模型。

3.2 计算后端和代理

TensorFlow Lite 提供执行各种计算单元的计算操作的选项。我们将这些选项称为推理后端。

3.2.1 内置内核

默认推理后端是带有 TensorFlow Lite 的参考内核的 CPU。内置内核完全支持 TensorFlow Lite 算子集。

内置内核是在启用 RUY 矩阵乘法库的情况下构建的，这提高了内核在浮点和量化操作方面的性能。

3.2.2 XNNPACK Delegate

XNNPACK 库是一个高度优化的浮点神经网络推理算子库，适用于 ARM、WebAssembly 和 x86 平台。XNNPACK 库可通过 TensorFlow Lite 中的 XNNPACK Delegate 获得。XNNPACK Delegate 计算在 CPU 上执行。

它优化实施了浮点算子的 TensorFlow Lite 算子集的子集。一般来说，它为浮点运算符提供了比内置内核更好的性能。

注意

自 TensorFlow Lite 2.6.0 以来，浮点模型默认通过 XNNPACK Delegate 执行。

3.2.3 NNAPI Delegate

NNAPI Delegate 实现了片上硬件加速器上的加速推理。该代理（delegate）基于 Android 的神经网络 API（NNAPI）规范。如需获取完整的规范，请访问：https://developer.android.com/ndk/reference/ group/neural-networks。

TensorFlow Lite 库使用 GPU/NPU 驱动程序中的 Android NNAPI 实施，使用 GPU/NPU 硬件加速器运行推理。实施的 NNAPI 版本是 1.2，与 TensorFlow Lite 的功能集相比，它在支持的 tensor 数据类型和运算方面有一些限制。因此，某些模型可在不启用加速的情况下正常运行，但在使用 NNAPI 时可能会失败。如需了解支持功能的完整清单，请参见 NNAPI 文档的“NN HAL 版本”章节。

NNAPI 规范自带了自己的算子集，其中包括 TensorFlow Lite 算子集中的大多数但并非所有算子。此外，NNAPI 不支持 TensorFlow Lite 算子的所有变体。这适用于硬件加速器算子支持，其中一些算子受加速器支持，但不属于 NNAPI 规范。因此，即使硬件加速器支持特定层，这些层的执行也会回到 CPU 执行。

对于 NNAPI Delegate 拒绝的模型中的所有算子，TensorFlow Lite Runtime 会打印一条警告消息，说明该算子被代理拒绝的原因：

```
WARNING: Operator ARG_MAX (v1) refused by NNAPI delegate: NNAPI only supports int32 output.
```

这些信息可用于优化模型以获得更好的性能。

注意

用于 Linux 平台的 NNAPI Delegate 未来将被弃用。Python API 中不支持 NNAPI Delegate。

3.2.4 VX Delegate

VX Delegate 是 i.MX 8 Linux 平台上 NNAPI Delegate 的后续版本。它可以在内部硬件加速器上加速推理。VX Delegate 直接使用硬件加速器驱动程序（带扩展的 OpenVX）来充分利用加速器功能。与 NNAPI Delegate 相比，它可以更好地与内部硬件加速器功能保持一致。

VX Delegate 可用作外部代理[1]。相应的库位于`/usr/lib/libvx_delegate.so` 中。

C++ 和 Python API 都支持 VX Delegate。如需了解如何使用 VX Delegate（或任何外部代理），请参见 C++中的 external_delegate_provider 实施和/或 Python 的 label_image.py。

3.3 交付包

TensorFlow Lite 可使用 Yocto Project 配方获得。

TensorFlow Lite 交付包中包含以下内容：

[1] 外部代理是一种特殊的 Tensorflow Lite 代理，只需加载一个封装了实际 Tensorflow Lite 代理实施的动态库便可进行初始化

- TensorFlow Lite 共享库
- TensorFlow Lite 头文件
- TensorFlow Lite 的 Python 模块
- C++ (label_image) 和 Python (label_image.py) 的图像分类示例应用程序
- TensorFlow Lite 基准测试应用程序（benchmark_model）
- TensorFlow Lite 评估工具（coco_object_detection_run_eval、imagenet_image_classification_run_eval、inference_diff_run_eval），详见 TensorFlow Lite Delegates。

对于应用程序开发，SDK 中提供了 TensorFlow Lite 共享库和头文件。如需了解更多详细信息，请参见“应用程序开发”章节。

TensorFlow Lite 2.6.0 交付包中提供以下代理：

- XNNPACK Delegate
- NNAPI Delegate
- VX Delegate

3.4 构建详情

TensorFlow Lite 使用 CMake 构建系统进行编译。软件包构建注意事项如下所示：

- 启用 RUY 矩阵乘法库（`TFLITE_ENABLE_RUY=On`）。与使用 Eigen 和 GEMLOWP 构建的内核相比，RUY 矩阵乘法库提供了更好的性能。
- 支持 XNNPACK Delegate（`TFLITE_ENABLE_XNNPACK=On`）
- 支持 NNAPI Delegate[2]（`TFLITE_ENABLE_NNAPI=On`），包括被拒绝操作的警告消息（`TFLITE_ENABLE_NNAPI_VERBOSE_VALIDATION=On`）
- 支持外部代理（`TFLITE_ENABLE_EXTERNAL_DELEGATE=On`）
- 构建运行时库，并将其作为共享库提供（`TFLITE_BUILD_SHARED_LIB=On`）。如果首选将 TensorFlow Lite 库静态链接到应用程序，请将此开关保持关闭状态（默认设置）。如果应用程序采用 CMake 构建（如“创建使用 TensorFlow Lite 的 CMake 项目”章节所述），这可能会很方便。
- 使用默认的-O2 优化级别对软件包进行编译。众所周知，一些 CPU 内核（例如 RESIZE_BILINEAR）在 -O3 优化级性能更好，但有些在 -O2 下表现更好，例如 ARG_MAX。我们建议根据应用需求调整优化级别。

Yocto Project 使用这些设置构建 TensorFlow Lite。可以通过更新 meta-imx 层中的 TensorFlow Lite Yocto 配方（位于 `meta-imx/meta-ml/recipes-libraries/tensorflow-lite/`），或使用 CMake 和 Yocto SDK 从源代码构建 TensorFlow Lite 来更改构建配置。

3.5 应用程序开发

本节介绍如何在应用程序开发中使用 TysFraceLite C++ API。

要开始 TensorFlow Lite C++应用程序开发，首先要生成 YOTCO SDK。如需了解如何生成用于交叉编译的 Yocto SDK 环境的详细信息，请参见《i.MX Yocto Project 用户指南》（IMXLXYOCTOUG）。要在主机上激活此 Yocto SDK 环境，请使用以下命令：

```
$ source <Yocto_SDK_install_folder>/environment-setup-aarch64-poky-linux
```

要构建使用 TensorFlow Lite 的应用程序，可以使用以下选项：

- 创建使用 TensorFlow Lite（CMake 超级构建模式）的 CMake 项目

[2] 仅适用于支持 OpenVX 的平台

- 使用 Yocto SDK 预编译库

TensorFlow Lite 的 CMake 配置文件位于根存储库中的 `tensorflow/lite/CMakeLists.txt` 中（适用于恩智浦 i.MX8 平台）。

3.5.1 创建使用 TensorFlow Lite 的 CMake 项目

建议创建一个使用 TensorFlow Lite 的 CMake 项目，如“使用 CMake 构建 TensorFlow Lite”章节所述。CMake 负责依赖项的准备工作，包括下载、配置和构建步骤。

为了演示这个构建选项，`tensorflow/lite/examples/minimal` 中提供了一个最小的示例项目。构建步骤如下所示：

1. 如上所述设置 Yocto SDK
2. 使用 CMake 配置项目：

```
$ mkdir build-minimal-example; cd build-minimal-example
$ cmake -DCMAKE_TOOLCHAIN_FILE=${OE_CMAKE_TOOLCHAIN_FILE} -DTFLITE_ENABLE_XNNPACK=on \
-DTFLITE_ENABLE_RUY=on -DTFLITE_ENABLE_NNAPI=on -DTFLITE_ENABLE_VX=on \
-DTFLITE_ENABLE_NNAPI_VERBOSE_VALIDATION=on \
-DTIM_VX_INSTALL=${SDKTARGETSYSROOT}/usr ../tensorflow/lite/examples/minimal
```

3. 构建项目：

```
$ cmake --build . -j4
```

4. 构建目录中提供了最小的示例：

```
$ file minimal
minimal: ELF 64-bit LSB shared object, ARM aarch64, version 1 (GNU/Linux), dynamically linked,
interpreter /lib/ld-linux-aarch64.so.1, BuildID[sha1]=4a928894439e0b33217ea28790378690ab4ce7cd,
for GNU/Linux 3.14.0, with debug_info, not stripped
```

5. 您可以选择剥离最终的二进制文件：

```
$ $STRIP --remove-section=.comment --remove-section=.note --strip-unneeded <file>
```

此构建选项有若干优势：

- 基于配置选项自动解析依赖性
- 可选择静态链接还是动态链接（`TFLITE_BUILD_SHARED_LIB=on/off`）
- 在调试模式下构建整个项目（包括其依赖项）（`CMAKE_BUILD_TYPE=Debug/Release/…`），以增强调试体验

3.5.2 使用 Yocto SDK 预编译库

另一种选择是使用 Yocto SDK 中直接提供的预编译二进制文件和头文件。TensorFlow Lite 构件位于 Yocto SDK 中，如下所示：

- `/usr/lib` 中的 TensorFlow Lite 共享库 (libtensorflow-lite.so)
- `/usr/include` 中的 TensorFlow Lite 头文件

注意

并非所有 TensorFlow Lite 依赖项都安装在 Yocto SDK 中，需要下载或手动构建。如需了解所需的版本，请参见 `tensorflow/lite/tools/cmake/modules/`文件夹。

要构建位于 `tensorflow/lite/examples/label_image/`中的图像分类演示项目（label_image），请执行以下步骤：

1. 创建构建目录：

```
$ mkdir build-manual
$ cd build-manual
```

2. 下载 Abseil 库依赖项：

```
$ wget https://github.com/abseil/abseil-cpp/archive/
6f9d96a1f41439ac172ee2ef7ccd8edf0e5d068c.tar.gz -O abseil-cpp.tar.gz
$ tar -xzf abseil-cpp.tar.gz
$ mv abseil-cpp-6f9d96a1f41439ac172ee2ef7ccd8edf0e5d068c abseil-cpp
```

3. 构建 label_image 示例：

```
$ $CC ../tensorflow/lite/examples/label_image/label_image.cc ../tensorflow/lite/examples/
label_image/bitmap_helpers.cc ../tensorflow/lite/tools/evaluation/utils.cc -Iabseil-cpp -O2 -
ltensorflow-lite -lstdc++ -lpthread -lm -ldl -lrt
```

3.6 运行图像分类示例

默认情况下包含机器学习层的 Yocto Linux BSP 镜像包含一个名为“label_image”的简单预装示例，可用于图像分类模型。示例二进制文件位于：

```
/usr/bin/tensorflow-lite-2.6.0/examples
```

图 4. TensorFlow 图像分类输入

演示说明：

要在 CPU 上使用 mobilenet 模型运行示例，请使用以下命令：

```
$ ./label_image -m mobilenet_v1_1.0_224_quant.tflite -i grace_hopper.bmp -l labels.txt
```

'grace_hopper.bmp' 输入图像的成功分类输出如下所示：

```
Loaded model mobilenet_v1_1.0_224_quant.tflite
resolved reporter
invoked
average time: 39.271 ms
0.780392: 653 military uniform
0.105882: 907 Windsor tie
0.0156863: 458 bow tie
```

```
0.0117647: 466 bulletproof vest
0.00784314: 835 suit
```

要使用 XNNPACK Delegate 在 CPU 上运行示例应用程序，请使用--usexnnpack=true 开关：

```
$ ./label_image -m mobilenet_v1_1.0_224_quant.tflite -i grace_hopper.bmp -l labels.txt --
use_xnnpack=true
```

要在 GPU/NPU 硬件加速器上运行使用同一模型的示例，请添加--use_nnapi=true（适用于 NNAPI Delegate）或--external_delegate_path=/usr/lib/libvx_delegate.so（适用于 VX Delegate）命令行参数。要区分 3D GPU 和 NPU，请使用 `USE_GPU_INFERENCE` 环境变量。例如，要使用 NNAPI Delegate 在 NPU 硬件上运行模型加速，请使用以下命令：

```
$ USE_GPU_INFERENCE=0 ./label_image -m mobilenet_v1_1.0_224_quant.tflite -i grace_hopper.bmp -l
labels.txt --external_delegate_path=/usr/lib/libvx_delegate.so
```

启用 NPU 加速时的输出应如下所示：

```
INFO: Loaded model ./mobilenet_v1_1.0_224_quant.tflite
INFO: resolved reporter
Vx delegate: allowed_builtin_code set to 0.
Vx delegate: error_during_init set to 0.
Vx delegate: error_during_prepare set to 0.
Vx delegate: error_during_invoke set to 0.
EXTERNAL delegate created.
INFO: Applied EXTERNAL delegate.
W [HandleLayoutInfer:257]Op 18: default layout inference pass.
INFO: invoked
INFO: average time: 2.567 ms
INFO: 0.768627: 653 military uniform
INFO: 0.105882: 907 Windsor tie
INFO: 0.0196078: 458 bow tie
INFO: 0.0117647: 466 bulletproof vest
INFO: 0.00784314: 835 suit
```

或者，可以运行使用 TensorFlow Lite interpreter-only Python API 的示例。示例文件位于：

```
/usr/bin/tensorflow-lite-2.6.0/examples
```

要使用预定义的命令行参数运行该示例，请使用以下命令：

```
$ python3 label_image.py
```

输出应如下所示：

```
Warm-up time: 159.1 ms
Inference time: 156.5 ms
0.878431: military uniform
0.027451: Windsor tie
0.011765: mortarboard
0.011765: bulletproof vest
0.007843: sax
```

Python 示例还支持外部代理。switch --ext_delegate <PATH>和--ext_delegate_options<EXT_DELEGATE_OPTIONS>可用于指定外部代理库及其参数（可选）。

3.7 运行基准测试应用程序

默认情况下，包含机器学习层的 Yocto Linux BSP 镜像包含预装的基准测试应用程序。它执行简单的 TensorFlow Lite 模型推理，并打印基准测试信息。应用程序二进制文件位于：

```
/usr/bin/tensorflow-lite-2.6.0/examples
```

基准测试说明如下所示：

要在 CPU 上运行计算基准测试，请使用以下命令：

```
$ ./benchmark_model --graph=mobilenet_v1_1.0_224_quant.tflite
```

可以选择使用`--num_threads=X` 参数指定线程数，以便在多个内核上进行推理。要获得最高性能，请将 X 设置为可用的内核数。

基准测试应用程序的输出应如下所示：

```
STARTING!
Log parameter values verbosely: [0]
Graph: [mobilenet_v1_1.0_224_quant.tflite]
Loaded model mobilenet_v1_1.0_224_quant.tflite
Going to apply 0 delegates one after another.
The input model file size (MB): 4.27635
Initialized session in 3.051ms.

Running benchmark for at least 1 iterations and at least 0.5 seconds but terminate if exceeding
150 seconds.
count=4 first=160408 curr=155384 min=155384 max=160408 avg=156869 std=2076
Running benchmark for at least 50 iterations and at least 1 seconds but terminate if exceeding
150 seconds.
count=50 first=155586 curr=155424 min=155274 max=155622 avg=155443 std=81

Inference timings in us: Init: 3051, First inference: 160408, Warmup (avg): 156869, Inference
(avg): 155443
Note: as the benchmark tool itself affects memory footprint, the following is only APPROXIMATE to the
actual memory footprint of the model at runtime. Take the information at your discretion.
Peak memory footprint (MB): init=4.49219 overall=10.6133
```

要使用 XNNPACK Delegate 进行推理，请添加`--use_xnnpack=true` 开关：

```
$ ./benchmark_model --graph=mobilenet_v1_1.0_224_quant.tflite --use_xnnpack=true
```

要使用 NNAPI Delegate 的 GPU/NPU 硬件加速器进行推理，请添加`--use_nnapi=true` 开关：

```
$ ./benchmark_model --graph=mobilenet_v1_1.0_224_quant.tflite --use_nnapi=true
```

要使用用于 VX Delegate 的 GPU/NPU 硬件加速器进行推理，请添加`--external_delegate_path=/usr/lib/libvx_delegate.so` 开关：

```
$ ./benchmark_model --graph=mobilenet_v1_1.0_224_quant.tflite --
external_delegate_path=/usr/lib/libvx_delegate.so
```

启用 GPU/NPU 模块加速时的输出（适用于 VX Delegate）应如下所示：

```
STARTING!
Log parameter values verbosely: [0]
Graph: [mobilenet_v1_1.0_224_quant.tflite]
External delegate path: [/usr/lib/libvx_delegate.so]
Loaded model mobilenet_v1_1.0_224_quant.tflite
```

```
Vx delegate: allowed_builtin_code set to 0.
Vx delegate: error_during_init set to 0.
Vx delegate: error_during_prepare set to 0.
Vx delegate: error_during_invoke set to 0.
EXTERNAL delegate created.
Going to apply 1 delegates one after another.
Explicitly applied EXTERNAL delegate, and the model graph will be completely executed by the delegate.
The input model file size (MB): 4.27635
Initialized session in 13.437ms.
Running benchmark for at least 1 iterations and at least 0.5 seconds but terminate if exceeding
150 seconds.
W [HandleLayoutInfer:257]Op 18: default layout inference pass.
count=1 curr=4586473
Running benchmark for at least 50 iterations and at least 1 seconds but terminate if exceeding
150 seconds.
count=398 first=2541 curr=2419 min=2419 max=2549 avg=2467.87 std=13
Inference timings in us: Init: 13437, First inference: 4586473, Warmup (avg): 4.58647e+06, Inference
(avg): 2467.87
Note: as the benchmark tool itself affects memory footprint, the following is only APPROXIMATE to the
actual memory footprint of the model at runtime. Take the information at your discretion.
Peak memory footprint (MB): init=7.24609 overall=34.0117
```

代理不需要支持 TensorFlow Lite Runtime 定义的全套算子。如果模型包含特定代理不支持的运算，则该运算执行将回退到使用 TensorFlow Lite 参考内核的 CPU。这样，由模型表示的计算图形被划分为多个块，然后按块执行。图形分割（graph segmentation 或 graph partitioning）是这样一个过程，即由模型定义的计算图形被划分为更小的块（或分区），每个块通过代理执行或回退到 CPU 上执行（取决于代理支持的算子类型）。

如果在 GPU/NPU 硬件加速器上加速，则基准应用程序还可用于检查模型的可选分块。为此，可以使用`--enable_op_profiling=true`和`--max_delegated_partitions=<big number>`（例如 1000）选项组合。

除了上面给出的输出外，NNAPI Delegate 还报告了代理为什么拒绝特定层的详细信息：

```
INFO: Created TensorFlow Lite delegate for NNAPI.
WARNING: Operator RESIZE_BILINEAR (v1) refused by NNAPI delegate: Operator refused due
performance reasons.
WARNING: Operator RESIZE_BILINEAR (v1) refused by NNAPI delegate: Operator refused due
performance reasons.
WARNING: Operator RESIZE_BILINEAR (v1) refused by NNAPI delegate: Operator refused due
performance reasons.
WARNING: Operator ARG_MAX (v1) refused by NNAPI delegate: NNAPI only supports int32 output.
Explicitly applied NNAPI delegate, and the model graph will be partially executed by the delegate w/
2 delegate kernels.
```

详细的分析信息如下所示：

```
Profiling Info for Benchmark Initialization:
================================= Run Order ===================================
[node type]                  [start]   [first]   [avg ms]       [%]       [cdf%]
ModifyGraphWithDelegate        0.000     4.597      4.597   95.791%      95.791%
AllocateTensors                4.528     0.198      0.101    4.209%     100.000%
======================== Top by Computation Time ==============================
[node type]                  [start]   [first]   [avg ms]       [%]       [cdf%]
ModifyGraphWithDelegate        0.000     4.597      4.597   95.791%      95.791%
AllocateTensors                4.528     0.198      0.101    4.209%     100.000%
Number of nodes executed: 2
=========================== Summary by node type ==============================
```

```
             [Node type] [count][avg ms] [avg %] [cdf %] [mem KB] [times called]
ModifyGraphWithDelegate       1    4.597 95.791% 95.791%  684.000              1
AllocateTensors               1    0.202  4.209% 100.000%   0.000              2
Timings (microseconds): count=1 curr=4799
Memory (bytes): count=0
2 nodes observed
Operator-wise Profiling Info for Regular Benchmark Runs:
================================ Run Order ====================================
        [node type]    [start]    [first]    [avg ms]         [%]        [cdf%]
TfLiteNnapiDelegate      0.000     14.890      14.894     11.349%       11.349%
    RESIZE_BILINEAR     14.896      1.331       1.331      1.014%       12.363%
TfLiteNnapiDelegate     16.227      2.944       2.909      2.216%       14.579%
    RESIZE_BILINEAR     19.137      0.279       0.277      0.211%       14.790%
    RESIZE_BILINEAR     19.415     44.316      44.496     33.905%       48.695%
            ARG MAX     63.912     67.438      67.332     51.305%      100.000%
========================= Top by Computation Time =============================
        [node type]     [start]   [first]   [avg ms]          [%]        [cdf%]
            ARG_MAX      63.912    67.438     67.332      51.305%       51.305%
    RESIZE_BILINEAR      19.415    44.316     44.496      33.905%       85.210%
TfLiteNnapiDelegate       0.000    14.890     14.894      11.349%       96.559%
TfLiteNnapiDelegate      16.227     2.944      2.909       2.216%       98.775%
    RESIZE_BILINEAR      14.896     1.331      1.331       1.014%       99.789%
    RESIZE_BILINEAR      19.137     0.279      0.277       0.211%      100.000%
Number of nodes executed: 6
========================== Summary by node type ===============================
        [Node type] [count] [avg ms]  [avg %]   [cdf %] [mem KB] [times called]
            ARG_MAX       1   67.332  51.306%   51.306%    0.000              1
    RESIZE_BILINEAR       3   46.102  35.129%   86.435%    0.000              3
TfLiteNnapiDelegate       2   17.802  13.565%  100.000%    0.000              2
Timings (microseconds): count=8 first=131198 curr=130580 min=130580 max=132766 avg=131238 std=616
Memory (bytes): count=0
6 nodes observed
```

根据输出中的“执行的节点数”部分，可以确定计算图的哪个部分是在 GPU/NPU 硬件加速器上执行的。除了 TfLiteNnapiDelegate 之外的每个节点都回退到 CPU。在上面的示例中，ARG_MAX 和 RESIZE_BILINEAR 节点回退到 CPU。

3.8 使用 TensorFlow Lite 转换器进行训练后量化

TensorFlow 提供了几种模型量化方法：

- 使用 TensorFlow Lite 转换器进行训练后量化
- 使用模型优化工具包和 TensorFlow Lite 转换器进行量化感知训练
- 以前的 TensorFlow 版本中提供的各种其他方法

注意

“eIQ 工具包”也支持模型量化。另请参见《*eIQ 工具包用户指南*》（EIQTUG）。

本文档未涵盖所有这些内容。本节描述了使用 TensorFlow Lite 转换器进行训练后量化的方法。

转换器作为标准 TensorFlow 桌面安装套件的一部分提供。它用于转换 TensorFlow 模型，用户还可以选择量化为 TensorFlow Lite 模型格式。可通过两个选项使用该工具：

- Python API（推荐）
- 命令行脚本

本章介绍了使用 Python API 的训练后量化。可从以下网址获得有关模型转换和量化的文档：

- Python API 文档：https://www.tensorflow.org/versions/r2.6/api_docs/python/tf/lite/TFLiteConverter
- 模型转换指南：www.tensorflow.org/lite/convert
- 模型量化指南：https://www.tensorflow.org/lite/performance/post_training_quantization
- 模型优化指南：https://www.tensorflow.org/model_optimization

注意

TensorFlow 页面上的指南通常涵盖最新版本的 TensorFlow，可能与恩智浦 eIQ 中提供的版本不同。如需查看可用的功能，请查看 TensorFlow 或 TensorFlow Lite 特定版本的相应 API。

恩智浦 eIQ 中提供的 TensorFlow Lite 的当前版本为 2.6.0。建议使用相应 TensorFlow 版本的 TensorFlow Lite 转换器。TensorFlow Lite runtime 应与以前版本的 TensorFlow Lite 转换器生成的模型兼容，但不保证向后兼容性。应避免使用后续版本的 TensorFlow Lite 转换器。

2.6.0 版本的转换器具有以下特性：

- 在训练后量化机制中，每通道量化是唯一的选项。按张量量化仅适用于量化感知训练。
- 在 `inference_input_type` 和 `inference_output_type` 中设置所需的数据类型，支持输入和输出张量量化。
- 提供基于 TOCO 或 MLIR 的转换。这由 `experimental_new_converter` 属性控制。随着 TOCO 逐渐过时，2.6.0 版本的转换器中已默认设置为基于 MLIR 的转换。

  MLIR 转换器使用动态张量 shape，这意味着未指定输入张量的批量大小。GPU 和 NPU 硬件加速器不支持动态张量 shape，应将其关闭。TensorFlow 的标准安装不提供 API 来控制动态张量 shape 功能，但可以在 TensorFlow 安装中停用，如下所示。找到`<python-install-dir>/site-packages/tensorflow/lite/python/lite.py` 文件并将 private 方法 TFLiteConverterBase._is_unknown_shapes_allowed(self)改为返回 False 值，如下所示：

```
def _is_unknown_shapes_allowed(self):
# Unknown dimensions are only allowed with the new converter.
# Return self.experimental_new_converter
# Disable unknown dimensions support.
return False
```

注意

MLIR 是 TensorFlow 使用的一种新的神经网络（NN）编译器，支持量化。在 MLIR 之前，由现已过时的 TOCO（或 TOCO 转换器）执行量化。详情请参见 https://www.tensorflow.org/api_docs/python/tf/compat/v1/lite/TocoConverter。如需了解 MLIR 的详细信息，请访问 https://www.tensorflow.org/mlir。

注意

不要对在 NN 加速器（GPU 或 NPU）上运行的模型使用动态范围方法。它只将权重转换为 8 位整型，但将激活仍用 32 位浮点，导致在 32 位浮点中运行推理，并增加数据转换的开销。事实上，与 32 位浮点的模型相比，使用动态转换的方法推理速度更慢，因为转换是动态切换的。

对于全整数训练后量化，需要具有代表性的数据集。正确选择代表性数据集中的样本对最终量化模型的精确度有很大影响。创建代表性数据集的最佳方法如下所示：

- 根据模型使用的指标（例如，分类模型的 SoftMax 分数、目标检测模型的 IOU 等），使用原始浮点模型精确度较高的训练样本。
- 代表性数据集中应有充足的样本。

- 就需要的模型精度而言，代表性数据集的大小和其中可用的特定样本被视为需要调整的超参数。

第 4 章
Arm 计算库

Arm 计算库（Arm Compute Library, ACL）是针对 Arm CPU 和 GPU 架构优化的低级函数集合，用于图像处理、计算机视觉和机器学习。

Arm 计算库旨在作为 Arm NN 框架的计算引擎，因此，除非需要更优化的运行时，否则建议使用 Arm NN。

如需获得源代码，请访问 https://source.codeaurora.org/external/imx/arm-computelibrary-imx。

特征：

- Arm Compute Library 21.08
- 在 Cortex-A CPU 内核上使用 Arm Neon SIMD 指令进行加速的多线程计算
- 仅限 C++ API
- 对计算的低级控制

注意
i.MX 8 芯片不支持 GPU OpenCL 后端。

4.1 使用随机权重和输入运行 DNN

Arm 计算库附带了可用于最常见的几种 DNN 架构的示例，这些架构有：AlexNet、MobileNet、ResNet、Inception v3、Inception v4、SqueezeNet 等。

所有可用的示例都可以在以下位置找到：

```
/usr/bin/arm-compute-library-21.08/examples
```

每个模型架构都可以使用 graph_[dnn_model]应用程序进行测试。

例如，要运行 MobileNet v2 DNN 模型，请使用以下命令：

```
$ ./graph_mobilenet_v2 --data=<path_cnn_data> --image=<input_image> --labels=<labels> --target=neon --
type=<data_type> --threads=<num_of_threads>
```

这些参数不是必选参数。如果没有提供，应用程序将使用随机权重和输入数据去运行模型。如果推理成功完成，则会打印“测试通过”（“Test passed"）消息。

4.1.1 使用图形 API 运行 AlexNet

2012 年，AlexNet 赢得了 ImageNet 大规模视觉识别挑战赛（ILSVRC），一举成名，这是一项旨在评估目标检测和图像分类算法的年度挑战赛。AlexNet 由 8 个可训练层组成：5 个卷积层和 3 个全连接层。除最后一个全连接层使用 Softmax 函数外，所有可训练层后面都有 ReLu 激活函数。

使用图形 API 的 C++ AlxNET 实施示例位于该文件夹中：

```
/usr/bin/arm-compute-library-21.08/examples
```

演示说明：

- 将存档文件（compute_library_alexnet.zip）下载到示例位置文件夹。

- 创建新的子文件夹并解压缩该文件：

```
$ mkdir assets_alexnet
$ unzip compute_library_alexnet.zip -d assets_alexnet
```

- 设置执行的环境变量：

```
$ export PATH_ASSETS=/usr/bin/arm-compute-library-21.08/examples/assets_alexnet/
```

- 使用以下命令行参数运行示例：

```
$ ./graph_alexnet --data=$PATH_ASSETS --image=$PATH_ASSETS/go_kart.ppm --labels=$PATH_ASSETS/
labels.txt --target=neon --type=f32 --threads=4
```

成功分类的输出应如下所示：

```
---------- Top 5 predictions ----------
0.9736 - [id = 573], n03444034 go-kart
0.0108 - [id = 751], n04037443 racer, race car, racing car
0.0118 - [id = 518], n03127747 crash helmet
0.0022 - [id = 817], n04285008 sports car, sport car
0.0006 - [id = 670], n03791053 motor scooter, scooter
Test passed
```

第 5 章
Arm NN

Arm NN 是“Linaro 人工智能计划”开发的开源推理引擎框架，恩智浦参与了该计划。它不单独执行计算，而是将多种模型格式（如 TensorFlow Lite 或 ONNX）的输入委托给专门的计算引擎。

如需获得源代码，请访问 https://source.codeaurora.org/external/imx/armnn-imx。

特性：

- Arm NN 21.08
- 使用 ACL Neon 后端提供的 Cortex-A 内核上的 Arm Neon SIMD 指令进行加速的多线程计算
- 使用 VSI NPU 后端提供的 GPU/NPU 硬件加速（在着色器或卷积单元上）进行并行计算
- C++ 和 Python API（支持的 Python 版本 3）
- 支持多种输入格式（TensorFlow Lite、ONNX）
- 用于序列化、反序列化和量化的离线工具（必须使用源代码构建）

5.1 Arm NN 软件栈

Arm NN 软件栈如下图所示。Arm NN 支持以下硬件单元的计算：

- CPU Arm Cortex-A 内核
- 使用 VSI NPU 后端的 GPU/NPU 硬件加速器，它可以在 GPU 和 NPU 上运行，具体取决于哪一个可用

如需了解每个硬件平台对 GPU/NPU 加速器支持情况的详细信息，请参见“软件栈简介”章节。

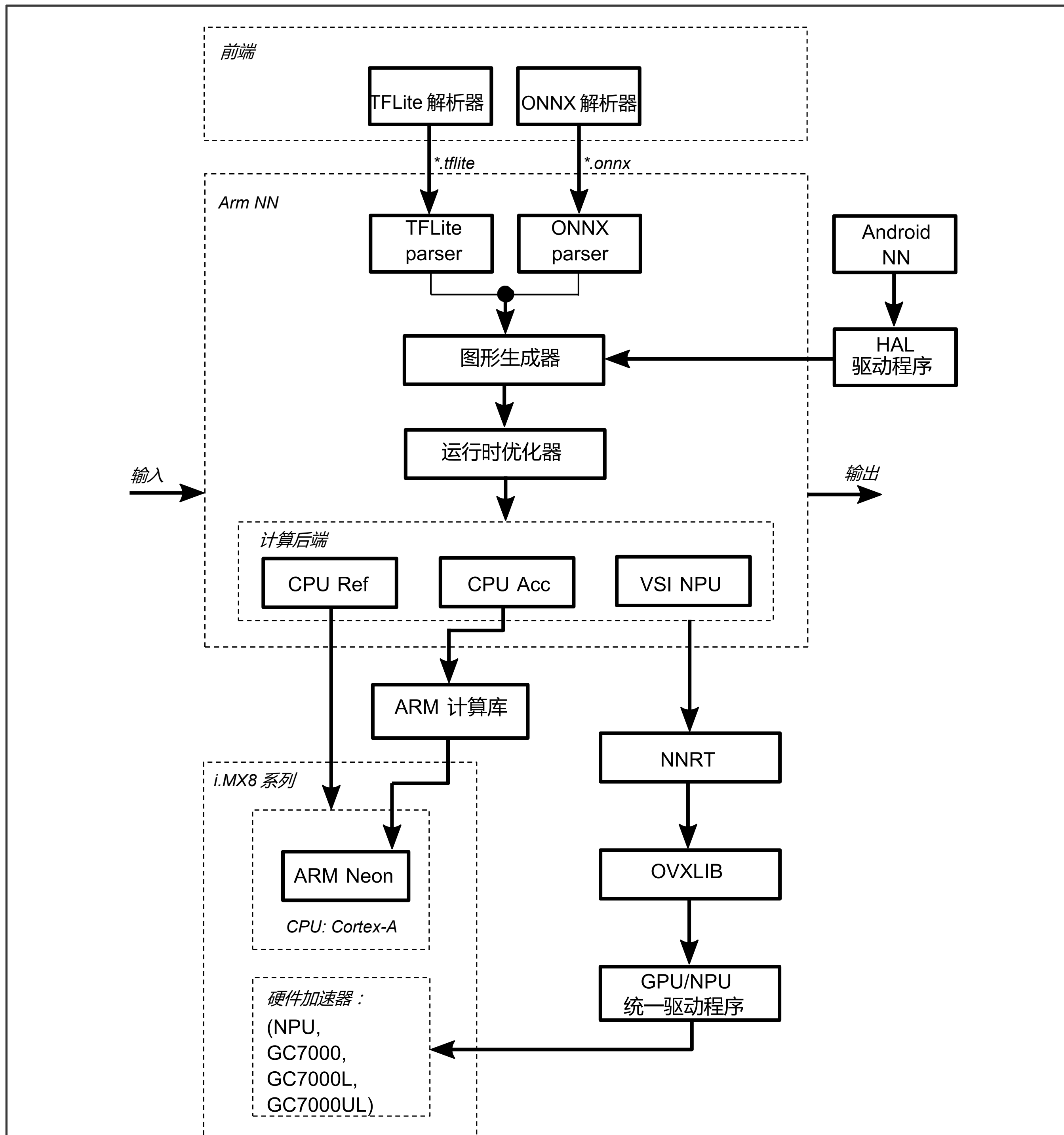

图 5. Arm NN 软件栈

5.2 计算后端

Arm NN 本身并不擅长实现计算操作。CPU 上只有 C++参考后端，它没有优化性能，应该用于测试、检查结果、原型设计，如果其他后端都不支持特定层，则最终回退到 CPU 上执行。其他后端将计算操作委托给其他更专业的库，如 Arm 计算库（ACL）。

- **对于 CPU**：有 NEON 后端，使用带有 Arm NEON SIMD 扩展的 Arm 计算库。

- **对于 GPU 和 NPU**：恩智浦提供 VSI NPU 后端，通过 OpenVX 充分利用 i.MX 8 的 GPU/NPU 的全部功能，并大大提升了性能。由于 i.MX 8 GPU 不满足 Arm NN OpenCL 要求，在源代码中会发现不支持 ACL OpenCL 后端。

要在运行以下部分介绍的示例时激活所选后端，请添加以下参数。用户可以为示例应用程序提供多个后端。模型中的一个层将由第一个后端执行，该后端支持以下层：

```
<example_binary> --compute=arg
```

其中 arg 可以是：

- **CpuRef** = Arm NN C++ 后端（无 SIMD 指令）；一组在 CPU 上没有加速的参考实现，用于测试、原型设计或作为最终回退。它非常慢。
- **CpuAcc** = ACL NEON 后端（在 CPU 上运行，NEON 指令 = SIMD）
- **VsiNpu** = 对于 GPU 和 NPU，恩智浦提供了 VSI NPU 后端，它利用了 i.MX 8 GPU 的全部功能。

要开发自己的应用程序，请确保将所选后端（CpuAcc、VsiNpu 或 CpuRef）传递给 Optimize 函数进行推理。

注意

VsiNpu 后端将执行委托给 OpenVX 驱动程序。工作负载是在 NPU 还是 GPU 上执行取决于驱动程序。

5.3 运行 Arm NN 测试

Arm NN SDK 提供了一组测试，这些测试也可以看作展示 Arm NN 的功能和使用方法的演示。它们加载各种格式的神经网络模型（TensorFlow Lite、ONNX），基于指定的输入数据运行推理，并输出推理结果。Arm NN 测试默认在构建 Yocto 镜像时构建，并安装在/usr/bin/armnn-21.08 中。请注意，输入数据、模型配置和模型权重不随 Arm NN 分发。用户必须单独下载，并确保在运行测试之前在器件上可用。然而，Arm NN 测试不附带文档。输入文件名采用了硬编码，因此请查看代码，以获知预期的输入文件名。

为了帮助您使用 Arm NN，以下各节提供了有关如何准备输入数据以及如何运行 Arm NN 测试的详细信息。它们都使用众所周知的神经网络模型。因此，除了少数例外，此类预训练网络可以在互联网上免费获得。使用代码分析推导出输入图像、模型、格式及其内容。然而，这不可能适用于所有的测试，因为这些模型不是公开提供的，或者无法清楚地推断出应用程序需要哪些输入文件。一般的工作流程是首先在主机上准备数据，然后将其部署到电路板上，在电路板上运行实际的 Arm NN 测试。

以下章节假设神经网络模型文件存储在名为 models 的文件夹中，输入图像文件存储在名为 data 的文件夹中。使用以下命令在较大的分区上创建此文件夹结构：

```
$ cd /usr/bin/armnn-21.08
$ mkdir data
$ mkdir models
```

5.3.1 TensorFlow Lite 测试

Arm NN SDK 为 TensorFlow Lite 模型提供以下测试：

```
/usr/bin/armnn-21.08/TfLiteInceptionV3Quantized-Armnn
/usr/bin/armnn-21.08/TfLiteInceptionV4Quantized-Armnn
/usr/bin/armnn-21.08/TfLiteMnasNet-Armnn
/usr/bin/armnn-21.08/TfLiteMobileNetSsd-Armnn
/usr/bin/armnn-21.08/TfLiteMobilenetQuantized-Armnn
/usr/bin/armnn-21.08/TfLiteMobilenetV2Quantized-Armnn
```

```
/usr/bin/armnn-21.08/TfLiteResNetV2-Armnn
/usr/bin/armnn-21.08/TfLiteVGG16Quantized-Armnn
/usr/bin/armnn-21.08/TfLiteResNetV2-50-Quantized-Armnn
/usr/bin/armnn-21.08/TfLiteMobileNetQuantizedSoftmax-Armnn
/usr/bin/armnn-21.08/TfLiteYoloV3Big-Armnn
```

注意

如需了解支持的算子的完整清单，请参见“TensorFlow Lite 支持”章节。

下表列出了每个 Arm NN TensorFlow Lite 二进制文件示例的所有依赖项。

表 1. Arm NN TensorFlow Lite 示例依赖项

Arm NN 二进制文件	模型文件名称	重命名的输入文件和数据
TfLiteInceptionV3Quantized-Armnn	inception_v3_quant.tflite	shark.jpg, Dog.jpg, Cat.jpg
TfLiteMnasNet-Armnn	mnasnet_1.3_224.tflite	shark.jpg, Dog.jpg, Cat.jpg
TfLiteMobilenetQuantized-Armnn	mobilenet_v1_1.0_224_quant.tflite	shark.jpg, Dog.jpg, Cat.jpg
TfLiteMobilenetV2Quantized-Armnn	mobilenet_v2_1.0_224_quant.tflite	shark.jpg, Dog.jpg, Cat.jpg
TfLiteResNetV2-50-Quantized-Armnn	模型不可用	N/A
TfLiteInceptionV4Quantized-Armnn	模型不可用	N/A
TfLiteMobileNetSsd-Armnn	模型不可用	N/A
TfLiteResNetV2-Armnn	模型不可用	N/A
TfLiteVGG16Quantized-Armnn	模型不可用	N/A
TfLiteMobileNetQuantizedSoftmax-Armnn	模型不可用	N/A
TfLiteYoloV3Big-Armnn	模型不可用	N/A

注意

一些模型或输入文件不公开提供。

执行以下步骤来运行上述每个示例：

1. 下载模型（表格第 2 列）并将其复制到器件上的 *models* 文件夹中。
2. 下载输入数据（表格第 3 列），并复制到器件上的 *data* 文件夹中。根据预期的输入（shark.jpg、Dog.jpg、Cat.jpg）重命名所有 JPG 图像。所有这些名称都区分大小写。
3. 运行测试：

```
$ cd /usr/bin/armnn-21.08
$ ./<armnn_binary> --data-dir=data --model-dir=models
```

5.3.2 ONNX 测试

Arm NN 为 ONNX 模型提供下面的测试：

```
/usr/bin/armnn-21.08/OnnxMnist-Armnn
/usr/bin/armnn-21.08/OnnxMobileNet-Armnn
```

下表列出了每个 Arm NN ONNX 二进制文件示例的所有依赖项。

表 2. Arm NN ONNX 示例依赖项

Arm NN 二进制文件	模型文件名称	重命名的输入文件和数据	重命名的模型文件名
OnnxMnist-Armnn	model.onnx	t10k-images.idx3-ubyte, t10k-labels.idx1-ubyte	mnist_onnx.onnx
OnnxMobileNet-Armnn	mobilenetv2-1.0.onnx	shark.jpg, Dog.jpg, Cat.jpg	mobilenetv2-1.0.onnx

执行以下步骤运行上述每个示例：

1. 下载模型（表格第 2 列）。
2. 将原模型名称重命名为新模型名称（表格第 4 列），并复制到器件上的 *models* 文件夹中。
3. 下载输入数据（表格第 3 列）并复制到器件上的 *data* 文件夹中。
4. 根据预期输入重命名所有 JPG 图片（shark.jpg、Dog.jpg、Cat.jpg）。所有这些名称都区分大小写。
5. 运行测试：

```
$ cd /usr/bin/armnn-21.08
$ ./<armnn_binary> --data-dir=data --model-dir=models
```

5.4 在自定义 C/C++应用程序中使用 Arm NN

您可以使用 Arm NN 功能为 i.MX 8 系列芯片创建自己的 C/C++应用程序。需要使用 Arm NNAPI 编写代码，设置构建依赖项，交叉编译 aarch64 架构的代码，并部署应用程序。以下是这些步骤的详细说明：

1. 编写代码

   要了解如何在自己的应用程序中使用 Arm NNAPI，首先请阅读 Arm 提供的“操作指南”。其中包括展示如何为 MNIST TensorFlow 模型加载和运行推理的应用程序。

2. 准备并安装 SDK。

   从软件开发人员的角度来看，Arm NN 是一个库。因此，要创建和构建使用 Arm NN 的应用程序，您需要头文件和匹配的库。如需了解如何构建 Yocto SDK，请参见《*i.MX Yocto Project 用户指南*》（IMXLXYOCTOUG）。默认情况下，头文件和库并不包括在里面。要确保 SDK 包含头文件和库，请将以下内容添加到 `local.conf`。

   ```
   TOOLCHAIN_TARGET_TASK_append += " armnn-dev"
   ```

3. 构建代码

   要构建 Arm 提供的“armnn mnist”示例，需要进行一些修改，使其能够与 Yocto 交叉编译环境协同工作：

   - 从 Makefile 中删除 *ARMNN_INC* 的定义及其所有用途。默认的 include 目录中已包含 Arm NN 头。
   - 从 Makefile 中删除 *ARMNN_LIB* 的定义及其所有用途。默认链接器搜索路径中已有 Arm NN 库。
   - 在 Makefile 中将“g++”替换为“${CXX}”。

   构建示例：

- 设置 SDK 环境：

```
$ source <Yocto_SDK_install_folder>/environment-setup-aarch64-poky-linux
```

- 运行 make：

```
$ make
```

4. 将构建的应用程序复制到电路板上。

“操作指南”中描述了输入数据。如果电路板上使用的镜像与为其构建 SDK 的镜像相同，那么电路板上应该有运行应用程序所需的所有运行时动态库。

5.5 对接 Arm NN（PyArmNN）的 Python 接口

PyArmNN 是 Arm NN SDK 的 Python 扩展。PyArmNN 提供与 ARM NN C++ API 类似的接口。只有 Python 3.x 支持它，Python 2.x 不支持。

如需获得完整的 API 文档，请参见 NXPmicro GitHub：https://github.com/NXPmicro/pyarmnn-release

5.5.1 开始

要使用 PyArmNN，最简单的方法是使用解析器。下面将演示如何使用解析器：

安装依赖项。

```
pip3 install imageio
```

创建解析器对象并加载模型文件。

```
import pyarmnn as ann
import imageio
# ONNX parser also exist.
parser = ann.ITfLiteParser()
network = parser.CreateNetworkFromBinaryFile('./model.tflite')
```

使用输入层的名称获取输入绑定信息。

```
input_binding_info = parser.GetNetworkInputBindingInfo(0, 'input_layer_name')
# Create a runtime object that will perform inference.
options = ann.CreationOptions()
runtime = ann.IRuntime(options)
```

选择首选后端执行并优化网络。

```
# Backend choices earlier in the list have higher preference.
preferredBackends = [ann.BackendId('CpuAcc'), ann.BackendId('CpuRef')]
opt_network, messages = ann.Optimize(network, preferredBackends, runtime.GetDeviceSpec(),
ann.OptimizerOptions())
# Load the optimized network into the runtime.
net_id, _ = runtime.LoadNetwork(opt_network)
```

使用输入和输出绑定信息生成工作负载张量。

```
# Load an image and create an inputTensor for inference.
# img must have the same size as the input layer; PIL or skimage might be used for resizing if img
has a different size
img = imageio.imread('./image.png')
input_tensors = ann.make_input_tensors([input_binding_info], [img])
# Get output binding information for an output layer by using the layer name.
```

```
output_binding_info = parser.GetNetworkOutputBindingInfo(0, 'output_layer_name')
output_tensors = ann.make_output_tensors([outputs_binding_info])
```

执行推理并将结果返回到 numpy 数组中。

```
runtime.EnqueueWorkload(0,  input_tensors,  output_tensors)
results  =  ann.workload_tensors_to_ndarray(output_tensors)
print(results)
```

5.5.2 运行示例

为了获得更完整的 Arm NN 体验，`/usr/bin/armnn-21.08/pyarmnn/`中有几个示例，它们需要 requests、PIL，可能还需要一些其他 Python3 模块，具体取决于您的图像。您可以使用 pip3 包安装程序安装缺失的模块。例如，对于图像分类演示：

```
$ cd /usr/bin/armnn-21.08/pyarmnn/image_classification
$ pip3 install -r requirements.txt
```

要运行这些示例，请使用 Python3 解析器执行它们。没有参数，资源通过脚本下载。例如，对于图像分类演示：

```
$ python3 tflite_mobilenetv1_quantized.py
```

输出应如下所示：

```
Downloading 'mobilenet_v1_1.0_224_quant_and_labels.zip' from 'https://storage.googleapis.com/
download.tensorflow.org/models/tflite/mobilenet_v1_1.0_224_quant_and_labels.zip' ...
Finished.
Downloading 'kitten.jpg' from 'https://s3.amazonaws.com/model-server/inputs/kitten.jpg' ...
Finished.
Running inference on 'kitten.jpg' ...
class=tabby ; value=99
class=Egyptian cat ; value=84
class=tiger cat ; value=71
class=cricket ; value=0
class=zebra ; value=0
```

注意

`example_utils.py` 是一个包含其余脚本常用函数的文件，它本身不执行任何操作。

5.6 面向 TensorFlow Lite 的 Arm NN Delegate

Arm NN Delegate 是一款独立的软件，可以与 TensorFlow Lite 框架一起使用，以加载 TensorFlow Lite 模型，并将工作负载委托给 Arm NN 库。

注意

5.1052-2.1.0 Yocto 版本只支持 TensorFlow Lite C++ API。Python TensorFlow Lite API 不支持加载动态代理。

5.6.1 ARM NN Delegate C++项目集成

下面的示例演示了使用 TensorFlow Lite 解析器将工作负载委托给 Arm NN 框架。

1. 激活主机上的 Yocto SDK 环境进行交叉编译（确保 SDK 中已安装 tensorflow-lite-dev 和 armnn-dev 软件包，构建 SDK 时应默认已安装），例如：`<yocto_sdk_install_dir>/environment-setup-cortexa53-crypto-poky-linux`
2. 源代码应该在 aarch64 sysroot 目录下，例如：`<yocto_sdk_install_dir>/sysroots/cortexa53-crypto-poky-linux/usr/bin/armnn-21.08/delegate`。交叉编译使用：`$CXX -o armnn_delegate_example armnn_delegate_example.cpp -larmnn -larmnnDelegate -ltensorflow-lite`
3. 将 armnn_delegate_example 复制到电路板并运行。输出应如下所示：

```
$ ./armnn_delegate_example
INFO: TfLiteArmnnDelegate: Created TfLite ArmNN delegate. Warm-
up time: 4662.1 ms
Inference time: 2.809 ms
TOP 1: 412
```

现在让我们看看 armnn_delegate_example.cpp 中的代码：

1. 首先，我们需要加载一个模型，创建 TensorFlow Lite 解析器，并分配适当大小的输入张量。您可以为自己的项目使用不同于下面所示的 tflite 模型：

```
std::unique_ptr<tflite::FlatBufferModel> model
= tflite::FlatBufferModel::BuildFromFile("/usr/bin/tensorflow-lite-2.6.0/examples/
mobilenet_v1_1.0_224_quant.tflite"); auto interpreter = std::make_unique<Interpreter>();
tflite::ops::builtin::BuiltinOpResolver resolver; tflite::InterpreterBuilder(*model, resolver)
(&interpreter); if (interpreter->AllocateTensors() != kTfLiteOk) { std::cout << "Failed to
allocate tensors!" << std::endl; return 0; }
```

2. 然后我们需要用一些数据填充张量。可以从文件加载数据，或者用随机数填充缓冲区。请注意，在我们的示例中，我们使用的是量化模型，因此输入应在<0，255>范围内，且输入张量有 3 个通道和 224x224 个输入：

```
srand (time(NULL));
uint8_t* input = interpreter->typed_input_tensor<uint8_t>(0);
for (int i = 0; i < (3 * 224 * 224); ++i) {
    input[i] = rand() % 256;
}
```

3. 要配置 Arm NN 后端，我们必须指定代理选项。根据层支持，后端从左到右分配给各个层：

```
std::vector<armnn::BackendId> backends = { armnn::Compute::VsiNpu,
armnn::Compute::CpuAcc, armnn::Compute::CpuRef };
armnnDelegate::DelegateOptions delegateOptions(backends);
std::unique_ptr<TfLiteDelegate, decltype(&armnnDelegate::TfLiteArmnnDelegateDelete)>
                    theArmnnDelegate(armnnDelegate::TfLiteArmnnDelegateCreate(delegateOptions),
                                     armnnDelegate::TfLiteArmnnDelegateDelete);
```

4. 现在我们必须将代理应用于图形。这会将图分为多个子图，如果可能，将使用 Arm NN delegate 执行这些子图。其余的将回退到 CPU 的 TensorFlow Lite 内置内核：

```
if (interpreter->ModifyGraphWithDelegate(theArmnnDelegate.get()) != kTfLiteOk)
{
    std::cout << "Failed to modify graph!" << std::endl;
    return EXIT_FAILURE;
}
```

5. 之后我们可以进行推理，检索结果，并对其进行处理。mobilenet 模型的输出是一个 softmax 数组，因此要检索顶部标签，我们必须应用 argmax 函数。请注意，在这个示例中，我们运行了两次推理。这是由于使用了预热时间较长的 VsiNpu 后端：

```
if (interpreter->Invoke() != kTfLiteOk)
{
    std::cout << "Failed to run second inference!" << std::endl;
    return EXIT_FAILURE;
}
...
uint8_t* output = interpreter->typed_output_tensor<uint8_t>(0);
```

第 6 章 ONNX Runtime

ONNX Runtime 是一个运行 ONNX 模型的开源推理引擎，它可以使用一组 API 对所有部署目标加速机器学习模型。如需获得源代码，请访问 https:// source.codeaurora.org/external/imx/onnxruntime-imx。

注意

如需获得 CPU 支持的算子的完整清单，请参见“算子内核”文档的“算子内核”章节。

特性：

- ONNX Runtime 1.8.2
- 使用 ACL 和 Arm NN 执行提供程序提供的 Cortex-A 内核上的 Arm Neon SIMD 指令进行加速的多线程计算
- 使用 VSI NPU 和 NNAPI 执行提供程序提供的 GPU/NPU 硬件加速（在着色器或卷积单元上）进行并行计算
- C++ 和 Python API（支持的 Python 版本 3）
- ONNX Runtime 1.8.2 支持 ONNX 1.9 和 opset 14。

注意

Opset 只定义所有可用的算子。这并不一定意味着它们是在正在使用的执行提供程序中实现的。如需了解更多详细信息，请参见“执行提供程序”章节。

6.1 ONNX Runtime 软件栈

ONNX Runtime 软件栈如下图所示。ONNX Runtime 支持在以下硬件单元上进行计算：

- 使用 CPU、ACL 和 Arm NN 执行提供程序的 CPU Arm Cortex-A 内核
- 使用 VSI NPU 或 NNAPI 执行提供程序的 GPU/NPU 硬件加速器

如需了解在不同硬件平台上支持 GPU/NPU 硬件加速器计算的详细信息，请参见“软件栈简介”章节。

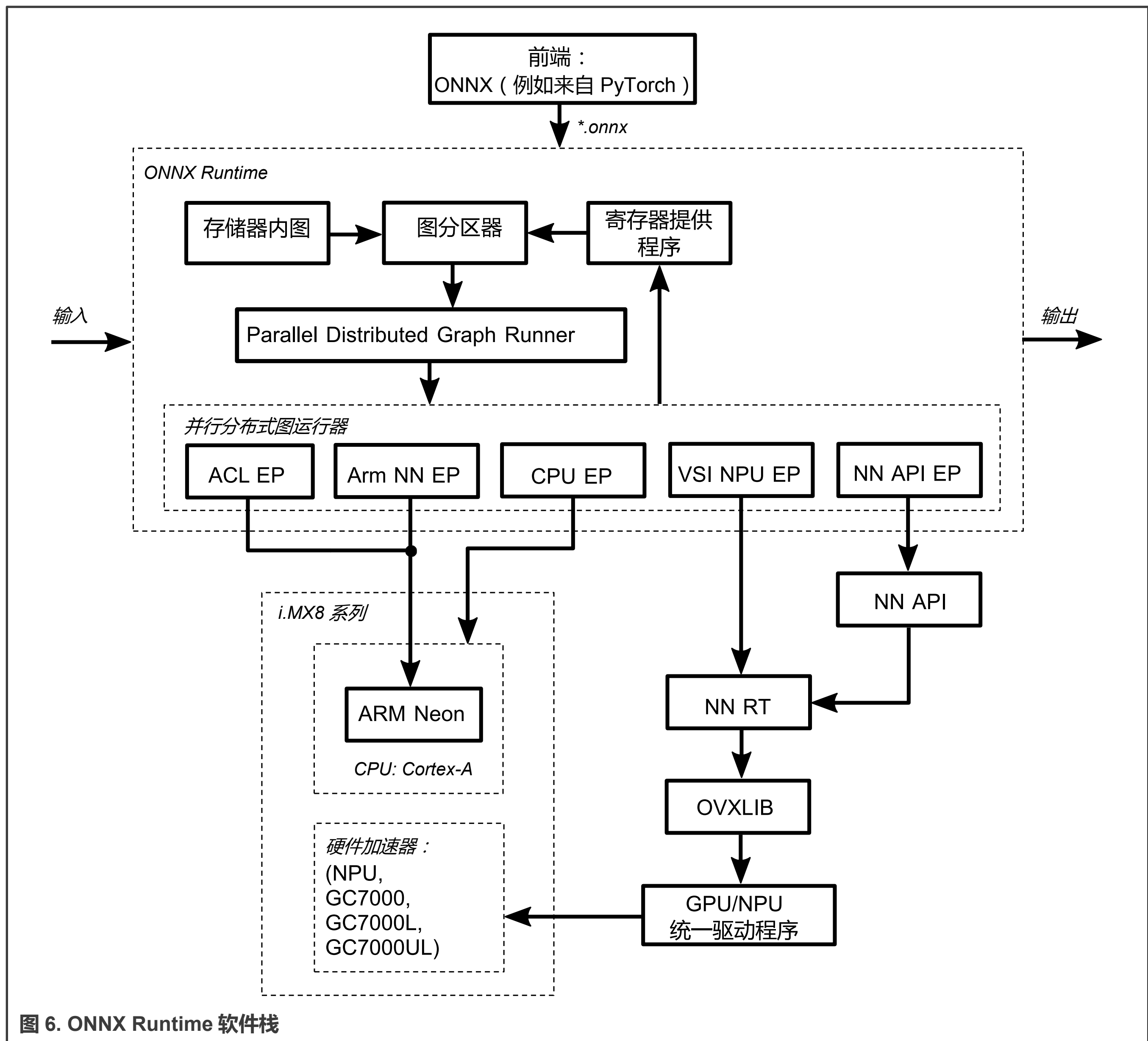

图 6. ONNX Runtime 软件栈

6.2 执行提供程序

执行提供程序（execution provider, EP）是一种将推理执行委托给底层框架或硬件的机制。默认情况下，ONNX Runtime 使用 CPU EP，在 CPU 上执行推理。

与默认 CPU EP 相比，官方支持的执行提供程序提供以下加速方式：

- acl - 在 CPU 上运行，并使用 Arm 计算库中的 NEON 实现直接加速。
- armnn - 在 CPU 上运行，并利用 Arm 计算库中的 NEON 后端实现加速。
- vsi_npu - 在 GPU 或 NPU 上运行，具体取决于可用的硬件。直接利用 OpenVX 实施。
- nnapi - 在 GPU 或 NPU 上运行，具体取决于可用的硬件。利用使用 OpenVX 的 NNAPI 实施。

6.2.1 ONNX 模型测试

ONNX Runtime 提供了一个工具，可以运行 ONNX Model Zoo 中提供的标准测试集合。名为 onnx_test_runner 的工具安装在 `/usr/bin/onnxruntime-1.8.2` 中。

ONNX 模型可在以下网址获得：https://github.com/onnx/models，由模型和样本测试数据组成。由于某些模型需要大量的磁盘空间，建议将 ONNX 测试文件存储在更大的分区上，如“SD 卡镜像闪存”章节所述。

以下是运行 mobilenet 版本 2 测试所需步骤的示例：

- 下载 mobilenet 版本 2 测试档案并将其解压缩到某个文件夹，例如`/home/root`：

```
$ cd /home/root
$ wget https://github.com/onnx/models/raw/master/vision/classification/mobilenet/model/
mobilenetv2-7.tar.gz
$ tar -xzvf mobilenetv2-7.tar.gz
$ ls ./mobilenetv2-7
mobilenetv2-7.onnx  test_data_set_0  test_data_set_1  test_data_set_2
```

- 运行 onnx_test_runner 工具，提供 `mobilenetv2-7` 文件夹路径，并将执行提供程序设置为 Arm NN：

```
$ /usr/bin/onnxruntime-1.8.2/onnx_test_runner -j 1 -c 1 -r 1 -e [cpu/armnn/acl/vsi_npu/nnapi] ./
mobilenetv2-7/
result:
Models: 1
Total test cases: 3
Succeeded: 3
Not implemented: 0
Failed: 0
Stats by Operator type:
Not implemented(0):
Failed:
Failed Test Cases:
$
```

注意

使用 `onnx_test_runner -h` 查看受支持选项的完整列表。

6.2.2 C API

ONNX Runtime 还提供了一个 C API 示例代码，如下所述：
https://github.com/microsoft/onnxruntime/blob/v1.8.2/docs/C_API_Guidelines.md。

要通过存储库构建示例，请在生成的 Yocto SDK 环境下运行以下 build 命令（确保 SDK 中已安装了 onnxruntime-dev Yocto 软件包，默认已安装）：

```
$CXX -std=c++0x -I$SDKTARGETSYSROOT/usr/include/onnxruntime/core/session -lonnxruntime
C_Api_Sample.cpp -o onnxruntime_sample
```

注意

BSP 中包含的 SqueezeNet 模型可以与可执行文件一起使用。

6.2.2.1 启用执行提供程序

要启用特定的执行提供程序，需要在代码中执行以下操作：

- 在代码中设置执行提供程序（请参见前面的 C API 示例，了解如何为 CUDA EP 执行此操作）。如果未设置，将使用默认 CPU EP：OrtSessionOptionsAppendExecutionProvider_<execution_provider>(<parameters>)；
- 基于代码中使用的 EP 的 Include 标头：#include "<execution_provider>_provider_factory.h"。
- 将 include 添加到 build 命令：-I/usr/include/onnxruntime/core/providers/<execution_provider>/

6.2.3 ONNX 性能测试

为了运行模型基准测试，ONNX Runtime 提供了测量性能的工具。这个名为 onnxruntime_perf_test 的工具安装在 /usr/bin/onnxruntime-1.8.2 中。为了运行它，用户必须提供一个*.onnx* 模型文件和测试数据。要使用 VSI NPU 执行提供程序对运行单个迭代的 SqueezeNet 模型进行基准测试，请运行以下命令：

```
$/usr/bin/onnxruntime-1.8.2/onnxruntime_perf_test /usr/bin/onnxruntime-1.8.2/squeezenet/model.onnx -r
1 -e vsi_npu
```

注意

使用 onnxruntime_perf_test -h 查看受支持选项的完整列表。

第 7 章
PyTorch

Pytork 是一个基于 Python 的科学计算软件包，有助于利用图形处理单元的强大功能构建深度学习项目。

特性：

- PyTorch 1.7.1
- 具有强大 GPU 加速能力的张量计算（如 NumPy）
- 建立在基于磁带的 autograd 系统上的深度神经网络

注意

此版本的 PyTorch 尚不支持恩智浦 GPU/NPU 上的张量计算。只支持 CPU。默认情况下，PyTorch runtime 使用浮点模型运行。要启用量化模型，应明确指定量化引擎，如下所示：

```
torch.backends.quantized.engine = 'qnnpack'
```

7.1 运行图像分类示例

examples 文件夹中有一个示例，它需要 urllib、PIL，可能还需要一些其他 Python3 模块，具体取决于您的图像。您可以使用 pip3 安装缺失的模块。

```
$ cd /usr/bin/pytorch/examples
```

要在 CPU 上运行推理计算示例，请使用以下命令。没有参数，脚本将自动下载资源：

```
$ python3 pytorch_mobilenetv2.py
```

输出应如下所示：

```
File does not exist, download it from https://download.pytorch.org/models/mobilenet_v2-
b0353104.pth
... 100.00%, downloaded size: 13.55 MB
File does not exist, download it from
https://raw.githubusercontent.com/Lasagne/Recipes/master/examples/resnet50/imagenet_classes.txt
... 100.00%, downloaded size: 0.02 MB
File does not exist, download it from
https://s3.amazonaws.com/model-server/inputs/kitten.jpg
... 100.00%, downloaded size: 0.11 MB
('tabby, tabby cat', 46.34805679321289)
('Egyptian cat', 15.802854537963867)
('lynx, catamount', 1.1611212491989136)
('lynx, catamount', 1.1611212491989136)
('tiger, Panthera tigris', 0.20774540305137634)
```

7.2 构建和安装 wheel 包

此版本包括 aarch64 平台 PyTorch 和 TorchVision 构建脚本。目前，它支持使用 BSP SDK 在恩智浦 aarch64 平台上进行原生构建。

注意

通常，在 BSP SDK 的 yocto rootfs 中已经集成了 PyTorch 和 TorchVision wheel 包。无需从头开始构建和安装。如果您想自己构建，请执行以下步骤。

7.2.1 如何构建

请执行以下步骤：

1．从 https://source.codeaurora.org/external/imx/imx-manifest 获取最新的 i.MX BSP。

2．为其中一个恩智浦 aarch64 平台搭建构建环境，并编辑 *local.conf*，为 PyTorch 原生构建添加以下依赖项：

```
IMAGE_INSTALL_append = " python3-dev python3-pip python3-wheel python3-pillow python3-setuptools
python3-numpy python3-pyyaml
python3-cffi python3-future cmake ninja packagegroup-core-buildessential git git-perltools
libxcrypt libxcrypt-dev
```

3．使用以下命令构建 BSP 镜像：

```
$ bitbake imx-image-full
```

4．进入 pytorch 文件夹，在恩智浦 aarch64 平台上执行 build 脚本，生成 wheel 包。如需获得源代码，请访问 https://github.com/NXPmicro/pytorch-release：

```
$ cd /path/to/pytorch/src
$ ./build.sh
```

7.2.2 如何安装

如果构建成功，wheel 包应该在*/path/to/pytorch/src/dist* 下：

```
$ pip3 install /path/to/torch-1.7.1-cp37-cp37m-linux_aarch64.whl
$ pip3 install /path/to/torchvision-0.8.2-cp37-cp37m-linux_aarch64.whl
```

第 8 章
OpenCV 机器学习演示

OpenCV 是一个开源计算机视觉库，其中一个名为 ML 的模块提供传统的机器学习算法。OpenCV 为神经网络推理（DNN 模块）和经典机器学习算法（ML 模块）提供了统一的解决方案。

特性：

- OpenCV 4.5.2
- C++ 和 Python API（支持的 Python 版本 3）
- 仅支持 CPU 计算
- 支持输入图像或实时摄像头（网络摄像头）

8.1 下载 OpenCV 演示

OpenCV DNN 演示（二进制文件）位于：

```
/usr/share/OpenCV/samples/bin
```

输入数据和模型配置位于：

```
/usr/share/opencv4/testdata/dnn
```

注意

要将“testdata/dnn”目录放在镜像上，请在构建镜像之前将以下内容放在 `local.conf` 中。请参见《*i.MX Yocto Project 用户指南*》（IMXLXYOCTOUG）中的“恩智浦 eIQ 机器学习”章节。

```
PACKAGECONFIG_append_pn-opencv_mx8 += " tests tests-imx"
```

由于尺寸限制，二进制模型不在镜像中。在运行 DNN 演示之前，应将以下文件下载到器件：

```
$ cd /usr/share/opencv4/testdata/dnn/
$ python3 download_models_basic.py
```

注意

如果需要所有可能的模型和配置文件（需要 10 GB SD 卡），请使用 `download_models.py` 脚本。如果只需要以下 DNN 示例的基本模型（需要 1 GB SD 卡），请使用 `download_models_basic.py` 脚本。

将所有可下载的依赖项（模型、输入和权重）复制到：

```
/usr/share/OpenCV/samples/bin
```

下载配置 model.yml。该文件包含一些 DNN 示例的预处理参数，接受`--zoo` 参数。将模型文件复制到：

```
/usr/share/OpenCV/samples/bin
```

8.2 OpenCV DNN 演示

OpenCV DNN 模块实施了一个推理引擎，不提供任何神经网络训练功能。

8.2.1 图像分类演示

本演示使用预训练的 SqueezeNet 网络进行图像分类。演示依赖项来自 opencv_extra-4.5.2.zip 或：

```
/usr/share/opencv4/testdata/dnn
```

- dog416.png
- squeezenet_v1.1.caffemodel
- squeezenet_v1.1.prototxt

其他演示依赖项：

- classification_classes_ILSVRC2012.txt 来自

  ```
  /usr/share/OpenCV/samples/data/dnn
  ```

- 来自 github 的 models.yml

使用默认位置的图像输入运行 C++ 示例：

```
$ ./example_dnn_classification --input=dog416.png --zoo=models.yml squeezenet
```

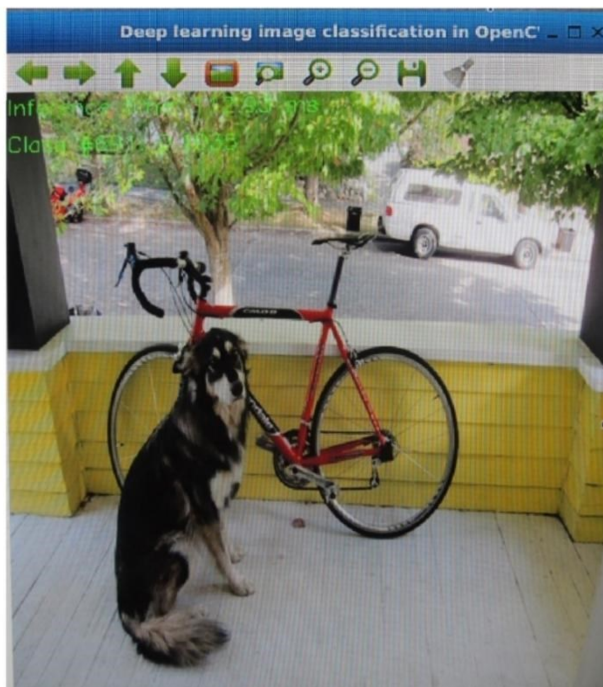

图 7. 图像分类图形输出

用连接到端口 3 的实时摄像头运行 C++示例：

```
$ ./example_dnn_classification --device=3 --zoo=models.yml squeezenet
```

注意

选择摄像头当前连接的正确端口。使用 `v4l2-ctl --list-devices` 命令进行检查。

8.2.2 YOLO 目标检测示例

YOLO 目标检测演示使用 You Only Look Once（YOLO）检测器执行目标检测。它可检测摄像头、视频或图像上的物体。如需了解有关此演示的更多信息，请查看 OpenCV Yolo DNNs 页面。演示依赖项来自 opencv_extra-4.5.2.zip 或：

```
/usr/share/opencv4/testdata/dnn
```

- dog416.png
- yolov3.weights
- yolov3.cfg

其他演示依赖项

- 来自 github 的 models.yml
- object_detection_classes_yolov3.txt，来自

```
/usr/share/OpenCV/samples/data/dnn
```

在默认位置使用图像输入运行 C++ 示例：

```
$ ./example_dnn_object_detection -width=1024 -height=1024 -scale=0.00392 -input=dog416.png -rgb -
zoo=models.yml yolo
```

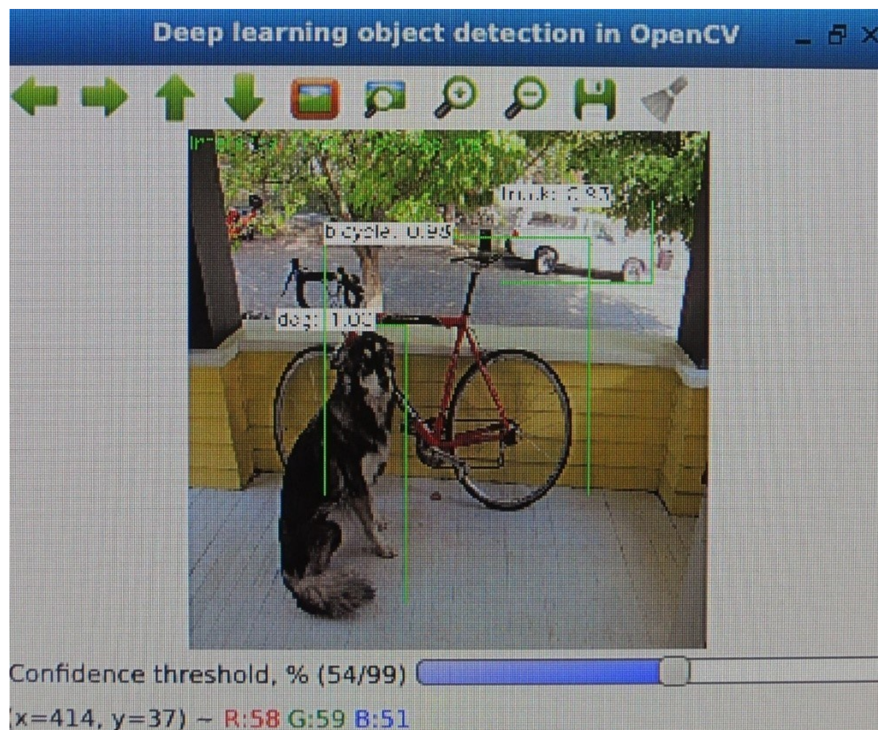

图 8. YOLO 目标检测图形输出

使用连接到端口 3 的实时摄像头运行 C++ 示例：

```
$ ./example_dnn_object_detection -width=1024 -height=1024 -scale=0.00392 --device=3 -rgb -
zoo=models.yml yolo
```

注意

选择摄像头当前连接的端口。 使用 `v4l2-ctl --list-devices` 命令进行检查。

注意

使用实时摄像头输入运行此示例非常慢，因为仅在 CPU 上运行该示例。

8.2.3 图像分割演示

图像分割是指根据一些标准将图像划分为像素组。根据颜色、纹理或其他标准进行分组。演示依赖项来自 opencv_extra-4.5.2.zip 或：

```
/usr/share/opencv4/testdata/dnn
```

- dog416.png
- fcn8s-heavy-pascal.caffemodel
- fcn8s-heavy-pascal.prototxt

其他演示依赖项是来自 github 的 models.yml。在默认位置使用图像输入运行 C++示例：

```
$ ./example_dnn_segmentation --width=500 --height=500 --rgb --mean=1 --input=dog416.png --
zoo=models.yml fcn8s
```

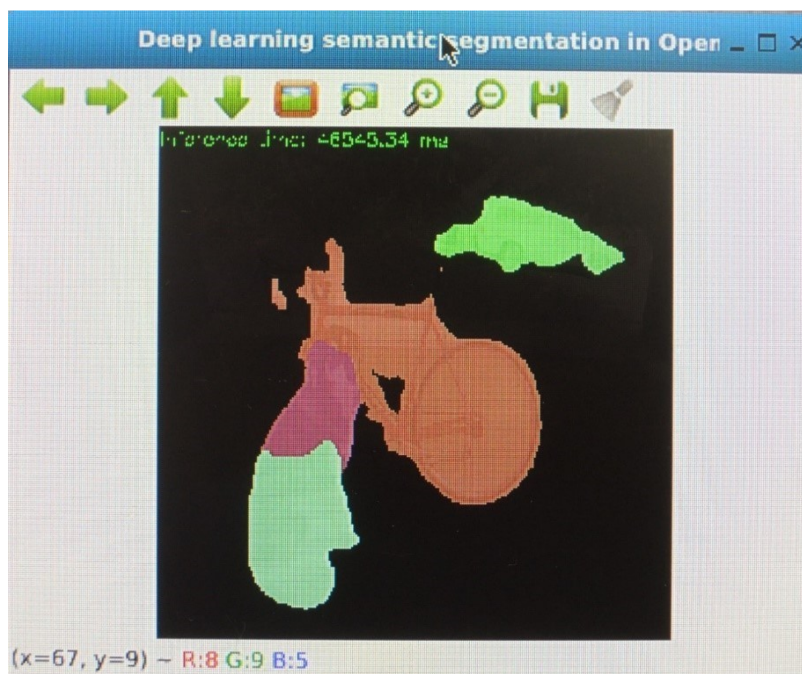

图 9. 图像分割图形输出

用连接到端口 3 的实时摄像头运行 C++示例：

```
$ ./example_dnn_segmentation --width=500 --height=500 --rgb --mean=1 --device=3 --zoo=models.yml fcn8s
```

注意

选择摄像头当前连接的端口。使用 `v4l2-ctl --list-devices` 命令进行检查。

注意

使用实时摄像头输入运行该示例非常慢，因为只在 CPU 上运行该示例。

8.2.4 图像着色演示

此示例演示了如何使用 DNN 对灰度图像进行重新着色。该演示仅支持输入图像，不支持实时摄像头输入。演示依赖项来自 opencv_extra-4.5.2.zip 或来自：

```
/usr/share/opencv4/testdata/dnn
```

- colorization_release_v2.caffemodel
- colorization_deploy_v2.prototxt

其他演示依赖项是 basketball1.png，来自

```
/usr/share/OpenCV/examples/data
```

在默认位置使用图像输入运行 C++ 示例：

```
$ ./example_dnn_colorization --model=colorization_release_v2.caffemodel --
proto=colorization_deploy_v2.prototxt --image=../data/basketball1.png
```

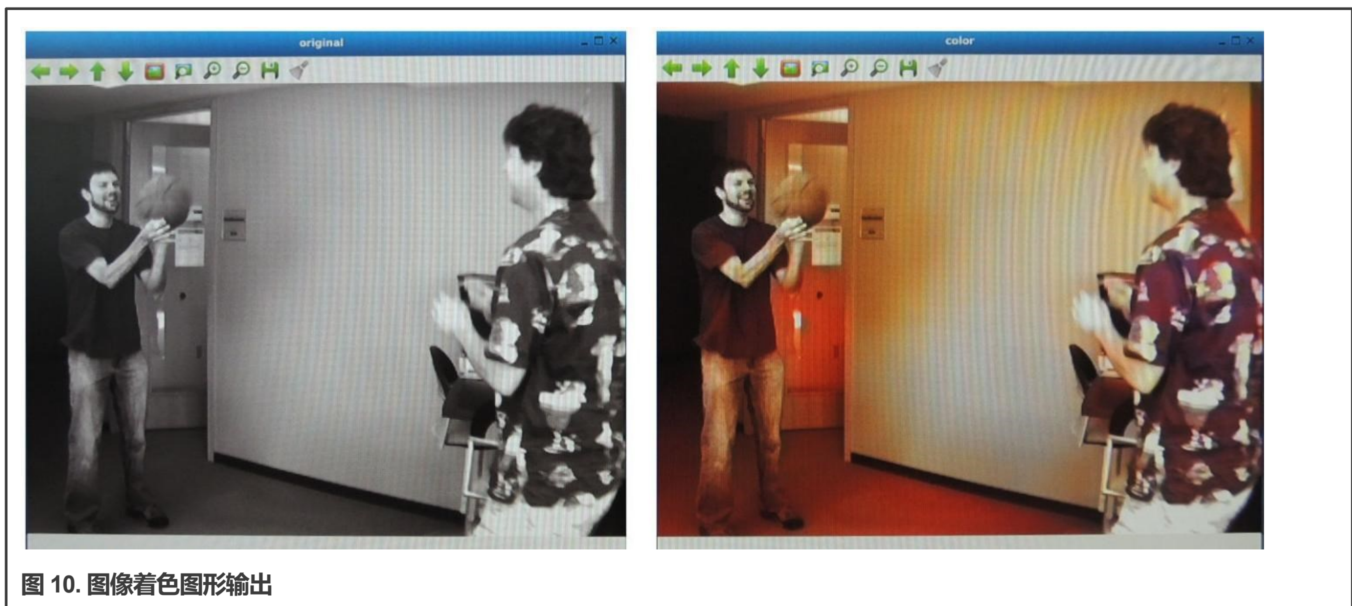

图 10. 图像着色图形输出

8.2.5 人体姿势检测演示

此应用程序演示了使用预训练的 OpenPose DNN 进行人体或手部姿势检测。该演示仅支持输入图像，不支持实时摄像头输入。演示依赖项来自 opencv_extra-4.5.2.zip 或来自：

```
/usr/share/opencv4/testdata/dnn
```

- grace_hopper_227.png
- openpose_pose_coco.caffemodel
- openpose_pose_coco.prototxt

在默认位置使用图像输入运行 C++ 示例：

```
$ ./example_dnn_openpose --model=openpose_pose_coco.caffemodel --proto=openpose_pose_coco.prototxt --
image=grace_hopper_227.png --width=227 --height=227 --dataset=COCO
```

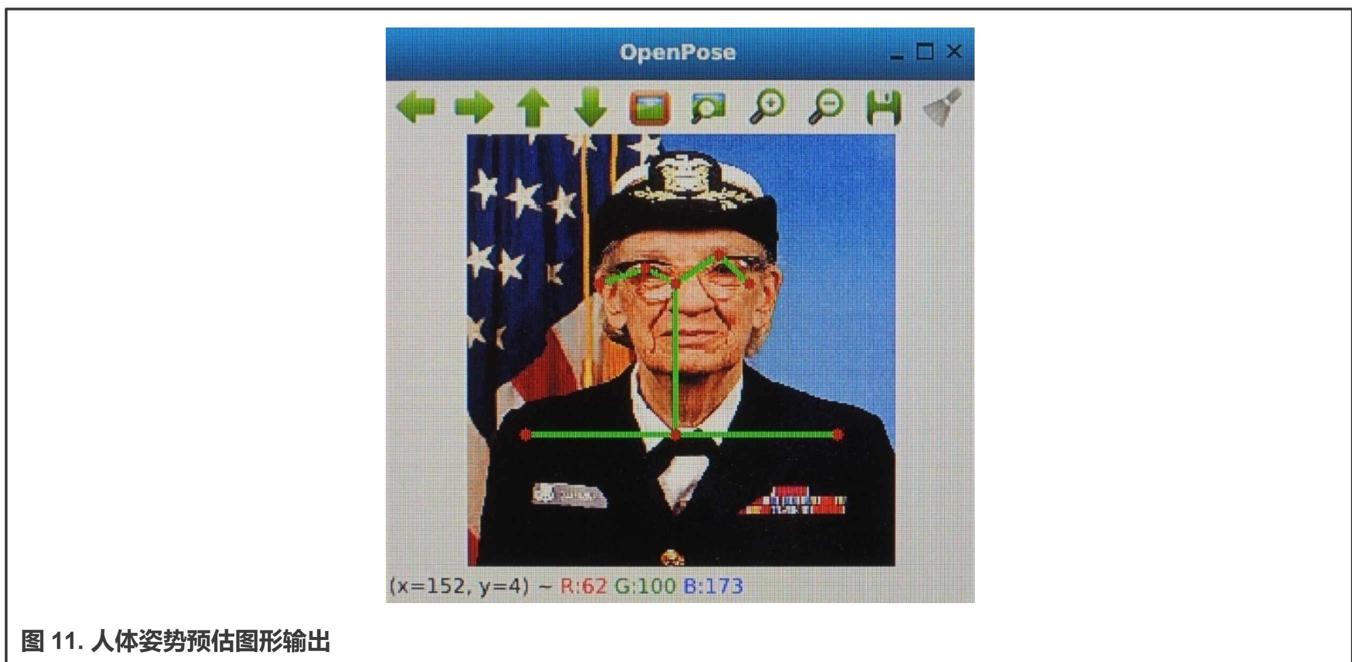

图 11. 人体姿势预估图形输出

8.2.6 目标检测示例

本演示使用预训练的 SqueezeDet 网络进行目标检测。该演示仅支持输入图像，不支持实时摄像头输入。演示依赖项如下所示：

- SqueezeDet.caffemodel 模型权重文件
- SqueezeDet_deploy.prototxt 模型定义文件
- 输入图像 aeroplane.jpg

在默认位置使用图像输入运行 C++ 示例：

```
$ ./example_dnn_objdetect_obj_detect SqueezeDet_deploy.prototxt SqueezeDet.caffemodel aeroplane.jpg
```

在 aeroplane.jpg 图像上运行模型会在控制台中生成以下文本结果：

```
------
Class: aeroplane
```

```
Probability: 0.845181
Co-ordinates:
```

图 12. 目标检测图形输出

8.2.7 CNN 图像分类示例

该演示使用预训练的 SqueezeNet 网络进行图像分类。该演示仅支持输入图像，不支持实时摄像头输入。演示依赖项如下所示：

- SqueezeNet.caffemodel 模型权重文件
- SqueezeNet_deploy.prototxt 模型定义文件
- 输入图片 space_shuttle.jpg，来自

```
/usr/share/opencv4/testdata/dnn
```

在默认位置使用图像输入运行 C++ 示例：

```
$ ./example_dnn_objdetect_image_classification SqueezeNet_deploy.prototxt SqueezeNet.caffemodel
space_shuttle.jpg
```

在 space_shuttle.jpg 图像上运行模型会在控制台中生成以下文本结果：

```
Best class Index: 812
Time taken: 0.649153
Probability: 15.8467
```

8.2.8 文本检测

该演示展示如何使用 EAST 算法检测图像中的文本。演示依赖项如下所示：

- 基于 EAST 的 freeze_east_text_detection.pb 模型文件
- crnn_cs.onnx 文本识别模型

其他演示依赖项：

- 输入文件，来自

  ```
  /usr/share/OpenCV/samples/data/imageTextN.png
  ```

- 用于基准评估的词汇文件，来自

  ```
  /usr/share/OpenCV/samples/data/alphabet_94.txt
  ```

在默认位置使用图像输入运行 C++ 示例：

```
$ ./example_dnn_text_detection --detModel=frozen_east_text_detection.pb --input=../data/
imageTextN.png --recModel=crnn_cs.onnx --vp=../data/alphabet_94.txt --rgb=1
```

注意

此示例仅接受 PNG 图像格式。

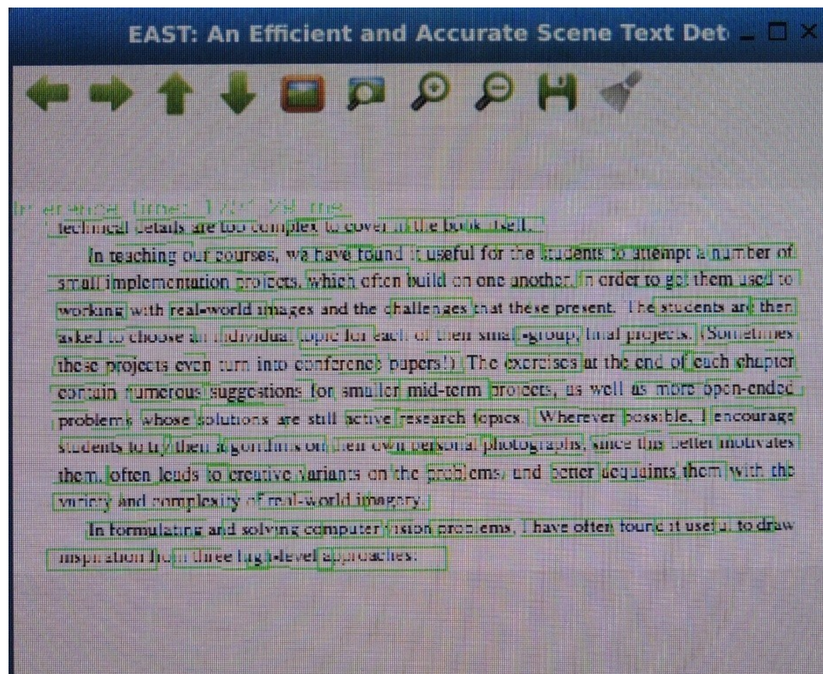

图 13. 文本检测图形输出

使用连接到端口 3 的实时摄像头运行 C++ 示例：

```
$ ./example_dnn_text_detection --detModel=frozen_east_text_detection.pb --recModel=crnn_cs.onnx --
vp=../data/alphabet_94.txt --rgb=1 --device=3
```

注意

选择摄像头当前连接的端口。 使用 `v4l2-ctl --list-devices` 命令进行检查。

8.3 OpenCV 经典机器学习演示

在目标设备上部署了 OpenCV 后，非神经网络演示安装在 rootfs 中

```
/usr/share/OpenCV/samples/bin/
```

8.3.1 SVM 简介

此示例演示了如何使用训练数据创建和训练 SVM 模型。模型经过训练后，会预测测试数据的标签。 有关该示例的完整描述，请参见（tutorial_introduction_to_svm）。要显示结果，需要启用 Qt5 的图像。

运行演示后，屏幕上会显示图形结果：

```
$ ./example_tutorial_introduction_to_svm
```

结果如下所示：

- 代码打开一个图像，显示两类训练示例。一类示例的点用白色圆圈表示，另一类示例用黑色点表示。
- SVM 经过训练，用于对图像的所有像素进行分类。将图像分为蓝色区域和绿色区域。两个区域之间的边界是最佳分离超平面。
- 最后，在训练示例周围使用灰色环显示支持向量。

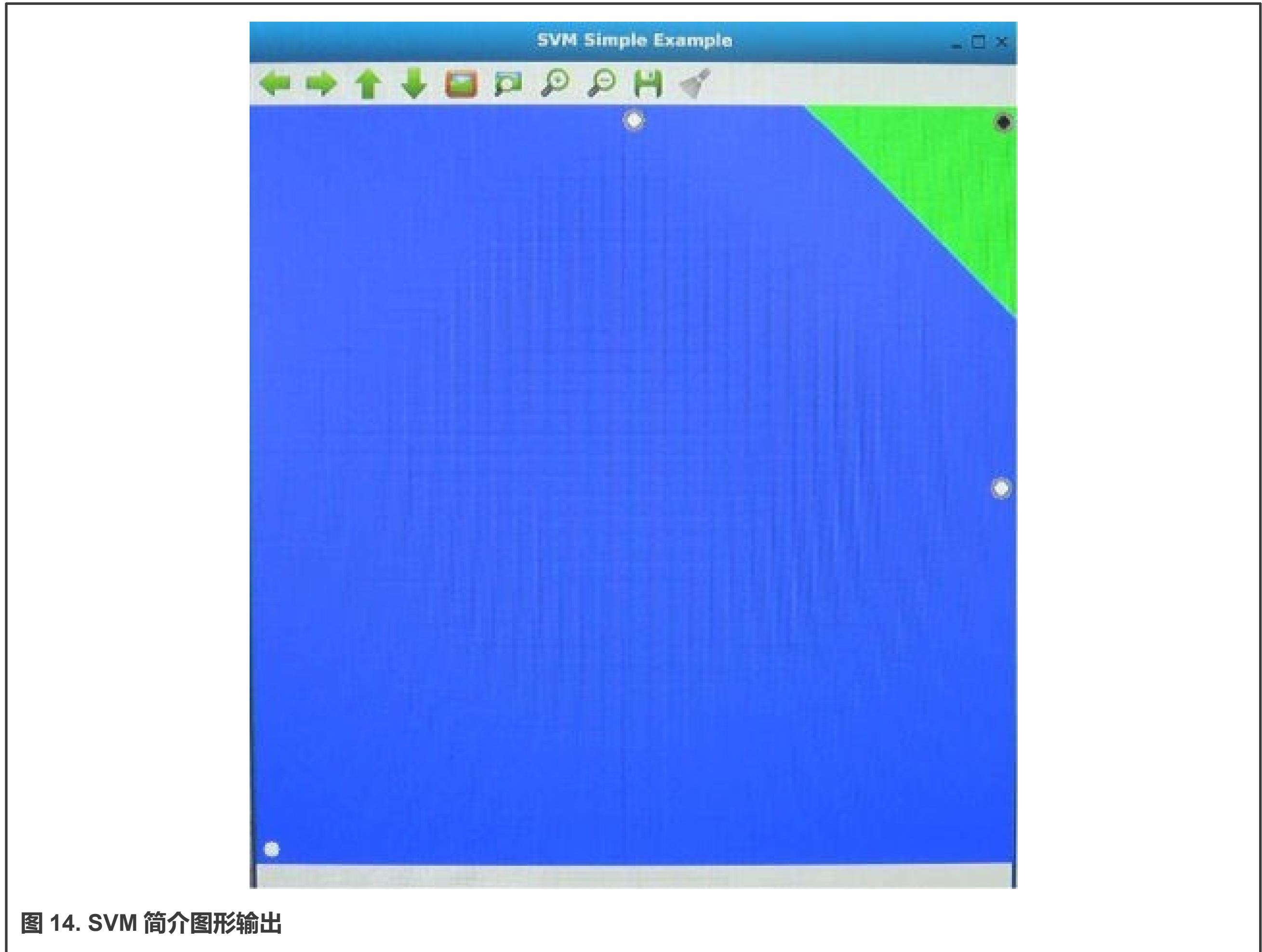

图 14. SVM 简介图形输出

8.3.2 非线性可分离数据的 SVM

此示例处理非线性可分离数据，并展示如何为此类数据设置具有线性核的 SVM 参数。如需了解更多详细信息，请查看 SVM_non_linearly_separable_data。

运行演示后，屏幕上显示图形结果（需要 Qt5 支持）：

```
$ ./example_tutorial_non_linear_svms
```

结果如下所示：

- 代码打开一个图像，显示两类训练数据。一类示例的点用浅绿色表示，另一类用浅蓝色的点表示。
- SVM 经过训练，用于对图像的所有像素进行分类。将图像划分为蓝色区域和绿色区域。两个区域之间的边界是分离超平面。由于训练数据是非线性可分离的，两个类的一些示例都被错误分类；一些绿点位于蓝色区域，一些蓝点位于绿色区域。
- 最后，在训练示例周围使用灰色环显示支持向量。

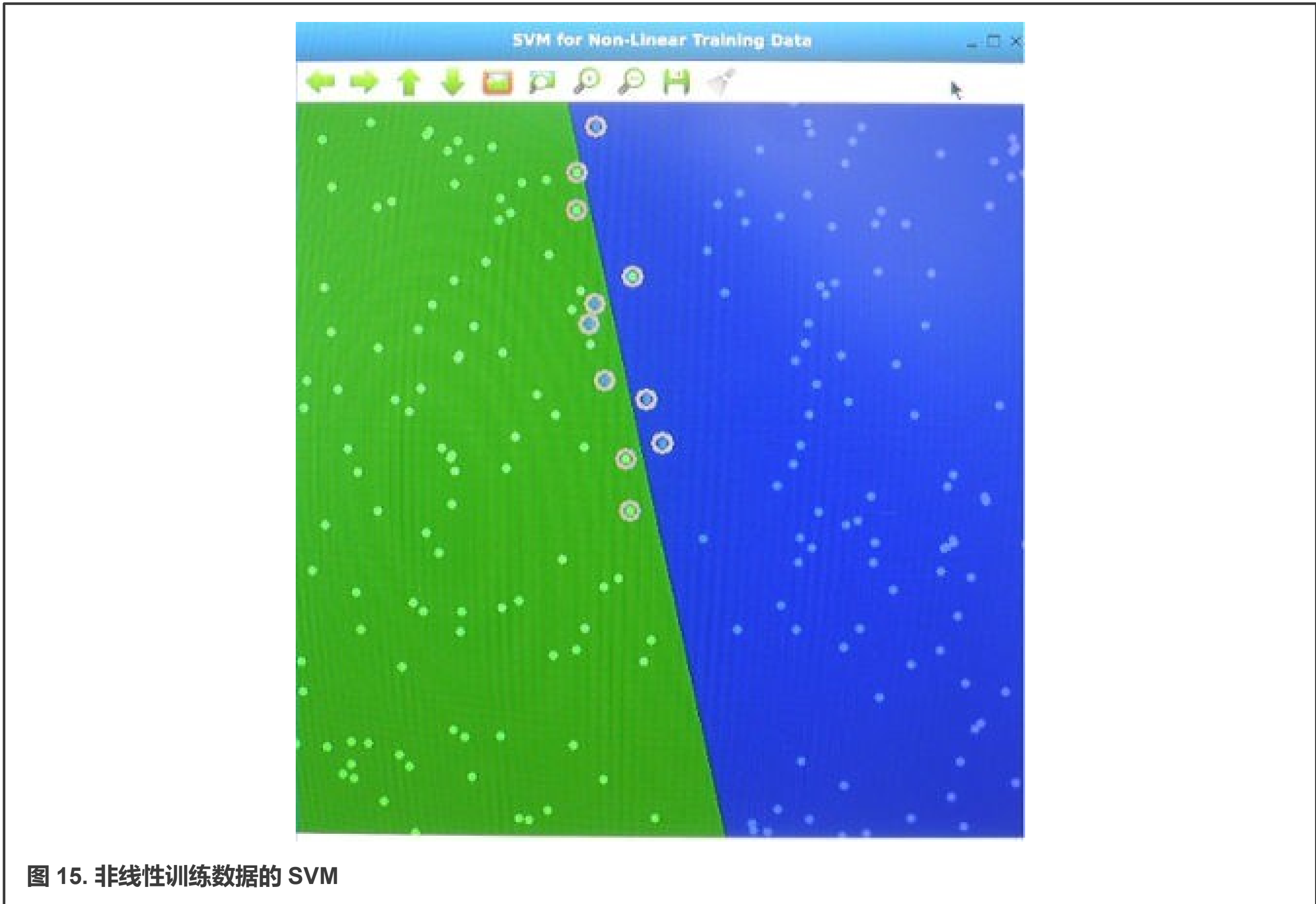

图 15. 非线性训练数据的 SVM

8.3.3 主成分分析（PCA）介绍

主成分分析（PCA）是一种提取数据集最重要特征的统计方法。本节介绍如何使用 PCA 计算物体的方向。如需了解更多详细信息，请查看 OpenCV 教程“PCA 入门 ”（Introduction_to_PCA）。

运行演示后，屏幕上会显示图形结果（需要 Qt5 支持）：

```
$ ./example_tutorial_introduction_to_pca ../data/pca_test1.jpg
```

结果如下所示：

- 打开一个图像（从`../data/pca_test1.jpg` 加载）。
- 寻找检测到的物体的方向。
- 通过绘制检测到的物体的轮廓、中心点以及提取方向的 x 轴、y 轴，展示结果。

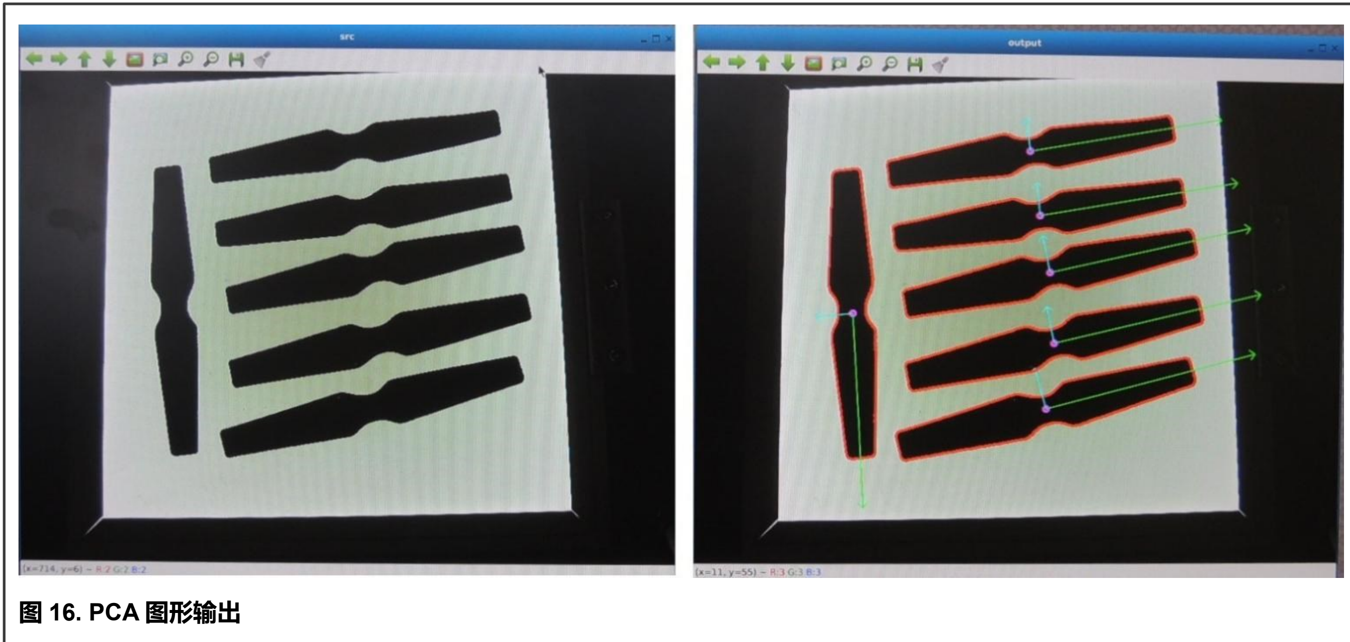

图 16. PCA 图形输出

8.3.4 逻辑回归

在此示例中，逻辑回归用于预测图像中的两个字符（0 或 1）。首先，每个图像矩阵都从其原始大小 28x28 重塑为 1x784。使用 20 张图像创建和训练逻辑回归模型。训练后，模型可以预测测试图像的标签。源代码位于 logistic_regression 链接上，可键入以下命令来运行源代码。

演示依赖项（准备训练数据文件）：

```
$ wget https://raw.githubusercontent.com/opencv/opencv/4.5.2/samples/data/data01.xml
```

运行演示后，屏幕上会显示图形结果（需要 Qt5 支持）：

```
$ ./example_cpp_logistic_regression
```

结果如下所示：

- 显示训练和测试数据
- 显示原始标签和预测标签之间的比较。

控制台文本输出如下所示（经过训练的模型准确率达到 95%）：

```
original vs predicted:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1]
accuracy: 95%
saving the classifier to NewLR_Trained.xml
loading a new classifier from NewLR_Trained.xml
predicting the dataset using the loaded classifier...done!
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1]
accuracy: 95%
```

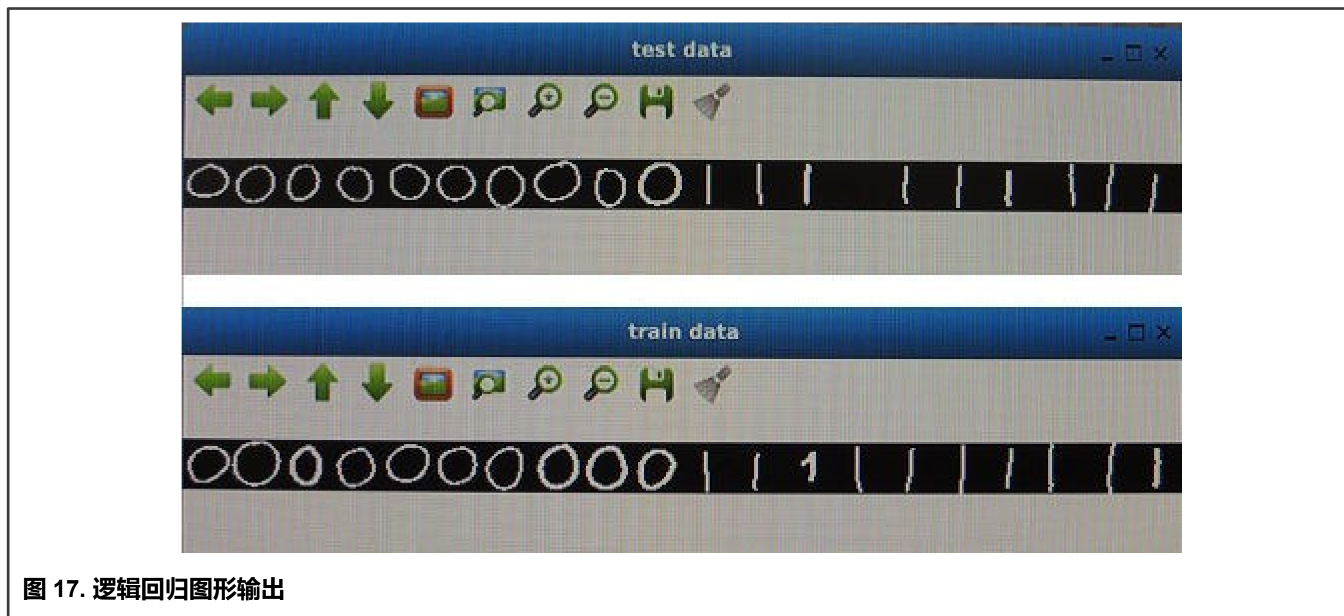

图 17. 逻辑回归图形输出

第 9 章 DeepViewRT

DeepViewRT 是针对恩智浦微处理器和微控制器进行了优化的专有神经网络推理引擎，它不仅实施自己的计算引擎，而且还能够利用主流第三方引擎。

特性：

- 支持各种计算引擎的插件 API：
  - DeepViewRT (CPU/Neon)
  - DeepViewRT (OpenVX)
  - TensorFlow Lite
  - Arm NN
  - ONNX Runtime
- C 和 Python API
- 支持按张量和按通道量化模型
- 为现有操作定义自定义操作或自定义行为
- 无需显式编程计算图即可将模型部署到所有目标

9.1 DeepViewRT 软件栈

DeepViewRT 软件栈包括 DeepViewRT 库、modelrunner 库和 modelrunner 服务器 – 请参见下图：

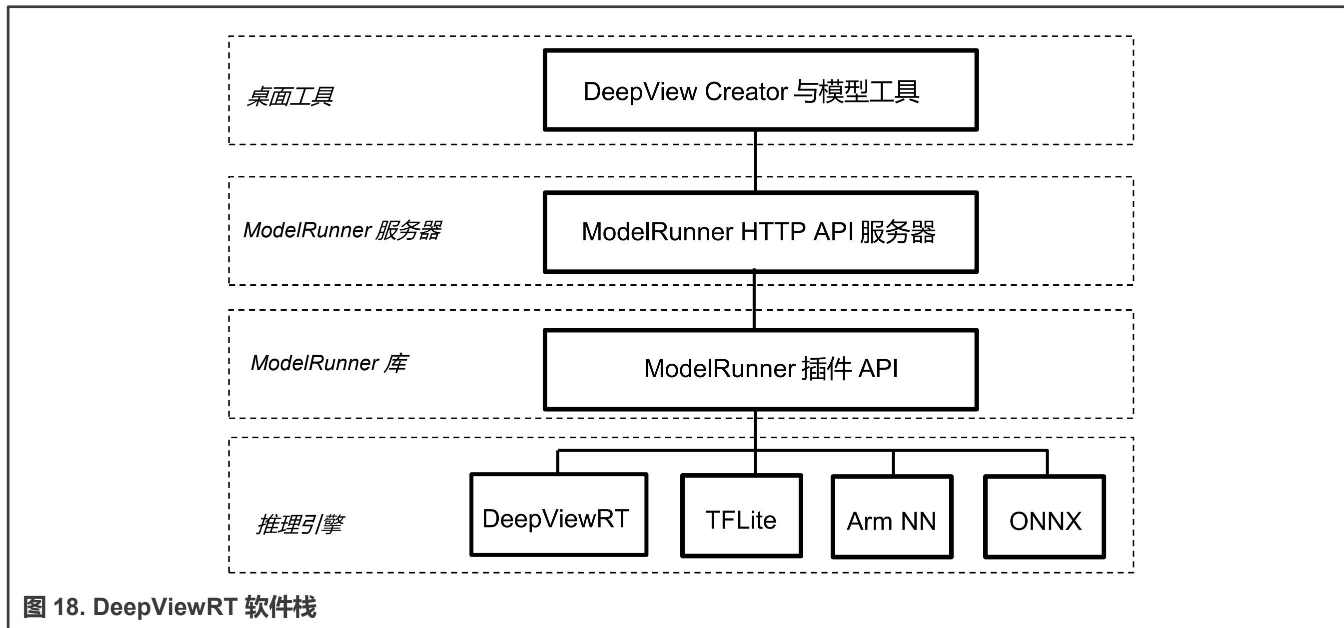

图 18. DeepViewRT 软件栈

注意

DeepView Creator 和模型工具是 eIQ 工具包的一部分。

DeepViewRT 支持以下硬件：

- CPU Arm Cortex-A 内核
- 使用 VSI NPU 后端的 GPU/NPU 硬件加速器，它可以在 GPU 和 NPU 上运行，取决于哪个可用

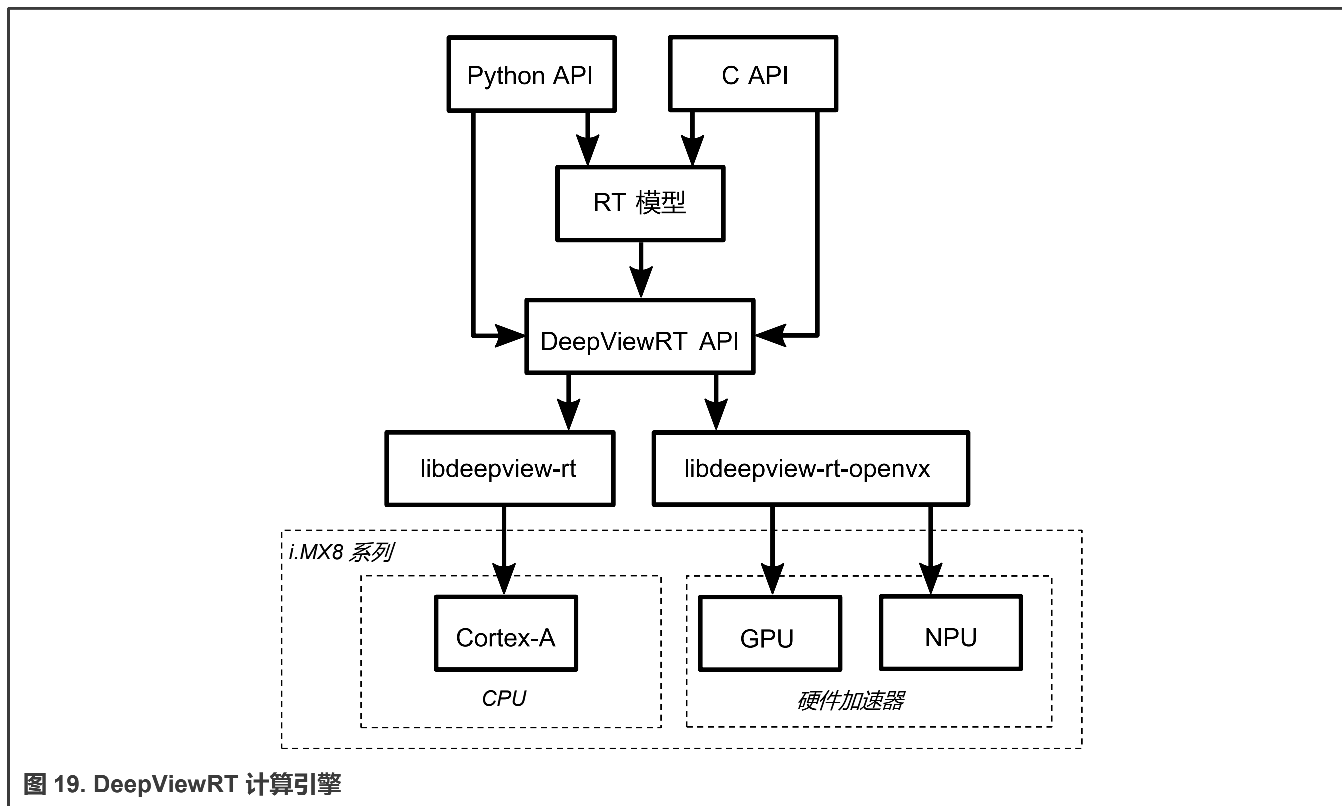

图 19. DeepViewRT 计算引擎

注意

如需了解 DeepViewRT API 的更多信息，请参见《*DeepViewRT 用户手册*》。

9.2 交付包

DeepViewRT 在 Yocto recipe 中可用，能够通过 DeepViewRT recipe 获得 DeepViewRT 包。

DeepViewRT 软件包包括 Yocto BSP 版本的以下组件：

- DeepViewRT 共享库（动态库）
- DeepViewRT 头文件
- DeepViewRT Python 模块
- ModelRunner 二进制文件和库
- ModelRunner 插件库（OpenVX、TensorFlow Lite、Arm NN、ONNX Runtime）
- DeepViewRT 示例（labelimg、detectimg、ssdcam-gst、labelcam-gst）

9.3 示例应用程序

所有示例应用程序都集成到 Yocto BSP 镜像中。您可以使用此 Yocto 命令提取源代码并构建所有示例：

```
bitbake -c patch deepview-rt-examples
```

deepview-rt-examples 源代码放在该目录下：`<Yocto_install_dir>/<build_project_dir>/tmp/work/cortexa53- crypto-mx8mp-poky-linux/deepview-rt-examples/1.1-r0/deepview-rt-examples-1.1/`。

文件夹结构如下所示：

图 20. DeepViewRT Yocto 文件夹结构

要交叉编译这些示例，请使用示例源文件夹下的 Makefile。

注意

所有示例都使用 DeepViewRT RTM 模型格式。可以从 *.tflite* 转换成 *.rtm*。如需了解模型转换，请参见《*eIQ 工具包用户指南*》（EIQTUG）。

9.3.1 图像标记应用

有两个示例应用程序演示了如何实施图像标记应用程序，目标是 direct DeepViewRT C API 或使用 libCurl 库的 ModelRunner REST API。

标记（“labelimg”）应用程序直接调用 DeepViewRT C API：

```
$ cd /usr/bin/deepview-rt-examples
$ ./labelimg mobilenet_v1_0.25_224_quant.rtm eagle.png
```

“labelimg_remote”应用程序通过 libCurl 库使用 ModelRunner REST API。运行它需要两个带有以下命令的终端：

```
# Terminal 1: use -e rt -c 1 (for NPU) or -e rt -c 0 (for CPU)
$ modelrunner -e rt -c 1 -H 10818 -m mobilenet_v1_0.25_224_quant.rtm
```

```
# Terminal 2:
$ ./labelimg_remote mobilenet_v1_0.25_224_quant.rtm eagle.png
```

9.3.2 目标检测应用程序

有两个示例应用程序演示了如何实施目标检测应用程序，针对 direct DeepViewRT C API 或使用 libCurl 库的 ModelRunner REST API。

“detectimg”应用程序直接调用 DeepViewRT C API：

```
$ cd /usr/bin/deepview-rt-examples
$ ./detectv4  DATA_PATH//mobilenet_ssd_v1_1.00_trimmed_new.rtm  DATA_PATH/ssd_resized.jpg -T 0.5 -I
0.5 -i 50 -e /usr/lib/deepview-rt-openvx.so
```

“detectimg_remote”应用程序通过 libCurl 库使用 ModelRunner REST API。运行它需要两个带有以下命令的终端：

```
# Terminal 1: use -e rt -c 1 (for NPU) or -e rt -c 0 (for CPU)
$ modelrunner -e rt -c 1 -H 10818 -m mobilenet_ssd_v1_1.00_trimmed_quant_anchors.rtm
```

```
# Terminal 2:
$ ./detectv4_remote -p 10896 -m mobilenet_ssd_v1_1.00_trimmed_quant_anchors.rtm -i horse.jpg -A
10.10.40.190 -t 0.6 -n 50 -r 0
```

9.3.3 Labelcam-gst 示例应用程序

此示例演示了一个基于 GStreamer 的应用程序，该应用程序提供一个摄像头到显示器 pipeline，并拆分到与 DeepViewRT 对接的 appsink。推理结果在视频显示器上显示为文本覆盖。

该示例可以通过 libCurl 库（通过 OpenVX 插件利用 NPU 加速）支持使用 DeepViewRT API（CPU）和 ModelRunner REST API 运行。该示例需要摄像头和显示器；可以采用 MIPI-CSI 摄像头或 USB 摄像头。如需了解如何使用 MIPI-CSI 摄像头和显示器，请参见《*i.MX 移植指南*》（IMXBSPG）。

假设用户有一个名为 *mobilenet_v1_0_1.0_224_quant_with_labels.rtm* 的模型并使用 USB 摄像头（*/dev/video3*）和 LCD，可以通过 DeepViewRT API（CPU）按如下方式执行演示。

```
$ ./labelcam-gst -m mobilenet_v1_0_1.0_224_quant_with_labels.rtm -c /dev/video3
IP not set! Streaming to localhost!!
video size: 640x480 center roi size: 480x480 model size: 224x224
```

LCD 将显示带有 possibility 值和 runtime 值的标签名称。

该演示还可以通过 ModelRunner REST API 使用 libCurl 库执行，如下所示。利用 NPU 加速：

```
# Terminal 1: use -e rt -c 1 (for NPU) or -e rt -c 0 (for CPU)
$ modelrunner -e rt -c 1 -H 10818 -m mobilenet_v1_0_1.0_224_quant_with_labels.rtm
```

```
# Terminal 2:
$ ./labelcam-gst -m mobilenet_v1_0_1.0_224_quant_with_labels.rtm -c /dev/video3 -r 127.0.0.1 -p 10818
```

```
-u 1
POST URL = http://127.0.0.1:10818/v1?run=1&output=MobilenetV1_Predictions_Reshape_1
IP not set! Streaming to localhost!!
video size: 640x480 center roi size: 480x480 model size: 224x224
```

LCD 将显示带有 possibility 值、往返时间和推理时间的标签名称。

9.3.4 Ssdcam-gst 示例应用

该项目演示了如何将 DeepViewRT 与 GStreamer 摄像头 pipeline 进行集成。在本示例中，我们从默认摄像头捕获输入，然后运行单镜头检测，为帧中每个检测到的物体生成边界框、标签和概率。

该示例可以通过 libCurl 库（通过 OpenVX 插件利用 NPU 加速）支持使用 DeepViewRT API（CPU）和 ModelRunner REST API 运行。该示例需要摄像头和显示器；可以采用 MIPI-CSI 摄像头或 USB 摄像头。如需了解如何使用 MIPI-CSI 摄像头和显示器，请参见《*i.MX 移植指南*》（IMXBSPG）。

假设您有名为 *mobilenet_v1_0_1.0_224_quant_with_labels.rtm*, *mobilenet_ssd_v1_1.00_trimmed_anchors_quant.rtm* 的模型并使用 USB 摄像头（*/dev/video3*）和 LCD，则可以通过 DeepViewRT API（CPU）执行演示，如下所示。

```
$ ./ssdcam-gst -m mobilenet_ssd_v1_1.00_trimmed_anchors_quant.rtm -c /dev/video3 -t 0.5 -n 0.5 Score
Threshold used = 0.50 video size: 640x480 model size: 300x300 Using display!
```

LCD 将显示推理时间、绘制的物体边界框、物体类名和概率。

还可以通过 ModelRunner REST API 使用 libCurl 库按如下方式执行演示，这将利用 NPU 进行加速：

```
# Terminal 1: use -e rt -c 1 (for NPU) or -e rt -c 0 (for CPU)
$ modelrunner -e rt -c 1 -H 10818 -m mobilenet_v1_0_1.0_224_quant_with_labels.rtm
```

```
# Terminal 2:
$ ./ssdcam-gst -m mobilenet_ssd_v1_1.00_trimmed_anchors_quant.rtm -c /dev/video3 -t 0.5 -n 0.5 -r
127.0.0.1 -p 10818 Score Threshold used = 0.50 video size: 640x480 model size: 300x300 Using display!
```

LCD 将显示推理时间、往返时间，绘制的物体边界框、物体类名和概率。

9.4 ModelRunner

ModelRunner 应用程序提供 HTTP 服务，用于对接 DeepViewRT 模型、TensorFlow Lite 模型、ONNX Runtime 模型和远程评估。该服务还提供用于低延迟视频处理的低级 UNIX 套接字服务。它通过 DeepViewRT Yocto recipe 集成到 BSP 中。

如需了解 ModelRunner HTTP REST API，请参见《*DeepViewRT 用户手册*》。

要使用 Modelrunner 进行基准评估，请参考以下命令（章节）来衡量性能。

9.4.1 DeepViewRT

要使用 DeepViewRT 后端运行 modelrunner 并测量其性能：

```
$ modelrunner -e rt -c 0 -m mobilenet_v1_1.0_224_quant.rtm -b 50 -t 4
Plugin: libmodelrunner-rt.so;
Average model run time: 129.0078 ms (layer sum: 0.0000 ms)
```

注意

线程数（-t 参数）应与器件计算核心数相关联，以获得最佳性能。例如，对于 i.MX 8QM 芯片，请使用-t 6。

9.4.2 OpenVX

要使用 OpenVX 运行 modelrunner，请使用 NPU 加速并测量其性能：

```
$ modelrunner -e rt -c 1 -m mobilenet_v1_1.0_224_quant.rtm -b 50
Plugin: libmodelrunner-ovx.so;
RTMx Output indices = [87 ]
Created empty VX graph, inputs = 1, outputs = 1
RTMx Layer count = 88
…
Average model run time: 2.2397 ms
```

9.4.3 TensorFlow Lite

使用 TensorFlow Lite 和 NNAPI delegate 运行 modelrunner 并测量其性能：

```
$ modelrunner -e tflite -c 1 -m mobilenet_v1_1.0_224_quant.tflite -b 50
Plugin: libmodelrunner-tflite.so;
Loaded model
resolved reporter
INFO: Created TensorFlow Lite delegate for NNAPI.
Applied NPU delegate.
interpreter invoked
average time: 2.51356 ms
Average layer sum: 2.5105 ms
```

注意

将 “-c 1” 替换为 “-c 0”，可改为使用 CPU。对 XNNPACK 使用 “-c 2”，对 VX Delegate 使用 “-c 3”。

9.4.4 Arm NN

使用 Arm NN 和 Vsi_Npu 后端运行 modelrunner 并测量其性能：

```
$ modelrunner -e armnn -c 3 -m mobilenet_v1_1.0_224_quant.tflite -b 50 -t 4
Plugin: libmodelrunner-armnn.so;
NPU backend preference
Model loaded and validated, size = 150528
…
Inference Time in ms = 2.56184
```

注意

将 “-c 3” 替换为 “-c 0”，可改为使用 CpuAcc。

9.4.5 ONNX Runtime

使用 ONNX Runtime 和 Vsi_Npu 运行 modelrunner 并测量其性能：

```
$ modelrunner -e onnx -c 3 -m mobilenet_v1_1.0_224_quant.onnx -b 50
Plugin: libmodelrunner-onnx.so;
WARNING: Since openmp is enabled in this build, this API cannot be used to configure intra op num
threads. Please use the openmp environment variables to control the number of threads.
```

```
Prefer Vsi_Npu execution provider
Input name=input, type=1, num_dims=4, shape=[ 1 3 224 224 ]
Number of outputs = 1
Output 0 : name=TFLITE2ONNX_Quant_MobilenetV1/Predictions/Reshape_1_dequantized
Loaded ONNX model.
Average model run time: 434.220155 ms
```

使用 ONNX Runtime 和 Arm NN 运行 modelrunner 并测量其性能：

```
$ modelrunner -e onnx -c 2 -m mobilenet_v1_1.0_224_quant.onnx -b 50 -t 4
Plugin: libmodelrunner-onnx.so;
WARNING: Since openmp is enabled in this build, this API cannot be used to configure intra op num
threads. Please use the openmp environment variables to control the number of threads.
Prefer ArmNN execution provider
Input name=input, type=1, num_dims=4, shape=[ 1 3 224 224 ]
Number of outputs = 1
Output 0 : name=TFLITE2ONNX_Quant_MobilenetV1/Predictions/Reshape_1_dequantized
Loaded ONNX model.
Average model run time: 233.127588 ms
```

注意

将 “-c 3” 替换为 “-c 2”，可改为使用 “ArmNN” 作为执行提供程序。

第 10 章
TVM

ApacheTVM 是一个面向 CPU、GPU 和机器学习加速器的开源机器学习编译器框架，目的是使机器学习工程师能够在任何硬件后端高效地优化和运行计算。

特性：

- TVM 0.7.0
- 将深度学习模型编译成最小可部署模块
- 在更多后端自动生成和优化模型的基础架构，性能更佳
- GPU/NPU 支持带有 OpenVX 库的 i.MX8（i.MX8MM 和 i.MX8MN 除外）平台
- TVM 构建程序（builder）支持 Ubuntu 18.04、x86_64 平台

注意

如需了解更多详细信息，请参见 TVM 文档。

10.1 TVM 软件工作流程

预训练的模型将转换为中继 IR，并传递到 TVM 模型优化，如常量折叠、存储器规划，最后传递到代码生成（codegen）阶段。在此阶段，目标器件支持的算子作为内在调用转换到分流库，分流库连接 GPU/NPU 等模型加速器装置。

图 21. TVM 软件工作流程

10.2 开始

10.2.1 通过 RPC 验证运行示例

TVM 提供远程过程调用（RPC）功能，以便在远程设备上运行模型。

用户可以通过 RPC 验证在 `tests/python/contrib/test_vsi_npu` 上运行示例。根据主机上相同输入的结果对设备上的模型运行结果进行验证。

- 在设备上启动 RPC 服务器

```
$ python3 -m tvm.exec.rpc_server --host 0.0.0.0 --port=9090
```

- 导出系统变量：

```
$ export TVM_HOME=/path/to/tvm
$ export PYTHONPATH=$TVM_HOME/python
```

- 在主 PC 上运行指定模型：

```
$ python3 tests/python/contrib/test_vsi_npu/test_tflite_models.py -i {device_ip} -
m mobilenet_v2_1.0_224_quant
```

- 在主 PC 上运行所有受支持的 TensorFlow Lite 模型：

```
$ python3 tests/python/contrib/test_vsi_npu/test_tflite_models.py -i {device_ip}
```

注意

该测试将自动下载模型，请确保网络可以访问公网。示例脚本可能会导入其他 Python 库。请检查脚本，确保已正确安装。

要测试 `pytorch/onnx/keras` 模型，需要在主 PC 上安装其他 python 软件包：

```
$ python3 -m pip install torch==1.7.0 torchvision==0.8.1
$ python3 -m pip install onnx=1.8.1 onnxruntime==1.8.1
$ python3 -m pip install tensorflow==2.5.0
```

10.2.2 在设备上单独运行示例

在此模式下，模型在主机上脱机编译并另存为 model.so。请参考 `tests/python/contrib/test_vsi_npu/compile_tflite_models.py`，在主机上编译 TensorFlow Lite 模型。

下面的脚本片段展示了如何在设备上加载和运行已编译的模型：

```
ctx = tvm.cpu(0)
# load the compiled model
lib = tvm.runtime.load_module(args.model)
m = graph_runtime.GraphModule(lib["default"](ctx))
# set inputs
data = get_img_data(args.image, (args.input_size, args.input_size), args.data_type)
m.set_input(args.input_tensor, data)
# execute the model
m.run()
# get outputs
tvm_output = m.get_output(0)
```

请参考 `tests/python/contrib/test_vsi_npu/label_image.py`，获得完整的标签图像示例，其中包含图像解码的预处理和后处理，以生成标签。

10.3 如何在主机上构建 TVM 堆栈

从概念上来看，TVM 可分为两部分：

- TVM build 堆栈：在主机上编译深度学习模型
- TVM runtime：在芯片上加载和解析模型

该 build 堆栈使用 LLVM 交叉编译生成的源代码，作为芯片的可部署动态库。请按照《LLVM 文档》所述在主机上安装 LLVM。如果安装成功，`/usr/bin` 下应有 llvm-config。

要构建 tvm，请确保主机上安装了以下依赖包：

- cmake
- python3-dev
- build-essential
- llvm-dev
- g++-aarch64-linux-gnu
- libedit-dev
- libxml2-dev
- python3-numpy
- python3-attrs
- python3-tflite

对于 Ubuntu 18.04，用户可以使用下面所示的命令安装所有依赖项：

```
$ sudo apt-get update
$ sudo apt-get install -y python3 python3-dev python3-setuptools
$ sudo apt-get install -y cmake llvm llvm-dev g++-aarch64-linux-gnu gcc-aarch64-linux-gnu
$ sudo apt-get install -y libtinfo-dev zlib1g-dev build-essential libedit-dev libxml2-dev
$ python3 -m pip install numpy decorator scipy attrs six tflite
```

按照以下说明在主机上构建 TVM 堆栈：

```
$ export TOP_DIR=`pwd`
$ git clone --recursive https://source.codeaurora.org/external/imx/eiq-tvm-imx/ tvm-host
$ cd tvm-host
$ mkdir build
$ cp cmake/config.cmake build
$ cd build
$ sed -i 's/USE_LLVM\ OFF/USE_LLVM\ \/usr\/bin\/llvm-config/' config.cmake
$ cmake ..
$ make tvm -j4 # make tvm build stack
```

10.4 支持的模型

以下模型通过 TVM 进行了验证。

表 3. TVM 模型 ZOO

模型	float32	int8	输入大小
mobilenet_v1_0.25_128	mobilenet_v1_0.25_128	mobilenet_v1_0.25_128_quan t	128
mobilenet_v1_0.25_224	mobilenet_v1_0.25_224	mobilenet_v1_0.25_224_quan t	224

下页继续......

表 3. TVM 模型 ZOO（续）

模型	float32	int8	输入尺寸
mobilenet_v1_0.5_128	mobilenet_v1_0.5_128	mobilenet_v1_0.5_128_quant	128
mobilenet_v1_0.5_224	mobilenet_v1_0.5_224	mobilenet_v1_0.5_224_quant	224
mobilenet_v1_0.75_128	mobilenet_v1_0.75_128	mobilenet_v1_0.75_128_quan t	128
mobilenet_v1_0.75_224	mobilenet_v1_0.75_224	mobilenet_v1_0.75_224_quan t	224
mobilenet_v1_1.0_128	mobilenet_v1_1.0_128	mobilenet_v1_1.0_128_quant	128
mobilenet_v1_1.0_224	mobilenet_v1_1.0_224	mobilenet_v1_1.0_224_quant	224
mobilenet_v2_1.0_224	mobilenet_v2_1.0_224	mobilenet_v2_1.0_224_quant	224
inception_v1	N/A	inception_v1_224_quant	224
inception_v2	N/A	inception_v2_224_quant	224
inception_v3	inception_v3	inception_v3_quant	299
inception_v4	inception_v4	inception_v4_299_quant	299
deeplab_v3_257_mv_gpu	deeplab_v3_256_mv_gpu	N/A	257
deeplab_v3_mnv2_pascal	N/A	deeplab_v3_mnv2_pascal	513
ssdlite_mobiledet	ssdlite_mobiledet_cpu_320x3 20_coco	N/A	320

第 11 章
在硬件加速器上的 NN 执行

11.1 硬件加速器介绍

i.MX8 类芯片部署了两种 NN 加速器：

- 神经处理单元（NPU）
- 图形处理单元（GPU）

神经处理单元针对定点运算进行了优化，采用 8 位和 16 位宽度。为了在 NPU 上获得最佳性能，应使用量化模型。

图形处理单元针对定点运算和半精度浮点运算进行了优化。为了在 GPU 上获得最佳性能，应使用半精度的量化模型或浮点模型。

注意

TensorFlow Lite 框架可以直接用 16 位半精度运算计算浮点模型。

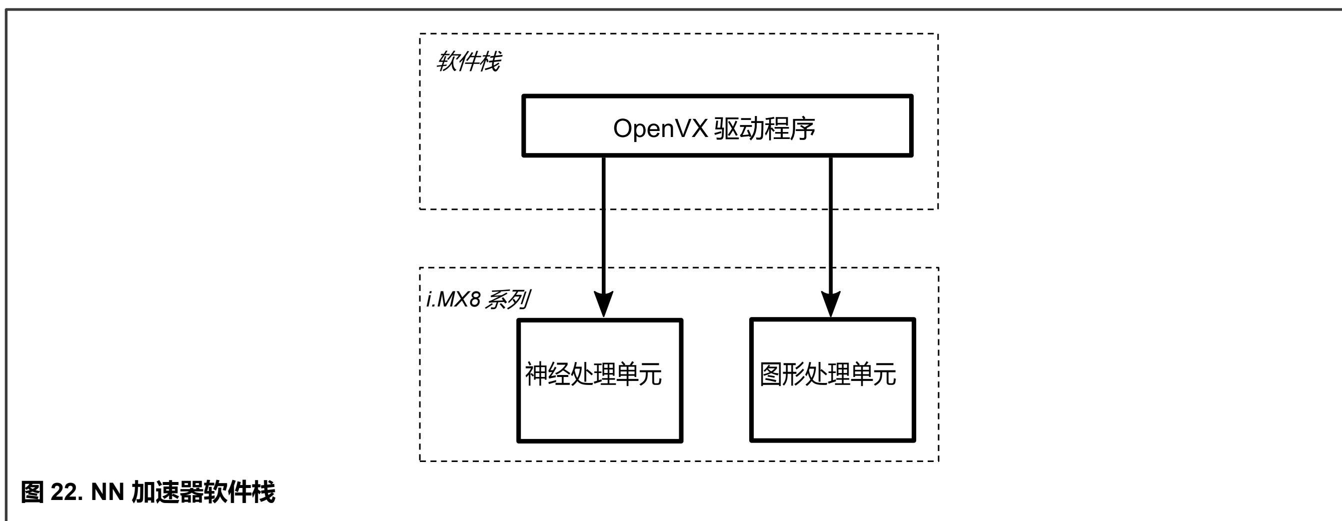

图 22. NN 加速器软件栈

通过带 NN 扩展的 OpenVX v1.2 提供对接 NPU/GPU 硬件加速器的接口。OpenVX 是一个开放的、免版税的计算机视觉应用程序跨平台加速标准。它提供以下内容[3]：

- 预定义和可定制的视觉功能库
- 基于图形的执行模型，结合了支持任务和数据独立执行的功能
- 一组抽象物理存储器的存储器对象

OpenVX 定义了一个 C 应用程序编程接口，用于构建、验证和协调图形执行以及访问存储器对象。如需了解 OpenVX 的更多信息，请访问 OpenVX 主页。

注意

在当前的 OpenVX 驱动程序实施中，OpenVX 图形支持的最大节点数为 2048。

11.2 硬件加速器上的分析

本节介绍如何在 GPU/NPU 上启用 profiler（分析器），以及如何捕获日志。

[3] OpenVX 1.2 规范；https://www.khronos.org/registry/OpenVX/specs/1.2/html/index.html

1. 在 U-Boot 中按 Enter 键停止 EVK 板。
2. 通过添加 `galcore.showArgs=1` 和`galcore.gpuProfiler=1`，更新 mmcargs。

```
u-boot=> editenv mmcargs
edit: setenv bootargs ${jh_clk} console=${console} root=${mmcroot}
galcore.showArgs=1 galcore.gpuProfiler=1
u-boot=> boot
```

3. 启动开发板并等待 Linux OS 提示。
4. 在执行应用程序之前，应启用以下环境变量。在分析过程中，`VIV_VX_DEBUG_LEVEL` 和 `VIV_VX_PROFILE` 标志应始终为 1。`CNN_PERF` 环境变量使驱动程序能够生成每层配置文件日志。`NN_EXT_SHOW_PERF` 展示编译器如何估算性能并据此确定平铺（tiling）的详细信息。

```
export CNN_PERF=1 NN_EXT_SHOW_PERF=1 VIV_VX_DEBUG_LEVEL=1 VIV_VX_PROFILE=1
```

5. 得到分析结果的 log 文件。我们使用标准恩智浦 Linux 版本的示例 ML 示例部分来解释以下部分。
   - TensorFlow Lite 分析

     使用 GPU/NPU 后端运行 TensorFlow Lite 应用程序，如下所示：

     ```
     $ cd /usr/bin/tensorflow-lite-2.6.0/examples
     $ ./label_image -m mobilenet_v1_1.0_224_quant.tflite -t 1 -i grace_hopper.bmp -l labels.txt
     --external_delegate_path=/usr/lib/libvx_delegate.so -v 0 > viv_test_app_profile.log 2>&1
     ```

   - Arm NN 分析

     使用 GPU/NPU 后端运行 Arm NN 应用程序（这里以 TfMobilNet 为例），如下所示：

     ```
     $ cd /usr/bin/armnn-21.08/
     $ ./TfMobileNet-Armnn --data-dir=data --model-dir=models --compute=VsiNpu >
     viv_test_app_profile.log 2>&1
     ```

注意

Armnn 分析示例假定模型文件和输入数据都位于相应的子文件夹中。另请参见“运行 Arm NN 测试”章节。

该日志捕获了每一层执行时钟周期和 DDR 数据传输的详细信息。

注意

随着分析器开销的增加，推理的平均时间可能比平时长。

11.3 硬件加速器预热时间

对于 Arm NN 和 TensorFlow Lite，由于 GPU/NPU 硬件加速器需要初始化模型图，初始执行模型推理需要更长的时间。初始化阶段称为预热。通过将初始 OpenVX 图形处理产生的信息存储在磁盘上，可以减少后续应用程序运行的持续时间。为此，应使用以下环境变量：

`VIV_VX_ENABLE_CACHE_GRAPH_BINARY`：启用/禁用 OpenVX 图形缓存的标志

`VIV_VX_CACHE_BINARY_GRAPH_DIR`：设置缓存信息在磁盘上的位置

例如，按以下方式在控制台上设置这些变量：

```
export VIV_VX_ENABLE_CACHE_GRAPH_BINARY="1"
export VIV_VX_CACHE_BINARY_GRAPH_DIR=`pwd`
```

通过设置这些变量，OpenVX 图形编译的结果将以网络二进制图形文件（*.nb）的形式存储在磁盘上。runtime 在网络上执行快速哈希检查，如果它与*.nb 文件哈希匹配，则将其直接加载到 NPU 存储器中。这些环境变量需要持久设置，例如，重启后可用。否则，即使*.nb 文件可用，也会绕过缓存机制。

图形初始化之后的迭代执行速度要快很多倍。在评估在 GPU/NPU 上运行的应用程序的性能时，应分别测量预热和推理的时间。预热时间通常只影响第一次推理运行。然而，根据机器学习模型类型，在前几次推理运行中可能会很明显。必须进行一些初步测试才能决定如何考虑预热时间。在划分好这个阶段后，后续的推理运行可被视为纯推理，并用于计算推理阶段的平均值。

11.4 GPU 和 NPU 之间的切换

一些平台同时部署了 3D GPU 和 NPU 硬件加速器。两者都可用于执行 OpenVX 图形（即用于 ML 推理）。有一个环境变量 USE_GPU_INFERENCE 可用于区分 GPU 和 NPU。可以从硬件加速驱动程序中读取该变量值。

其行为如下所示：

- 如果 USE_GPU_INFERENCE=1，则图形在 GPU 上执行
- 否则，图形在 NPU 上执行（如可用）

默认情况下，NPU 用于 OpenVX 图形执行。

TensorFlow Lite 的示例如下所示：

```
$ USE_GPU_INFERENCE=1 ./label_image -m mobilenet_v1_1.0_224_quant.tflite -i grace_hopper.bmp -l
labels.txt --external_delegate_path=/usr/lib/libvx_delegate.so
```

第 12 章
eIQ 演示

以下章节演示了如何使用恩智浦 eIQ 框架的 3 个列子。

12.1 GStreamer

GStreamer 是一个基于 pipeline 的多媒体框架，将各种媒体处理系统链接在一起，以完成复杂的工作流程。GStreamer 框架的许多优点源于其模块化；GStreamer 可以无缝地整合新的插件模块。该软件基于一个新的 GStreamer 插件模块，即用于恩智浦 i.MX 处理器的神经网络推理。目前，它支持目标检测和姿势估计示例。

特性：

- TensorFlow Lite 推理和神经网络 API delegate
- 面向 i.MX8 平台的 GPU/NPU 硬件加速
- OpenCV 推理结果形状绘图

12.1.1 Gstreamer 软件工作流程

当 Gstreamer 执行特定任务时，需要通过相应的命令创建 pipeline。pipeline 由链接在一起的元素组成，让数据流经这个元素链。一个元素有一个特定的功能，例如从文件读取数据、解码该数据或将该数据输出到显卡。下图展示了 eIQ 演示软件工作流程：

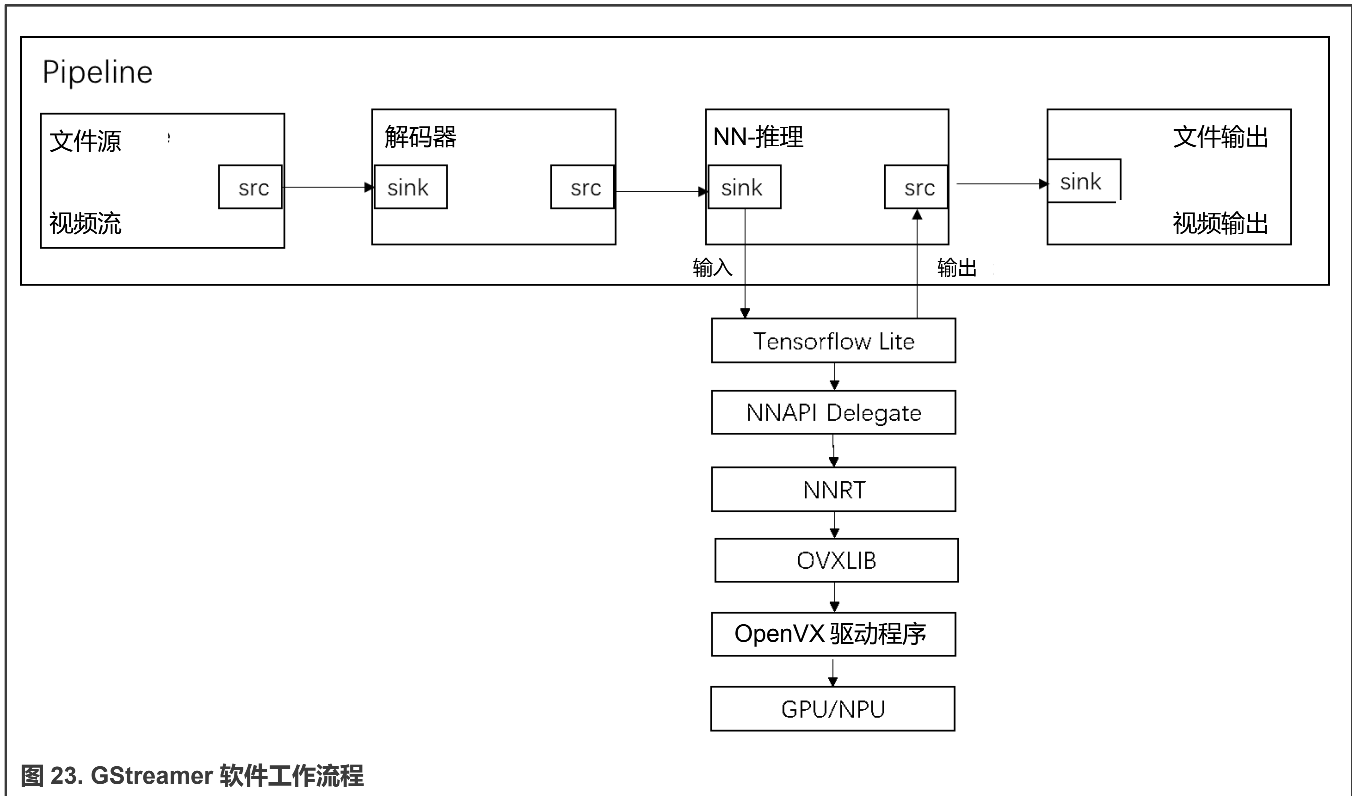

图 23. GStreamer 软件工作流程

视频文件或摄像头输入用作 pipeline 的源。解码帧通过解码器块生成。然后这些帧被转换为 RGB 数据，并设置为 TensorFlow Lite 解析器的输入张量。推理是基于 NNAPI delegate、NNRT、OVXLIB、OpenVX 驱动程序和硬件加速 GPU/NPU 实现的。推理结果形状（如目标检测矩形和包含关键点列表的姿势对象）由 OpenCV 绘制。还将显示平均推理时间、当前时间和每秒推理帧数。

12.1.2 开始

首先，下载相关模型并将其复制到设备上的目录，如下所示：

```
$ wget https://github.com/google-coral/project-posenet/raw/master/models/mobilenet/posenet_mobilenet_v1_075_353_481_quant_decoder.tflite
$ cp posenet_mobilenet_v1_075_353_481_quant_decoder.tflite {rootfs}/usr/share/gstnninferencedemo/google-coral/project-posenet/
$ wget https://dl.google.com/coral/canned_models/all_models.tar.gz
$ tar -xvzf all_models.tar.gz
$ cp mobilenet_ssd_v2_coco_quant_postprocess.tflite {rootfs}/usr/share/gstnninferencedemo/google-coral/examples-camera/
```

然后，可以运行以下示例，这些示例已经安装在 Yocto rootfs 中。

注意

如需了解源代码演示位置，请参见 eiq-apps-imx 和 coral-posenet-imx 存储库。

12.1.2.1 使用视频流进行目标检测

下面是一个使用视频流进行目标检测的示例。建议使用 720p30 视频：

```
$ /usr/bin/gstnninferencedemo-mobilenet-ssd-video </path/to/video_file>
```

12.1.2.2 使用摄像头流进行目标检测

下面是一个使用摄像头流进行目标检测的示例。MIPI-CSI 摄像头或 USB 摄像头均可使用。摄像头设备名称为 *<dev/video?>*：

```
$ /usr/bin/gstnninferencedemo-mobilenet-ssd-camera </dev/video?>
```

12.1.2.3 使用视频流运行姿势估计

下面是一个使用视频流运行姿势估计的示例。建议使用 720p30 视频：

```
$ /usr/bin/gstnninferencedemo-posenet-video </path/to/video_file>
```

12.1.2.4 使用摄像头流运行姿势估计

下面是一个使用摄影头流运行姿势估计的示例。MIPI-CSI 摄像头或 USB 摄像头均可使用。摄像头设备名称为*<dev/video?>*：

```
$ /usr/bin/gstnninferencedemo-posenet-camera </dev/video?>
```

注意

选择摄像头当前连接的端口。使用 `v4l2-ctl --list-devices` 命令进行检查。

12.1.2.5 Pipeline 演示命令

对于上述示例，可以使用 shell 脚本来运行演示。每个 shell 脚本中都有一个相应的 GStreamer 命令，以及几个可以为 pipeline 更改的变量。以上面的“使用视频流运行姿势估计”为例，完整的命令 pipeline 如下所示：

```
GST_COMMAND="gst-launch-1.0 -v filesrc location=${VIDEO_FILE} ! decodebin ! queue max-size-
time=0 ! nninferencedemo rotation=${ROT} demo-mode=${DEMO_MODE} model=${MODEL} label=${LABEL}
use-nnapi=${USE_NNAPI} num-threads=${NUM_THREADS} display-stats=${DISPLAY_STATS} enable-inference=$
{ENABLE_INFERENCE} ! waylandsink sync=${SYNC}"
```

可以根据需要定义变量。以下设置代表默认值：

```
DEMO_MODE=posenet
MODEL=/usr/share/gstnninferencedemo/google-coral/project-
posenet/posenet_mobilenet_v1_075_353_481_quant_decoder.tflite
LABEL=no-label
DISPLAY_STATS=true
ENABLE_INFERENCE=true
USE_NNAPI=true
ROT=none (Rotation)
SYNC=true
```

12.2 NNStreamer

NNStreamer 是一种高效灵活的流 pipeline 框架，适用于复杂的神经网络应用。它最初是由三星开发的，然后作为孵化项目转移到了 LF AI 基金会。

它是一组 GStreamer 插件，帮助 GStreamer 开发人员轻松高效地采用神经网络模型，让神经网络开发人员能够轻松高效地管理神经网络 pipeline 及其过滤器。

该项目通过其专用的 github 文档站点进行了详细记录，但为方便起见，主要内容如下所述。

除了标准的 GStreamer 数据类型外，NNStreamer 还添加了新的数据类型“other/tensor”和“other/tensors”，这要归功于专用的转换器元件。此数据类型分别表示多维数组流和此类数组的多个实例的容器流。

NNStreamer 提供了一组对张量进行多种操作的流过滤器：

- **tensor_converter** 将音频、视频、文本或任意二进制流转换为 other/tensor 流。
- **tensor_decoder** 使用分配的子插件将“other/tensor(s)”转换为视频或文本流。
- **tensor_filter** 调用具有给定模型路径和神经网络框架名称的神经网络模型。
- **tensor_transform** 将各种算子应用于张量，包括 typecast、add、mul、transpose 和 normalize。为了加快处理速度，它在单个过滤器中支持 SIMD 指令和多个算子。
- **tensor_crop** 裁剪入局张量的区域。
- **tensor_rate** 控制张量流的帧速率。
- **tensor_mux**、**tensor_demux**、**tensor_merge**、**tensor_split**、**tensor_if** 和 **tensor_aggregator** 支持张量流路径控制。
- **tensor_sink** 是一个 sink 插件，用于使应用程序获取 other/tensor(s)的缓冲区。
- **tensor_source** 允许非 GStreamer 标准输入源（例如传感器）提供 *other/tensor(s)*流。
- **tensor_reposink** 和 **tensor_reposrc** 实施递归路径助手，通过专用的共享存储库缩短 GStreamer pipeline 周期。**tensor_reposink** 将数据推送到存储库，后者通过 tensor_reposrc 元素将数据重新注入上游。

下图描述了 NNStreamer pipeline 的一般架构：

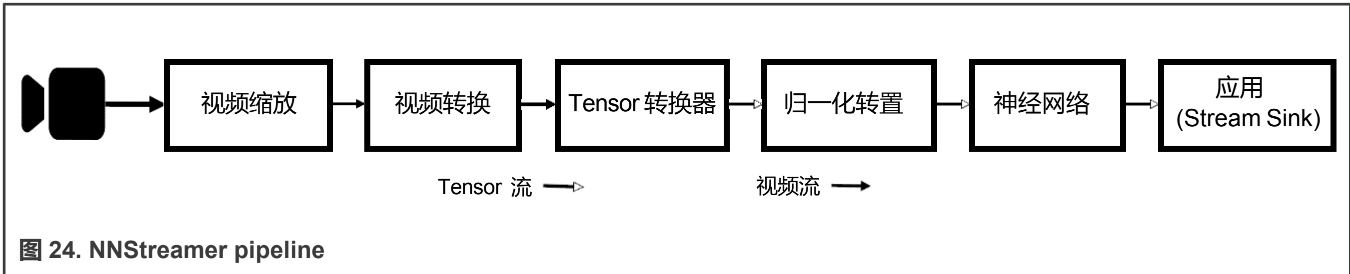

图 24. NNStreamer pipeline

有两个元件允许在运行时添加用户创建的功能：*tensor_filter* 和 *tensor_decoder*：

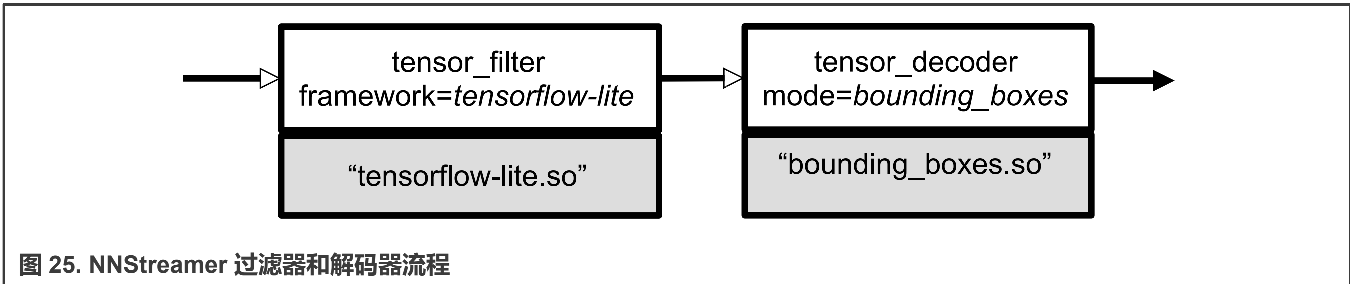

图 25. NNStreamer 过滤器和解码器流程

在实例化 *tensor_filter* 和 *tensor_decoder* 时，由于运行时加载了专用的共享库，框架和模式选项分别指定了目标实现。NNStreamer 提供了一组过滤器和解码器（下面进行了简要介绍），以及用于实施自定义用户子插件的 API。因此，可以使用专有推理引擎子插件作为张量过滤器，或使用专门的 NN 解码器。

NNStreamer 支持最主流的推理引擎（无论开源与否）。本版本支持 TensorFlow Lite 和 Arm NN 引擎。后续版本将支持更多推理引擎。

表 4. NNStreamer 支持的功能

框架/工具	i.MX8M Plus	i.MX8M Quad	i.MX8M Mini	i.MX8M Nano	i.MX8Q Max	i.MX8Q XP
TensorFlow Lite	CPU/NPU/GPU	CPU/GPU	CPU	CPU/GPU	CPU/GPU	CPU/GPU
Arm NN	CPU/NPU/GPU	CPU/GPU	CPU	CPU/GPU	CPU/GPU	CPU/GPU
自定义 C++	CPU	CPU	CPU	CPU	CPU	CPU
自定义 Python	CPU	CPU	CPU	CPU	CPU	CPU
NNShark	CPU	-	-	-	-	-

如果多个硬件后端支持推理引擎，则可以指定映射神经网络的器件。

尽管张量解码器元件可能不适合构建一个通常不会仅出于显示目的而使用神经网络输出的应用程序，但它对于在开发阶段实现原型非常有用，该原型可能侧重于神经网络模型或优化数据路径。事实上，大多数神经网络拓扑都支持经典计算机视觉用例：分类、目标检测、姿势估计或分割。

NNStreamer tensor 过滤器元件必须配置为使用特定的引擎和硬件加速器。下表列出了可用选项：

表 5. TensorFlow Lite 引擎

代理	张量过滤器属性	USE_GPU_INFERENCE 环境变量
无代理	framework=tensorflow-lite custom=NumThreads:4	-
NNAPI Delegate	framework=tensorflow-lite custom=Delegate:NNAPI	0: NPU 1: GPU
VX Delegate	framework=tensorflow-lite custom=Delegate:External,ExtDelegateLib:libvx_delegate.so	0: NPU 1: GPU
Arm NN Delegate	framework=tensorflow-lite custom=Delegate:External,ExtDelegateLib:libarmnnDelegate.so,ExtDelegateKeyVal:backends#<backend> backend = VsiNpu (NPU/GPU), CpuAcc	0: NPU 1: GPU

注意

XNNPACK Delegate 在此版本中不起作用。

表 6. Arm NN 引擎

后端	张量过滤器属性	USE_GPU_INFERENCE 环境变量
CPU	framework=armnn accelerator=true:cpu.neon	-
GPU/NPU	framework=armnn accelerator=true:npu	0: NPU 1: GPU

12.2.1 目标检测 pipeline 示例

在该示例中，将利用 i.MX 8M Plus 上可用的大部分计算后端来实施以下 pipeline，构建目标检测场景。

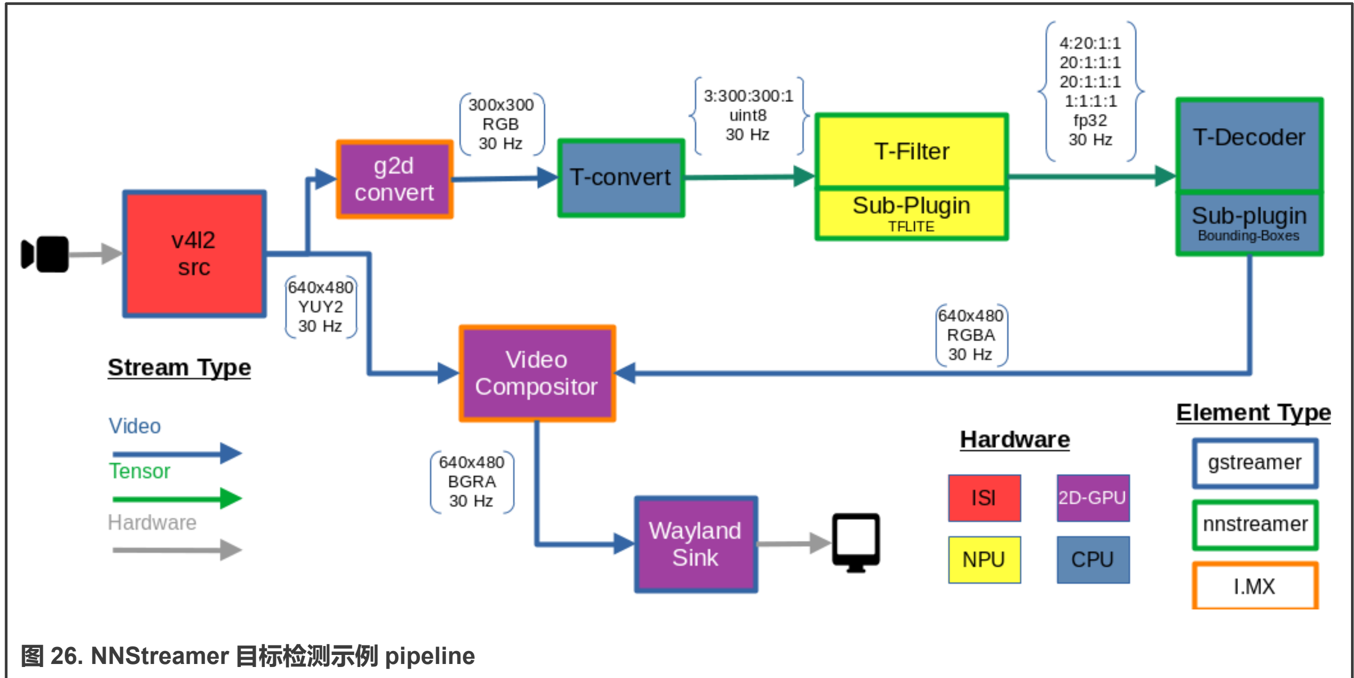

图 26. NNStreamer 目标检测示例 pipeline

在目标上，从 google coral github 网站下载经过训练的神经网络，并将文件名导出到 bash 环境变量：

```
root:~#  wget https://github.com/google-coral/test_data/raw/master/ssd_mobilenet_v2_coco_quant_postprocess.tflite
root:~#  wget  https://github.com/google-coral/test_data/raw/master/coco_labels.txt
root:~#  export  MODEL=$(pwd)/ssd_mobilenet_v2_coco_quant_postprocess.tflite
root:~#  export  LABELS=$(pwd)/coco_labels.txt
```

然后构建并执行 Gstreamer pipeline：

```
root:~# gst-launch-1.0 --no-position v4l2src device=/dev/video3 ! \
video/x-raw,width=640,height=480,framerate=30/1  !  \
tee name=t t. ! queue max-size-buffers=2 leaky=2 ! \
imxvideoconvert_g2d ! \
video/x-raw,width=300,height=300,format=RGBA  !  \
videoconvert ! video/x-raw,format=RGB ! \
tensor_converter ! \
tensor_filter framework=tensorflow-lite model=${MODEL} custom=Delegate:NNAPI ! \
tensor_decoder mode=bounding_boxes option1=tf-ssd option2=${LABELS} \
option3=0:1:2:3,50 option4=640:480 option5=300:300 ! \
mix. t. ! queue max-size-buffers=2 ! \
imxcompositor_g2d name=mix sink_0::zorder=2 sink_1::zorder=1 ! waylandsink
```

注意

如有必要，按 CTRL+C 键停止执行。

12.2.2 Pipeline 分析

NNStreamer 团队开发了 NNShark，这是一个基于 GstShark 的分析工具，用于监测多个用于评估 SoC 硬件使用情况的 pipeline 指标。

NNShark 只能在 i.MX8M Plus 上使用，其中添加了特定指标：

- 2D GPU (GC520L) 利用率负载
- 3D GPU (GC7000UL) 利用率负载

- NPU (GC8000) 利用率负载
- Linux 内核性能工具报告的 SoC 主带宽
- 此外，如果用户可以使用功率测量评估套件，功率测量工具（PMT）会报告电源域消耗。

考虑到涉及并发阶段的复杂 GPU/NPU 架构，其报告的利用率负载应视为一个数量级，可能无法准确地反映每个阶段的状态。

注意

如需了解源代码演示位置，请参见 nnshark 存储库。

12.2.2.1 启用 NNShark 分析

建议通过 SSH 连接到目标，因为 NNShark UI 刷新率可能无法在串行控制台上正常显示。

通过环境变量启用 NNShark 分析：

```
root:~# export GST_DEBUG="GST_TRACER:7"
root:~# export GST_TRACERS="live"
```

为了获得 GPU 使用率测量值，必须禁用 GPU 驱动程序（galcore）中的节能功能，需要使用命令行内核参数。在执行 boot 命令之前，可以手动编辑 bootargs uboot 变量，添加以下参数：

```
galcore.gpuProfiler=1 galcore.powerManagement=0
```

然后运行上一个 gst-launch 命令行，终端屏幕上应显示如下内容。可以使用上/下方向键滚动浏览所有 pipeline 元件，以选择需要的元件并显示其与其他 pipeline 元件的连接。

可以使用左/右方向键选择元件引脚，以突出显示其与其他元件引脚的连接。

在本示例中，张量过滤器的平均处理时间为 21.64 毫秒，其用橙色突出显示的 sink 引脚连接到 tensorconverter0 元件的源引脚（用绿色突出显示）。

按 “q” 或 “Q” 退出分析工具并返回 shell 终端。如前所述，可以通过 CTRL+C 退出应用程序。

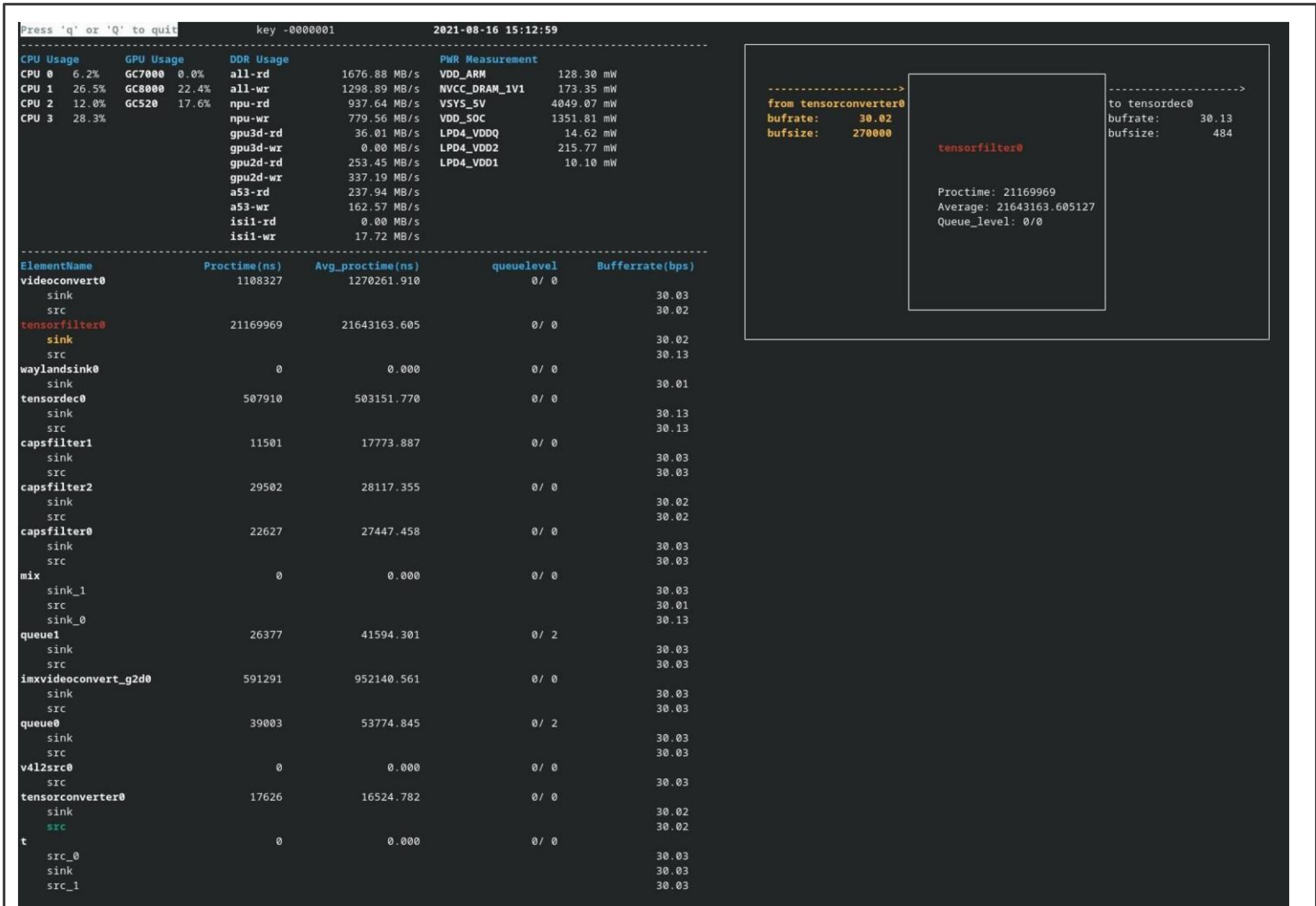

图 27. NNShark i.MX8M Plus 示例截图

12.2.2.2 向 NNShark 添加功率测量

在连接到功率测量评估套件的台式计算机上，在服务器模式下执行功率测量工具（PMT），例如收集功率测量并在 65432 TCP/IP 端口上提供。

```
user@localhost:pmt# python3 main.py server -b imx8mpevkpwra0 -p 65432
```

在目标上，导出台式计算机的 IP 地址（本示例中为 192.168.1.99）：

```
root:~# export GST_TRACERS_PWR_SERVER_IP=192.168.1.99
```

注意

用户可以在不使用功率测量套件的情况下运行 NNShark。

12.2.2.3 已知问题和限制

如果 perf 报告的数字不一致，则意味着 perf 进程仍在上次运行的后台运行。在这种情况下，必须手动终止它们的执行。

为方便起见，可以使用以下命令：

```
root:~# kill -9 $(ps -ef | grep nnshark-perf-ddr.sh | grep -v grep | tr -s ' ' | cut -d ' ' -f 2)
```

12.3 AWS 端到端 SageMaker 演示

AWS SageMaker 演示展示了如何使用 i.MX BSP 中预构建的 AWS IoT Greengrass 和 SageMaker Edge Manager 包，通过云服务构建、部署和管理机器学习模型和设备软件。

AWS IoT Greengrass 是一款将云功能扩展到本地设备的软件。它通过 MQTT 协议启用本地设备消息传输，建立与云端的安全连接。AWS SageMaker Edge Manager 提供了一个在边缘设备上运行以进行模型推理的软件代理，以及用于管理边缘设备上的模型的单独的 SageMaker Neo 云服务。

特性：

- AWS IoT Greengrass v2
- AWS Sagemaker Edge Manager 代理
- AWS 命令行接口 (AWS CLI) v1.21.12
- 基于 AWS CLI 的自动脚本示例，用于提供、操作云服务和设备
- 视频推理演示，执行以下任务：
  - 在云端部署模型
  - USB 摄像头捕捉图像帧
  - 推理结果返回云端

12.3.1 AWS Greengrass/SageMaker 演示工作流程

该端到端流程（另请参见下图）使用预训练的 mobilenetv2 图像分类模型，使用从 USB 摄像头捕获的图像在边缘执行图像分类。推理在 i.MX 8M Plus 的 NPU 上执行，与仅在 CPU 上运行相比，它将性能提高了 50 倍。结果上传到 AWS IoT，输入和输出张量上传到 Amazon S3。

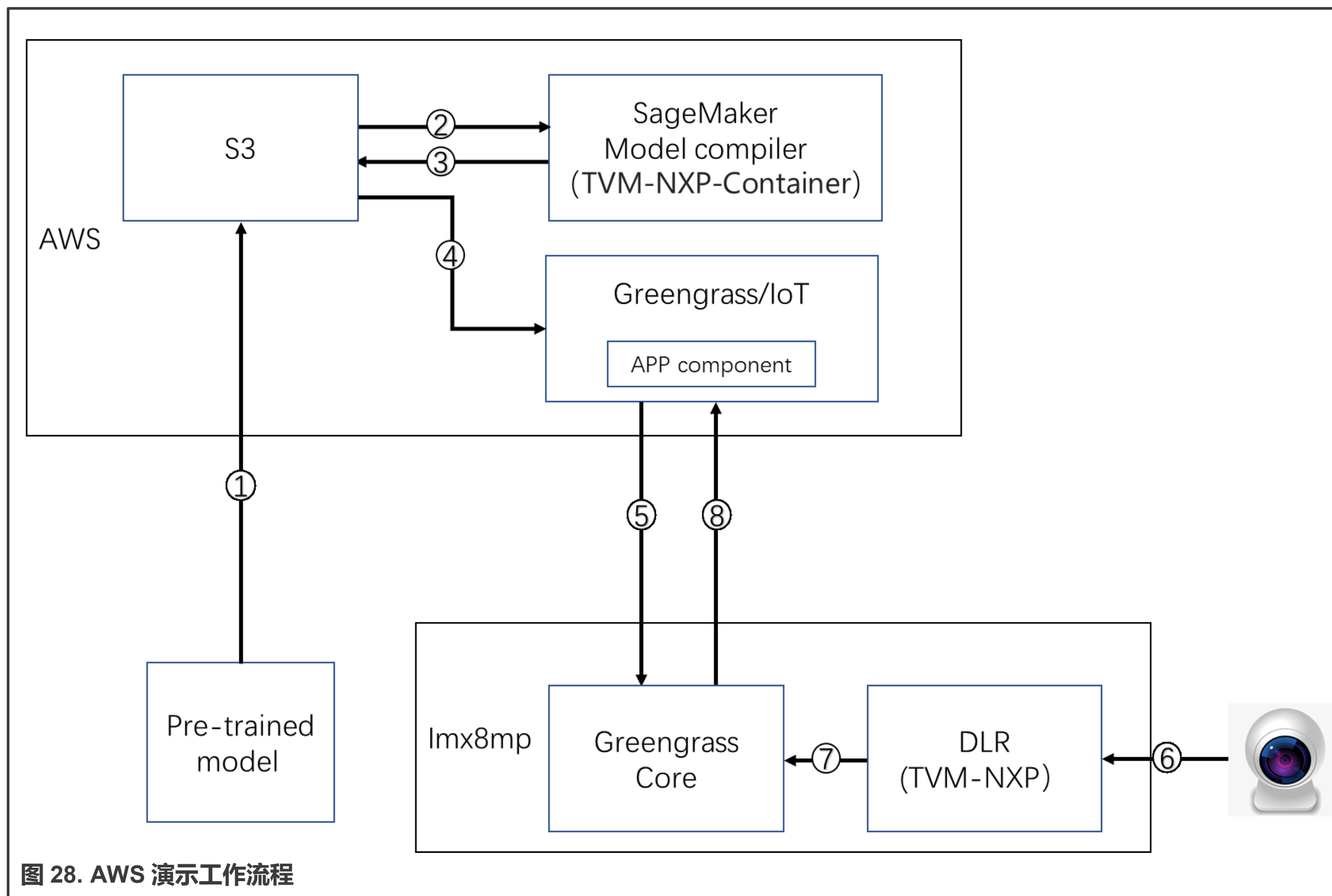

图 28. AWS 演示工作流程

演示工作流程如下所示：

1. 用户将预训练模型上传到 AWS S3。
2. SageMaker 模型编译器（恩智浦提供给 AWS 的容器）获取模型并为 i.MX 8M Plus NPU 编译二进制文件。
3. 容器将二进制文件上传回 S3。
4. Greengrass/IoT 将模型二进制文件和用户代码打包到 APP 组件中。
5. Greengrass/IoT 将 APP 组件部署到边缘设备（i.MX 8M Plus）。
6. APP 组件从摄像头获取图像。
7. APP 组件在 DLR（恩智浦提供的 TVM runtime）上运行模型。
8. Greengrass Core 将推理结果发送到 AWS。

要求具备以下项目：

- 恩智浦 i.MX8MP-EVK BSP ，预内置了 AWS 器件包
- AWS 账户
- AWS 账户的证书和私钥
- 连接到恩智浦 i.MX8MP-EVK 的 USB 摄像头

12.3.2 开始

12.3.2.1 构建 BSP 镜像

采用 AWS 软件包和演示脚本进行构建：

- 遵循《*i.MX Yocto Project 用户指南*》（*IMXLXYOCTOUG*）设置项目
- 构建镜像：

```
$ DISTRO=fsl-imx-wayland MACHINE=imx8mpevk source imx-aws-setup-release.sh -b build-imx8mp
$ bitbake imx-image-full
```

- 将镜像闪存到 SD 卡

```
$ sudo dd if=imx-image-full-imx8mpevk.wic of=/dev/xxxx
```

- 使用此 SD 卡启动电路板

12.3.2.2 在器件上运行演示脚本

启动电路板后，可以在`/usr/bin/dlr-demo-scripts` 文件夹下找到演示脚本。这些脚本可以使用云资源运行，还可以搭建演示环境：

```
root@imx8mpevk:/usr/bin/dlr-demo-scripts# ls -l *.sh
00_setup_cloud_services.sh
01_create_greengrass_core.sh
02_create_greengrass_role.sh
03_upload_component_version.sh
04_create_device_fleet_register_device.sh
05_compile_and_package_neo_model.sh
06_create_greengrass_deployment.sh
07_setup_device_greengrass.sh
10_clean_up.sh
setup_cloud_service_and_device.sh
```

在运行这些脚本之前，需要指定以下环境变量：

- 设置 AWS 密钥环境：

```
$ export AWS_ACCESS_KEY_ID="YOUR AWS ACCESS KEY ID"
$ export AWS_SECRET_ACCESS_KEY="YOUR AWS SECRET ACCESS KEY"
$ export AWS_SESSION_TOKEN="YOUR AWS SESSION TOKEN"
$ export AWS_REGION="us-west-2"    #replace with your aws region
```

- 如有必要，可选择设置 ARN 权限边界。您可以在 AWS 管理控制台->IAM->策略（*AWS management Console->IAM->Policies*）中找到它：

```
$ export PERMISSIONS_BOUNDARY="YOUR PERMISSIONS BOUNDARY ARN"
```

- 如有必要，可选择设置摄像头设备 ID。默认值为 3：

```
$ export CAMERA_DEVICE=3
```

- 将 PROJECT_NAME 设置为一个只有小写字母的唯一字符串：

```
$ export PROJECT_NAME={project_name}
```

- 运行演示脚本：

```
$ cd /usr/bin/dlr-demo-scripts
$ ./setup_cloud_service_and_device.sh
```

12.3.2.3 查看推理结果

可通过两种方式查看推理结果：

1. 从设备 Greengrass 日志文件：

```
$ cd /greengrass/v2/logs
$ tail -f aws.sagemaker.${project_name}_edgeManagerClientCamera Integration.log

stdout. {'index': '750', 'confidence': '0.4980392156862745', 'performance': '9.131669998168945',
'model_name': 'mobilenetv2-224-10-quant'}.

stdout. {'index': '831', 'confidence': '0.49411764705882355', 'performance':
'15.126943588256836', 'model_name': 'mobilenetv2-224-10-quant'}.
```

2. 从云服务控制台：

   转到 AWS IoT 控制台->测试->MQTT 测试客户端（*AWS IoT Console -> Test -> MQTT test client*）（参见下图）。在“订阅”菜单下，选择“em/inference”。每一秒，推理结果都应该到达“em/inference”主题，带有结果和置信度水平。

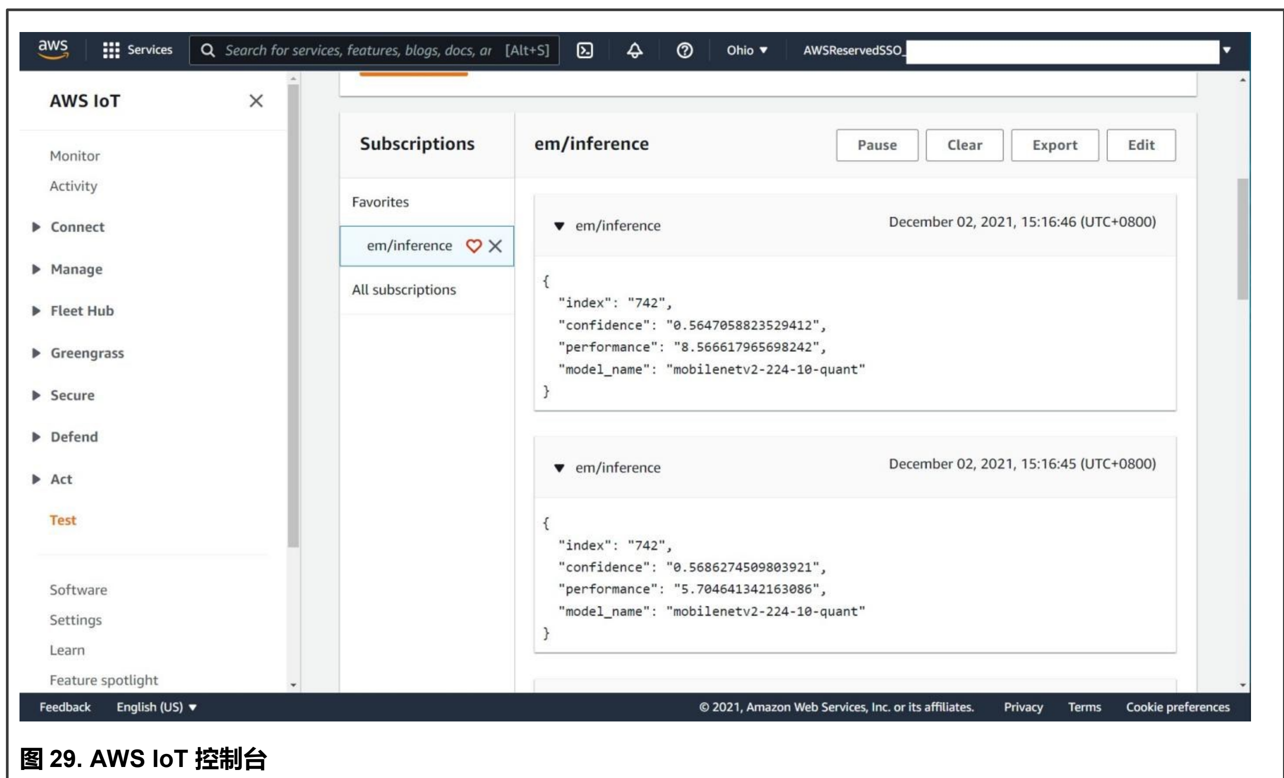

图 29. AWS IoT 控制台

12.3.2.4 清理云环境

测试完成后，释放云资源以节省成本：

```
$ /usr/bin/dlr-demo-scripts/10_clean_up.sh
```

12.3.3 其他资源

如需了解 AWS IoT Greengrass 的更多详细信息，请参见以下链接：

- AWS IoT Greengrass：什么是 AWS IoT Greengrass？ - AWS IoT Greengrass (amazon.com)
- SageMaker Edge Manager：SageMaker Edge Manager - Amazon SageMaker

- Greengrass sagemaker 示例：Greengrass-v2-sagemaker-edge-manager-python
- IAM 和权限边界：权限边界

第 13 章
修订历史

此表提供修订历史信息。

表 7. 修订历史

修订号	日期	实质性变更
L5.4.47_2.2.0	2020 年 9 月	初始版
L5.4.70_2.3.0	2021 年 1 月	i.MX 5.4 合并了 GA 用于发布 i.MX 电路板，包括 i.MX 8M Plus 和 i.MX 8DXL。
Linux LF5.10.9_1.0.0	2021 年 3 月	内核升级到 5.10.9 ，机器学习升级
L5.4.70_2.3.2	2021 年 4 月	发布补丁
Linux LF5.10.35_2.0.0	2021 年 6 月	升级到 Yocto Project Hardknott ，内核升级到 5.10.35
Linux LF5.10.52_2.1.0	2021 年 9 月	为 i.MX 8ULP Alpha 进行了更新，内核升级到 5.10.52
Linux LF5.10.72_2.2.0	2021 年 12 月	将内核升级到 5.10.72 并更新了 BSP

附录 A 版本说明

面向 LF5.10.72-2.2.0 的 TensorFlow Lite 2.6.0：

主要特性和改进：

- VX Delegate 改为外部代理。
- Python API 支持外部代理：
  - Python API 支持通过 `tlite.load_delegate()`调用使用外部代理
  - NNAPI delegate 在 Python API 中不可用。对于硬件加速器上的模型加速，可以使用 VX delegate：

    ```
    ext_delegate = [ tflite.load_delegate("/usr/lib/libvx_delegate.so") ]
    interpreter = tflite.Interpreter(model_path=args.model_file,
    experimental_delegates=ext_delegate, num_threads=args.num_threads)
    ```

  - 通过此更改，label_image.py Python 示例支持使用带参数的外部代理。如需了解更多信息，请参见“帮助”。
- 优化 NPU 硬件加速器上的 PCQ 转置卷积算子。

已知问题和限制：

- 由于 Yocto SDK 中缺少 TensorFlow Lite 的 protobuf include 文件，无法使用 Yocto SDK 构建评估工具：

  TensorFlow Lite 使用的 protobuf 版本与 Yocto SDK 中的版本不同（分别为 3.9.2 和 3.15.2）。生成的 Yocto SDK 上未安装 TensorFlow Lite 的 protobuf（tensorflow-protobuf-dev 软件包），因此尝试构建 TensorFlow Lite 模型评估工具会失败。需要手动将 tensorflow-protobuf-dev（libprotobuf-dev_3.9.2-r0_arm64.deb）包提取到 Yocto SDK 中：

  ```
  dpkg -x libprotobuf-dev_3.9.2-r0_arm64.deb <PATH_TO_YOCTO_SDK>/sysroots/cortexa53-crypto-poky-linux/
  ```

  该软件包位于 Yocto build 文件夹中的 `tmp/deploy/deb/cortexa53-crypto/`中。
- NNAPI 实施不支持 TransposeConv2D 的隐式填充（implicit padding）：
  - TransposeConv2D 使用隐式填充模式的模型无法使用 NNAPI Delegate 运行，因为底层 NNAPI 实施不支持隐式填充模式。在这些模型中使用 VX Delegate。

面向 LF5.10.52-2.1.0 的 TensorFlow Lite 2.5.0：

主要特性和改进：

- Tf.lite：
  - 向 TensorFlow Lite 添加了 VX Delegate。 VX Delegate 是将 ML 推理分流到 i.MX8 片上加速器（GPU 或 NPU）的替代代理。

已知问题和限制：

- 由于 Yocto SDK 中没有 TensorFlow Lite 的 protobuf include 文件，无法使用 Yocto SDK 构建评估工具。
- TensorFlow Lite 使用的 protobuf 版本与 Yocto SDK 中的版本不同（分别为 3.9.2 与 3.15.2）。生成的 Yocto SDK 上未安装 TensorFlow Lite 的 protobuf（tensorflow-protobuf-dev 软件包），因此尝试构建 TensorFlow Lite 模型评估工具会失败。需要手动将 tensorflow-protobuf-dev (libprotobuf-dev_3.9.2-r0_arm64.deb)包提取到 Yocto SDK 中：

```
dpkg -x libprotobuf-dev_3.9.2-r0_arm64.deb <PATH_TO_YOCTO_SDK>/sysroots/cortexa53-crypto-poky-linux/
```

该软件包位于 Yocto build 文件夹中的 `tmp/deploy/deb/cortexa53-crypto/`。

- NNAPI 实施不支持 TransposeConv2D 的隐式填充：
  - — 为 TransposeConv2D 使用隐式填充模式的模型无法使用 NNAPI Delegate 运行，因为底层 NNAPI 实施不支持隐式填充模式。在这些模型中使用 VX Delegate。

面向 LF5.10.72-2.2.0 的 Arm 计算库 21.08：

主要特性和改进：

- 进行了重大版本更新，从 21.02 升级到 21.08。
- 有关社区添加的变更的完整列表，请参见 https://arm-software.github.io/ComputeLibrary/v21.08/ versions_changelogs.xhtml 和 https://arm-software.github.io/ComputeLibrary/v21.02/index.xhtml#S2_2_changelog

已知问题和限制：

- 仅构建 CPU 加速的 NEON 后端。将 Arm NN 与 VSI NPU 后端一起使用，利用 GPU 或 NPU 进行加速。

面向 LF5.10.72-2.2.0 的 Arm NN 21.08：

主要特性和改进：

- 进行了重大版本更新，从 21.02 升级到 21.08。
- TensorFlow Parser、Caffe Parser 和 Quantizer 已被删除，不再可用。现在只有 ONNX Parser、TensorFlow Lite Parser 和 Arm NN Delegate for TF Lite 可用于加载 .tflite 和 .onnx 模型。
- 如需了解社区添加的变更的完整列表，请参见 https://github.com/ARM-software/armnn/releases

已知问题和限制：

- 仅构建 ACL NEON 后端。 使用 VSI NPU 后端而不是 ACL OpenCL ，利用 GPU 或 NPU 进行加速。
- NPU 对 TransposeConv2D 进行了显著的性能优化，而 VSI NPU 后端不支持这些优化。如果您的模型大量使用 TransposeConv2D，请尝试将 TF Lite 与 VXDelegate 一起使用。

面向 LF5.10.72-2.2.0 的 ONNX Runtime 1.8.2：

主要特性和改进：

- 小幅版本更新，从 1.8.1 升级到 1.8.2。
- （实验性）Python API 支持，包括支持所有可用执行提供程序（CPU、ACL、Arm NN、NNAPI、VSI NPU）。
- 添加了 `/usr/bin/onnxruntime-1.8.2/onnxruntime_peft_test`。使用它而不使用 `onnx_test_runner` 来衡量模型的性能。
- 修复了 NPU 推理期间的详细日志记录。
- 更新了 ACL 和 Arm NN 后端，以利用 ACL 和 Arm NN 21.08。
- 所有 ONNX Runtime 工件（artifacts）都被安装到 `/usr/bin/onnxruntime-1.8.2`，而不是 `/usr/bin`。
- 如需了解社区添加的变更的完整列表，请参见 https://github.com/microsoft/onnxruntime/releases。

已知问题和限制：

- NPU 对 TransposeConv2D 进行了显著的性能优化，而 VSI NPU 执行提供程序不支持这些优化。如果您的模型大量使用 TransposeConv2D，请尝试将 TF Lite 与 VXDelegate 一起使用。

- 用 NNAPI 执行提供程序运行 SqueezeNet 会产生不正确的结果。

面向 LF5.10.72-2.2.0 的 DeepViewRT 2.4.36：

主要特性和改进：

- 小幅版本更新，从 2.4.30 升级到 2.4.36。
- 提供支持 NPU 的 C API。
- 修复洗牌层的错误

已知问题和限制：

- nn_tensor_load_file_ex 是一个便捷函数，没有得到很好的优化。

附录 B 使用的变量列表

下表汇总了本文档中描述的特定推理引擎所使用的变量。通过 export 命令应用以下变量：

表 8. 系统变量综述

变量名称	说明
CNN_PERF	0：禁用（默认） 1：打印每次运算的执行时间（需要 VIV_VX_DEBUG_LEVEL=1）。如果设置了 VIV_VX_PROFILE=1，则默认值为 1。
NN_EXT_SHOW_PERF	0：禁用（默认） 1：显示更多分析细节（需要 VIV_VX_DEBUG_LEVEL=1）
PATH_ASSETS	设置用户资产的导出路径。
USE_GPU_INFERENCE	在 3D GPU (1) 和 NPU (其他情况) 之间进行选择。
VIV_VX_CACHE_BINARY_GRAPH_DIR	指定缓存的 NBG 的路径。默认为当前工作目录。
VIV_VX_DEBUG_LEVEL	0：禁用（默认） 1：在控制台打印驱动程序的调试信息。通常来说，这个环境变量和其他环境变量一起用于打印日志。
VIV_VX_ENABLE_CACHE_GRAPH_BINARY	0：禁用（默认） 1：启用图形缓存模式。如果存在缓存的 NBG 文件，网络将加载要运行的 NBG 文件。否则，它会生成一个 NBG 文件。这可以节省验证阶段的时间。
VIV_MEMORY_PROFILE	0：禁用（默认） 1：打印系统（CPU）和 GPU（VIP）的存储器尺寸（需要 VIV_VX_DEBUG_LEVEL=1）
VIV_VX_PROFILE	0：禁用（默认） 1：打印 DDR 读写带宽、AXI_SRAM 读写带宽、VIP 执行周期数。计数器是每节点进程（需要 VIV_VX_DEBUG_LEVEL=1）。 2：打印 DDR 读写带宽、AXI_SRAM 读写带宽、VIP 执行的周期数。计数器是每图进程（需要 VIV_VX_DEBUG_LEVEL=1）。

附录 C
神经网络 API 参考

下表列出了神经网络操作和支持的 API 函数。

表 9. 神经网络操作和支持的 API 函数

操作类别/名称	Android NNAPI 1.2	DeepViewRT 2.4.36	TensorFlow Lite 2.6.0	Arm NN 21.08	ONNX 1.8.2
激活					
elu	-	-	ELU	-	Elu
floor	ANEURALNETWORKS_FLOOR	-	Floor	Floor	Floor
leakyrelu	-	leaky_relu	-	Activation/LeakyReLu	LeakyReL
prelu	ANEURALNETWORKS_PRELU	prelu	PRELU	PreLu	PreLu
relu	ANEURALNETWORKS_RELU	relu	RELU	Activation/ReLu	ReLu
relu1	ANEURALNETWORKS_RELU1	-	RELU1	-	-
relu6	ANEURALNETWORKS_RELU6	relu6	RELU6	-	-
Hard_swish	ANEURALNETWORKS_HARD_SWISH	swish	HARD_SWISH	-	-
rsqrt	ANEURALNETWORKS_RSQRT	rsqrt	RSQRT	-	-
sigmoid	ANEURALNETWORKS_LOGISTIC	sigmoid/sigmoid_fast	LOGISTIC	Activation/Sigmoid	Sigmoid
softmax	ANEURALNETWORKS_SOFTMAX	softmax	SOFTMAX	Softmax	Softmax
softrelu	-	-	-	Activation/SoftReLu	-
sqrt	ANEURALNETWORKS_SQRT	sqrt	SQRT	Activation/Sqrt	Sqrt
tanh	ANEURALNETWORKS_TANH	tanh	TANH	Activation/TanH	TanH
bounded	-	-	-	Activation/BoundedReLu	-
linear	-	linear	-	Activation/Linear	-

下页继续……

表 9. 神经网络操作和支持的 API 函数（续）

操作类别/名称	Android NNAPI 1.2	DeepViewRT 2.4.36	TensorFlow Lite 2.6.0	Arm NN 21.08	ONNX 1.8.2
密集层					
dense	-	dense	-	-	-
Element Wise					
abs	ANEURALNETWORKS_ABS	abs	ABS	Activation/Abs	Abs
add	ANEURALNETWORKS_ADD	add	ADD	Addition	Add
clip_by_value	-	-	-	-	Clip
div	ANEURALNETWORKS_DIdV	divide	DIV	Division	Div
equal	ANEURALNETWORKS_EQUAL	-	EQUAL	-	Equal
exp	ANEURALNETWORKS_EXP	exp	EXP	-	Exp
log	ANEURALNETWORKS_LOG	log	LOG	-	Log
greater	ANEURALNETWORKS_GREATER	-	GREATER	-	Greater
greater_equal	ANEURALNETWORKS_GREATER_EQUAL	-	GREATER_EQUAL	-	-
less	ANEURALNETWORKS_LESS	-	LESS	-	Less
less_equal	ANEURALNETWORKS_LESS_EQUAL	-	LESS_EQUAL	-	-
logical_and	ANEURALNETWORKS_LOGICAL_AND	-	LOGICAL_AND	-	And
logical_or	ANEURALNETWORKS_LOGICAL_OR	-	LOGICAL_OR	-	Or
minimum	ANEURALNETWORKS_MINIMUM	-	MINIMUM	Minimum	Min
maximum	ANEURALNETWORKS_MAXIMUM	-	MAXIMUM	Maximum	Max

下页继续……

表 9. 神经网络操作和支持的 API 函数（续）

操作类别/名称	Android NNAPI 1.2	DeepViewRT 2.4.36	TensorFlow Lite 2.6.0	Arm NN 21.08	ONNX 1.8.2
multiply	ANEURALNETWORKS_MUL	multiply	MUL	Multiplication	Mul
negative	ANEURALNETWORKS_NEG	-	NEG	-	Neg
not_equal	ANEURALNETWORKS_NOT_EQUAL	-	NOT_EQUAL	-	-
pow	ANEURALNETWORKS_POW	-	POW	-	POW
select	ANEURALNETWORKS_SELECT	-	SELECT	-	-
square	-	-	-	Activation/Square	-
sub	ANEURALNETWORKS_SUB	substract	SUB	Substraction	Sub
where	-	-	-	-	Where
图像处理					
resize_bilinear	ANEURALNETWORKS_RESIZE_BILINEAR	-	RESIZE_BILINEAR	-	Unsample
resize_nearest_neighbor	ANEURALNETWORKS_RESIZE_NEAREST_NEIGHBOR	resize	RESIZE_NEAREST_NEIGHBOR	-	Resize
矩阵乘法					
fullconnect	ANEURALNETWORKS_FULLY_CONNECTED	-	FULLY_CONNECTED	FullyConnected	-
matrix_mul	-	matmul/ matmul_cache	-	-	-
归一化					-
batch_normalize	-	batchnorm	-	BatchNormalization	BatchNormalization
instance _normalize	-	-	-	Normalization	InstanceNormalization
l2normalize	ANEURALNETWORKS_L2_NORMALIZATION	-	L2_NORMALIZATION	L2Normalization	-

下页继续……

表 9. 神经网络操作和支持的 API 函数（续）

操作类别/名称	Android NNAPI 1.2	DeepViewRT 2.4.36	TensorFlow Lite 2.6.0	Arm NN 21.08	ONNX 1.8.2
localresponsenormalization	ANEURALNETWORKS_LOCAL_RESPONSE_NORMALIZATION	-	LOCAL_RESPONSE_NORMALIZATION	-	LRN
Reshape					
batch2space	ANEURALNETWORKS_BATCH_TO_ SPACE_ND	-	BATH_TO_SPACE_ND	BatchToSpaceNd	-
concat	ANEURALNETWORKS_CONCATENATION	-	CONCATENATION	Concat	Concat
depth_to_space	ANEURALNETWORKS_DEPTH_TO_ SPACE	-	DEPTH_TO_SPACE	-	DepthToSpace
expanddims	ANEURALNETWORKS_EXPAND_DIMS	-	EXPAND_DIMS	-	-
flatten	ANEURALNETWORKS_RESHAPE	-	-	-	-
gather	ANEURALNETWORKS_GATHER	-	GATHER	-	Gather
pad	ANEURALNETWORKS_PAD	-	PAD	Pad	Pad
permute	ANEURALNETWORKS_TRANSPOSE	-	TRANSPOSE	Permute	Transpose
reducemean	ANEURALNETWORKS_MEAN	reduce_mean	MEAN	Mean	ReduceMean
reducesum	ANEURALNETWORKS_SUM	reduce_sum	REDUCE_SUM	-	ReduseSum
gathernd	-	-	-	-	GatherND
reducemax	ANEURALNETWORKS_REDUCE_MAX	reduce_max	REDUCE_MAX	-	ReduceMax
reducemin	ANEURALNETWORKS_REDUCE_MIN	reduce_min	REDUCE_MIN	-	ReduceMin
reduceproduct	-	reduce_product	-	-	-

下页继续……

表 9. 神经网络操作和支持的 API 函数（续）

操作类别/名称	Android NNAPI 1.2	DeepViewRT 2.4.36	TensorFlow Lite 2.6.0	Arm NN 21.08	ONNX 1.8.2
reshape	ANEURALNETWORKS_RESHAPE	-	RESHAPE	Reshape	Reshape
reverse	-	-	-	-	ReverseSequence
slice	ANEURALNETWORKS_SLICE	-	SLICE	-	Slice
space2batch	ANEURALNETWORKS_SPACE_TO_ BATCH_ND	-	SPACE_TO_BATCH_ND	SpaceToBatchNd	-
split	ANEURALNETWORKS_SPLIT	-	SPLIT	Split	Split
squeeze	ANEURALNETWORKS_SQUEEZE	-	SQUEEZE	Squeeze	Squeeze
strided_slice	ANEURALNETWORKS_STRIDED_SLICE	-	STRIDED_SLICE	StridedSlice	-
unstack	-	-	-	Unpack	-
RNN					
gru	-	-	-	-	GRU
lstm	-	-	UNIDIRECTIONAL_SEQUEENCE_LSTM	-	-
lstmunit	ANEURALNETWORKS_LSTM	-	LSTM	LstmUnit	LSTM
rnn	ANEURALNETWORKS_RNN	-	RNN	-	-
滑动窗口					
avg_pool	ANEURALNETWORKS_AVERAGE_POOL	avgpool/ avgpool_ex	AVERAGE_POOL_2D	Pooling2D/avg	AveragePool
convolution	ANEURALNETWORKS_CONV_2D	conv/conv_ex	CONV_2D	Convolution2D	Conv
deconvolution	ANEURALNETWORKS_TRANSPOSE_CONV_2D	transpose_conv2d_ex	TRANSPOSE_CONV	-	ConvTranspose
depthhwise_convolution	ANEURALNETWORKS_	-	DEPTHWISE_CONV_2D	Depthwise Convolution	-

下页继续......

表 9. 神经网络操作和支持的 API 函数（续）

操作类别/名称	Android NNAPI 1.2	DeepViewRT 2.4.36	TensorFlow Lite 2.6.0	Arm NN 21.08	ONNX 1.8.2
	DEPTHWISE_ CONV_2D				
Log_softmax	ANEURALNETWO RKS_LOG_ SOFTMAX	-	LOG_SOFTMAX	-	Logsoftmax
l2pooling	ANEURALNETWO RKS_L2_POOL	-	L2_POOL_2D	Pooling2D/L2	-
max_pool	ANEURALNETWO RKS_MAX_POOL	maxpool/ maxpool_ex	MAX_POOL_2D	Pooling2D/max	MaxPool
其他					
argmax	ANEURALNETWO RKS_ARGMAX	argmax	ARGMAX	-	ArgMax
argmin	ANEURALNETWO RKS_ARGMIN	-	ARGMIN	-	ArgMin
dequantize	ANEURALNETWO RKS_ DEQUANTIZE	-	DEQUANTIZE	Dequantize	DequantizeLinear
quantize	ANEURALNETWO RKS_QUANTIZE	-	QUANTIZE	Quantize	QuantizeLinear
roi_pool	ANEURALNETWO RKS_ROI_ALIGN	-	-	-	-
shuffle_channel	ANEURALNETWO RKS_CHANNEL_ SHUFFLE	-	-	-	-
tile	ANEURALNETWO RKS_TILE	-	TILE	-	Tile
svdf	ANEURALNETWO RKS_SVDF	-	SVDF	-	-
embedding_lookup	ANEURALNETWO RKS_ EMBEDDING_ LOOKUP	-	EMBEDDING_ LOOKUP	-	-
cast	ANEURALNETWO RKS_CAST	-	CAST	-	Cast
ssd	-	ssd_decode_nms_ standard_bbx/ ssd_decode_nms_ variance_bbx/ ssd_nms_full	-	-	-

附录 D
GPU 支持 OVXLIB 操作

本节概述了恩智浦图形处理单元（GPU）IP 支持的神经网络 OVXLIB 操作，硬件支持 OpenVX 和 OpenCL 以及兼容的软件栈。下表列出了 OVXLIB 操作。

以下缩写用于表示格式类型：

- **asym-u8：**asymmetric_affine-uint8
- **asym-i8：**asymmetric_affine-int8
- **fp32：**float32
- **pc-sym-i8：**perchannel_symmetric_int8
- **fp16：**float16
- **bool8：**bool8
- **int16：**int16
- **int32：**int32

表 10. GPU 支持 OVXLIB 操作

OVXLIB 操作	张量			执行引擎	
	输入	内核	输出	OpenVX	OpenCL
基本操作					
VSI_NN_OP_ CONV2D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_ CONV1D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_ DEPTHWISE_ CONV1D	asym-u8	asym-u8	asym-u8	✓	
	asym-i8	asym-i8	asym-i8	✓	
VSI_NN_OP_DEC ONVOLUTION1D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_ DECONVOLUTION	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓

下页继续……

表 10. GPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎	
	输入	内核	输出	OpenVX	OpenCL
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_FCL	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_GROUPED_CONV1D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_GROUPED_CONV2D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
激活操作					
VSI_NN_OP_ELU	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_HARD_SIGMOID	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SWISH	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_LEAKY_RELU	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓

下页继续……

表 10. GPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎	
	输入	内核	输出	OpenVX	OpenCL
	fp16		fp16	✓	✓
VSI_NN_OP_ PRELU	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ RELU	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ RELUN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ RSQRT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ SIGMOID	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ SOFTRELU	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ SQRT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ TANH	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓

下页继续......

表 10. GPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎	
	输入	内核	输出	OpenVX	OpenCL
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ABS	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_CLIP	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_EXP	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_LOG	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_NEG	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_MISH	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_LINEAR	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ERF	asym-u8		asym-u8	✓	✓

下页继续……

表 10. GPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎	
	输入	内核	输出	OpenVX	OpenCL
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ SOFTMAX	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ LOG_SOFTMAX	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ SQUARE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SIN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
Elementwise 操作					
VSI_NN_OP_ADD	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ SUBTRACT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ MULTIPLY	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓

下页继续……

表 10. GPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎	
	输入	内核	输出	OpenVX	OpenCL
	fp16		fp16	✓	✓
VSI_NN_OP_DIVIDE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_MAXIMUN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_MINIMUM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_POW	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_FLOORDIV	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_MATRIXMUL	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_RELATIONAL_OPS	asym-u8		bool8	✓	✓
	asym-i8		bool8	✓	✓
	fp32		bool8	✓	✓
	fp16		bool8	✓	✓
	bool8		bool8	✓	✓
VSI_NN_OP_LOGICAL_OPS	bool8		bool8	✓	✓

下页继续……

表 10. GPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎	
	输入	内核	输出	OpenVX	OpenCL
VSI_NN_OP_LOGICAL_NOT	bool8		bool8	✓	✓
VSI_NN_OP_SELECT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
	bool8		bool8	✓	✓
VSI_NN_OP_ADDN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
归一化操作					
VSI_NN_OP_BATCH_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_LRN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_LRN2	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_L2_NORMALIZE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_L2NORMALZESCALE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓

下页继续……

表 10. GPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎	
	输入	内核	输出	OpenVX	OpenCL
	fp16		fp16	✓	✓
VSI_NN_OP_LAYER_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_INSTANCE_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_GROUP_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_BATCHNORM_SINGLE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_MOMENTS	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
Reshape 操作					
VSI_NN_OP_EXPAND_BROADCAST	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SLICE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓

下页继续……

表 10. GPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎	
	输入	内核	输出	OpenVX	OpenCL
VSI_NN_OP_SPLIT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_CONCAT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_STACK	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_UNSTACK	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_RESHAPE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SQUEEZE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_PERMUTE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_REORG	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓

下页继续……

表 10. GPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎	
	输入	内核	输出	OpenVX	OpenCL
	fp16		fp16	✓	✓
VSI_NN_OP_ SPACE2DEPTH	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ DEPTH2SPACE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ BATCH2SPACE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ SPACE2BATCH	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_PAD	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ REVERSE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ STRIDED_SLICE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ CROP	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓

下页继续......

表 10. GPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎	
	输入	内核	输出	OpenVX	OpenCL
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_REDUCE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ARGMX	asym-u8		asym-u8/int16/int32	✓	✓
	asym-i8		asym-u8/int16/int32	✓	✓
	fp32		int32	✓	✓
	fp16		asym-u8/int16/int32	✓	✓
VSI_NN_OP_ARGMIN	asym-u8		asym-u8/int16/int32	✓	✓
	asym-i8		asym-u8/int16/int32	✓	✓
	fp32		int32	✓	✓
	fp16		asym-u8/int16/int32	✓	✓
VSI_NN_OP_SHUFFLECHANNEL	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
RNN 操作					
VSI_NN_OP_LSTMUNIT_OVXLIB	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_LSTM_OVXLIB	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓

下页继续……

表 10. GPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎	
	输入	内核	输出	OpenVX	OpenCL
VSI_NN_OP_GRUCELL_OVXLIB	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_GRU_OVXLIB	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
VSI_NN_OP_SVDF	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	fp16	fp16	fp16	✓	✓
Pooling 操作					
VSI_NN_OP_ROI_POOL	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_POOLWITHARGMAX	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_UPSAMPLE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
其他操作					
VSI_NN_OP_PROPOSAL	asym-u8		asym-u8	✓	
	asym-i8		asym-i8	✓	
	fp32		fp32	✓	
	fp16		fp16	✓	

下页继续……

表 10. GPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎	
	输入	内核	输出	OpenVX	OpenCL
VSI_NN_OP_VARIABLE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_DROPOUT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_RESIZE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_INTERP	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_DATACONVERT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_A_TIMES_B_PLUS_C	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_FLOOR	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_EMBEDDING_LOOKUP	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓

下页继续……

表 10. GPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎	
	输入	内核	输出	OpenVX	OpenCL
	fp16		fp16	✓	✓
VSI_NN_OP_GATHER	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_GATHER_ND	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SCATTER_ND	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_TILE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_RELU_KERAS	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ELTWISEMAX	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_INSTANCE_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_FCL2	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓

下页继续……

表 10. GPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎	
	输入	内核	输出	OpenVX	OpenCL
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_POOL	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SIGNAL_FRAME	asym-u8		asym-u8	✓	
	asym-i8		asym-i8	✓	
	fp32		fp32	✓	
	fp16		fp16	✓	
VSI_NN_OP_CONCATSHIFT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_UPSAMPLESCALE	asym-u8		asym-u8	✓	
	asym-i8		asym-i8	✓	
	fp16		fp16	✓	
VSI_NN_OP_ROUND	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_CEIL	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_SEQUENCE_MASK	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_REPEAT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓

下页继续……

表 10. GPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎	
	输入	内核	输出	OpenVX	OpenCL
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_ONE_HOT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓
VSI_NN_OP_CAST	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	fp16		fp16	✓	✓

附录 E
NPU 支持 OVXLIB 操作

本节概述了恩智浦神经处理器单元（NPU）IP 和兼容软件栈支持的神经网络 OVXLIB 操作。下表列出了 OVXLIB 操作。

以下缩写用于表示格式类型：

- **asym-u8：**asymmetric_affine-uint8
- **asym-i8：**asymmetric_affine-int8
- **fp32：**float32
- **pc-sym-i8：**perchannel_symmetric-int8
- **fp16：**float16
- **bool8：**bool8
- **int16：**int16
- **int32：**int32

以下缩写用于表示硬件中的参考密钥执行引擎（NPU）：

- **NN：**神经网络引擎
- **PPU：**并行处理单元
- **TP：**张量处理器

表 11. NPU 支持 OVXLIB 操作

OVXLIB 操作	张量			执行引擎（NPU）		
	输入	内核	输出	NN	TP	PPU
基本操作						
VSI_NN_OP_CONV2D	asym-u8	asym-u8	asym-u8	✓		
	asym-i8	pc-sym-i8	asym-i8	✓		✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16			✓
VSI_NN_OP_CONV1D	asym-u8	asym-u8	asym-u8	✓		
	asym-i8	pc-sym-i8	asym-i8	✓		✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16			✓
VSI_NN_OP_DEPTHWISE_CONV1D	asym-u8	asym-u8	asym-u8			✓
	asym-i8	asym-i8	asym-i8			✓
VSI_NN_OP_DECONVOLUTION	asym-u8	asym-u8	asym-u8	✓		
	asym-i8	pc-sym-i8	asym-i8	✓		✓

下页继续……

表 11. NPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎（NPU）		
	输入	内核	输出	NN	TP	PPU
	fp32	fp32	fp32			✓
	fp16	fp16	fp16			✓
VSI_NN_OP_DECONVOLUTION1D	asym-u8	asym-u8	asym-u8	✓		
	asym-i8	pc-sym-i8	asym-i8	✓		✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16			✓
VSI_NN_OP_FCL	asym-u8	asym-u8	asym-u8		✓	
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16		✓	
VSI_NN_OP_GROUPED_CONV1D	asym-u8	asym-u8	asym-u8	✓		
	asym-i8	pc-sym-i8	asym-i8	✓		✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16			✓
VSI_NN_OP_GROUPED_CONV2D	asym-u8	asym-u8	asym-u8			
	asym-i8	pc-sym-i8	asym-i8			✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16			✓
激活操作						
VSI_NN_OP_ELU	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_HARD_SIGMOID	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_SWISH	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓

下页继续……

表 11. NPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎（NPU）		
	输入	内核	输出	NN	TP	PPU
	fp16		fp16		✓	
VSI_NN_OP_LEAKY_RELU	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_PRELU	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_RELU	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_RELUN	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_RSQRT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_SIGMOID	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_SOFTRELU	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_SQRT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓

下页继续……

表 11. NPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎（NPU）		
	输入	内核	输出	NN	TP	PPU
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_TANH	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_ABS	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_CLIP	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_EXP	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_LOG	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_NEG	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_MISH	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_SOFTMAX	asym-u8		asym-u8			✓

下页继续……

表 11. NPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎（NPU）		
	输入	内核	输出	NN	TP	PPU
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ LOG_ SOFTMAX	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ SQUARE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ SIN	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_LI NEAR	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_E RF	asym-u8		asym-u8		✓	✓
	asym-i8		asym-i8		✓	✓
	fp32		fp32			✓
	fp16		fp16		✓	✓
Elementwise 操作						
VSI_NN_OP_ ADD	asym-u8		asym-u8	✓		
	asym-i8		asym-i8	✓		
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ SUBTRACT	asym-u8		asym-u8	✓		
	asym-i8		asym-i8	✓		
	fp32		fp32			✓

下页继续……

表 11. NPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎（NPU）		
	输入	内核	输出	NN	TP	PPU
	fp16		fp16			✓
VSI_NN_OP_MULTIPLY	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_DIVIDE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_MAXIMUN	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_MINIMUM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_POW	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_FLOORDIV	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_MATRIXMUL	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_RELATIONAL_OPS	asym-u8		bool8			✓
	asym-i8		bool8			✓

下页继续……

表 11. NPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎（NPU）		
	输入	内核	输出	NN	TP	PPU
	fp32		bool8			✓
	fp16		bool8			✓
	bool8		bool8			✓
VSI_NN_OP_LOGICAL_OPS	bool8		bool8			✓
VSI_NN_OP_LOGICAL_NOT	bool8		bool8			✓
VSI_NN_OP_SELECT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
	bool8		bool8			✓
VSI_NN_OP_ADDN	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
归一化操作						
VSI_NN_OP_BATCH_NORM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_LRN	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_LRN2	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_L2_NORMALIZE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓

下页继续……

表 11. NPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎（NPU）		
	输入	内核	输出	NN	TP	PPU
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ L2NORMALZE SCALE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ LAYER_NORM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ INSTANCE_ NORM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ BATCHNORM_ SINGLE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ MOMENTS	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_G ROUP_NORM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
Reshape 操作						
VSI_NN_OP_ E XPAND_BROA DCAST	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓

下页继续......

表 11. NPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎（NPU）		
	输入	内核	输出	NN	TP	PPU
	fp16		fp16			✓
VSI_NN_OP_SLICE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_SPLIT	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_CONCAT	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_STACK	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_UNSTACK	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_RESHAPE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_SQUEEZE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_PERMUTE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	

下页继续……

表 11. NPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎（NPU）		
	输入	内核	输出	NN	TP	PPU
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_REORG	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_SPACE2DEPTH	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_DEPTH2SPACE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
	bool8		bool8			
VSI_NN_OP_BATCH2SPACE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_SPACE2BATCH	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_PAD	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_REVERSE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	

下页继续……

表 11. NPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎（NPU）		
	输入	内核	输出	NN	TP	PPU
VSI_NN_OP_STRIDED_SLICE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_CROP	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_REDUCE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ARGMAX	asym-u8		asym-u8/int16/int32			✓
	asym-i8		asym-u8/int16/int32			✓
	fp32		int32			✓
	fp16		asym-u8/int16/int32			✓
VSI_NN_OP_ARGMIN	asym-u8		asym-u8/int16/int32			✓
	asym-i8		asym-u8/int16/int32			✓
	fp32		int32			✓
	fp16		asym-u8/int16/int32			✓
VSI_NN_OP_SHUFFLECHANNEL	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
RNN 操作						
VSI_NN_OP_LSTMUNIT_OVXLIB	asym-u8	asym-u8	asym-u8		✓	✓
	asym-i8	pc-sym-i8	asym-i8		✓	✓

下页继续……

表 11. NPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎（NPU）		
	输入	内核	输出	NN	TP	PPU
	fp32	fp32	fp32			✓
	fp16	fp16	fp16		✓	✓
VSI_NN_OP_LSTM_OVXLIB	asym-u8	asym-u8	asym-u8		✓	✓
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16		✓	✓
VSI_NN_OP_GRUCELL_OVXLIB	asym-u8	asym-u8	asym-u8		✓	✓
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16		✓	✓
VSI_NN_OP_GRU_OVXLIB	asym-u8	asym-u8	asym-u8		✓	✓
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16		✓	✓
VSI_NN_OP_SVDF	asym-u8	asym-u8	asym-u8		✓	✓
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓
	fp16	fp16	fp16		✓	✓
Pooling 操作						
VSI_NN_OP_ROI_POOL	asym-u8		asym-u8		✓	✓
	asym-i8		asym-i8		✓	✓
	fp32		fp32			✓
	fp16		fp16		✓	✓
VSI_NN_OP_POOLWITHARGMAX	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_UPSAMPLE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓

下页继续……

表 11. NPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎（NPU）		
	输入	内核	输出	NN	TP	PPU
其他操作						
VSI_NN_OP_PROPOSAL	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_VARIABLE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_DROPOUT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_RESIZE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_INTERP	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_DATACONVERT	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_A_TIMES_B_PLUS_C	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_FLOOR	asym-u8		asym-u8			✓

下页继续……

表 11. NPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎（NPU）		
	输入	内核	输出	NN	TP	PPU
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_EMBEDDING_LOOKUP	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_GATHER	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_GATHER_ND	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_SCATTER_ND	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_TILE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_RELU_KERAS	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_ELTWISEMAX	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓

下页继续……

表 11. NPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎（NPU）		
	输入	内核	输出	NN	TP	PPU
VSI_NN_OP_INSTANCE_NORM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_FCL2	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_POOL	asym-u8		asym-u8	✓	✓	
	asym-i8		asym-i8	✓	✓	
	fp32		fp32			✓
	fp16		fp16		✓	
VSI_NN_OP_SIGNAL_FRAME	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_CONCATSHIFT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_UPSAMPLESCALE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp16		fp16			✓
VSI_NN_OP_ROUND	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_CEIL	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓

下页继续……

表 11. NPU 支持 OVXLIB 操作（续）

OVXLIB 操作	张量			执行引擎（NPU）		
	输入	内核	输出	NN	TP	PPU
VSI_NN_OP_SEQUENCE_MASK	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_REPEAT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_ONE_HOT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓
VSI_NN_OP_CAST	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	fp16		fp16			✓

How To Reach Us **Home Page:**
nxp.com
Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX,EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE, VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© NXP B.V. 2021. **All rights reserved.**

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 17 December 2021 Document identifier: IMXMLUG