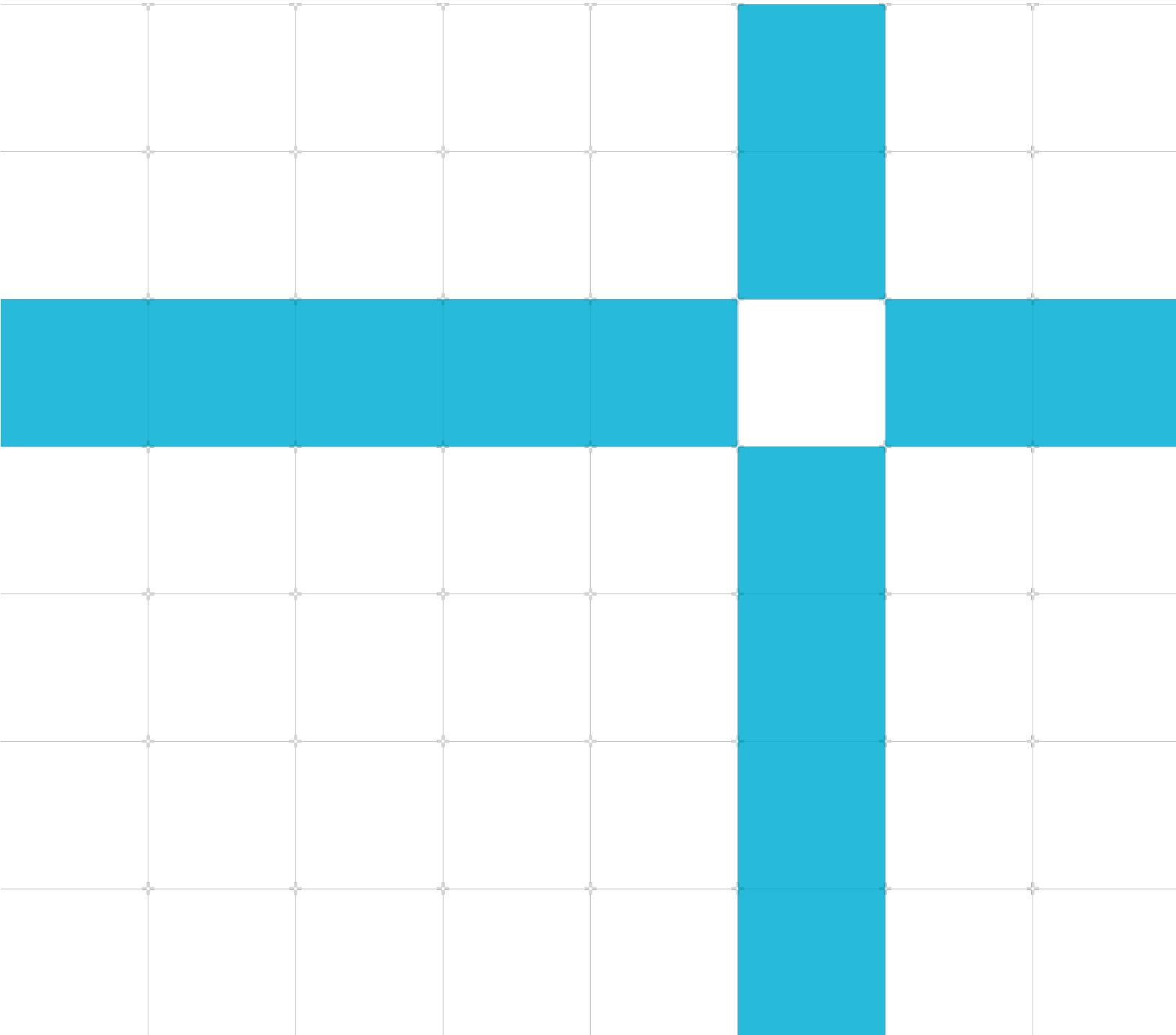
Revision: 1.1

Configure the Arm NN SDK build environment

Non-Confidential

Issue 02
ARM062-948681440-3565



Configure the Arm NN SDK build environment

Release information

Document history

Issue	Date	Confidentiality	Change
01	19 February 2021	Non-Confidential	First release
02	18 May 2021	Non-Confidential	Minor change to the Overview

Confidentiality Status

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Web Address

developer.arm.com

Progressive terminology commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used terms that can be offensive. Arm strives to lead the industry and create change.

This document includes terms that can be offensive. We will replace these terms in a future issue of this document. If you find offensive terms in this document, please email **terms@arm.com**.

Contents

1 Overview

Arm NN is an inference engine for Arm CPUs, GPUs, and NPUs. Arm NN supports models created with TensorFlow Lite and ONNX frameworks.

This guide shows you how to download and configure Arm NN from start to finish so that you can use the TensorFlow Lite and ONNX frameworks with Arm NN. Alternatively, you can use our Debian package, which is the easiest way to install Arm NN and does not require you to build everything from the source. You can find more details about installing Arm NN using the Debian package in the `InstallationViaAptRepository.md` file in Arm NN. The guide does not currently cover how to include Arm NN as part of the TensorFlow Lite delegate runtime.

Arm NN deprecated support for the Quantizer, Caffe parser and TensorFlow parser in the 21.02 release. These will be removed in the 21.05 release.

1.1 Before you begin

Your platform or board must have:

- An Armv7-A or Armv8-A CPU, and optionally an Arm Mali GPU using the OpenCL driver
- At least 4GB of RAM
- At least 1GB of free storage space

Before you configure and build your environment, you must install the following tools on your platform or board:

- A Linux distribution
- Git. Arm tests Git 2.17.1. Other versions might work
- SCons. Arm tests SCons 2.4.1 on Ubuntu 16.04 and SCons 2.5.1 on Debian. Other versions might work
- CMake. Arm tests CMake 3.5.1 on Ubuntu 16.04 and CMake 3.7.2 on Debian. Other versions might work
- GNU Wget. Arm tests GNU Wget 1.17.1 built on GNU/Linux. Other versions might work
- UnZip. Arm tests UnZip 6.00. Other versions might work
- xxd. Arm tests xxd V1.10. Other versions might work

We estimate that you need about 3-4 hours to complete the instructions in this guide.

2 Download libraries

You must create a directory on the platform or board that you use for building Arm NN. This action is the default installation behavior for Arm NN. You can optionally add support for other frameworks. Use the following instructions to download the required libraries:

1. Open a new terminal session and enter the following commands on the command line to create a new directory called `armnn-dist`:

```
$ mkdir armnn-dist && cd armnn-dist
$ export BASEDIR=`pwd`
```

Some of the example commands that we use in this guide expect that the `$BASEDIR` environment variable is set correctly. So, if you use multiple terminal sessions, ensure that the variable is set correctly in each session.

2. Use the following commands to download the Arm Compute Library, Arm NN SDKs, and install the Boost package:

Arm Compute Library and Arm NN are closely developed and released. You must have corresponding versions of Arm Compute Library and Arm NN. For example, if you have the 20.11 release of Arm NN, you must use the 20.11 release of Arm Compute Library.

```
$ git clone https://github.com/Arm-software/ComputeLibrary.git
$ git clone https://github.com/Arm-software/armnn
$ sudo apt-get install libboost-all-dev
```

3. Use the following commands to download the required Git repositories and source bundles:

```
$ git clone -b v3.12.0 https://github.com/google/protobuf.git
$ git clone https://github.com/tensorflow/tensorflow.git
cd tensorflow/
git checkout fcc4b966f1265f466e82617020af93670141b009
$ wget -O flatbuffers-
1.12.0.tar.gz https://github.com/google/flatbuffers/archive/v1.12.0.tar.gz
$ tar xf flatbuffers-1.12.0.tar.gz
```

3 Build the Arm Compute Library

The Arm Compute Library is a machine learning library. It provides a set of functions that are optimized for both Arm CPUs and GPUs. Arm NN directly uses the Arm Compute Library to optimize the running of machine learning workloads on Arm CPUs and GPUs. To build the Arm Compute Library on your platform or board, complete the following steps:

1. Use the following command to open a terminal or bash screen and change directory to the Arm Compute Library directory:
```
$ cd $BASEDIR/ComputeLibrary
```
2. Compile the Arm Compute Library using SCons.
    a. To compile the Arm Compute Library on an Armv7-A based system, enter the following command:
```
$ scons extra_cxx_flags="-fPIC" benchmark_tests=0 validation_tests=0
```
    b. To compile the Arm Compute Library on an Armv8-A based system, enter the following command:
```
$ scons arch=arm64-v8a extra_cxx_flags="-fPIC" benchmark_tests=0
validation_tests=0
```
    If you want to enable benchmark tests, set `benchmark_tests` to 1. If you want to enable validation tests, set `validation_tests` to 1.

    If you have one, you can enable support for OpenCL on an Arm Mali GPU.

    If you want to support OpenCL for your Arm Mali GPU, add these arguments to the SCons command:
```
opencl=1 embed_kernels=1
```
    You can enable support for the Arm Neon architecture on supported Arm CPUs. To support the Arm Neon architecture, add this argument to your SCons command:
```
neon=1
```

4 Build the Boost library

Boost provides free, peer-reviewed, and portable C++ source libraries that work well with the C++ Standard Library.


Note

The 20.11 release of Arm NN removes Boost from the runtime. However, you must use Boost to run unit tests. You do not require the Boost library if you do not run unit tests. For more information on running unit tests, see the Test your build section.

If you have already installed `libboost-all-dev` package in your environment, you can skip this section and move to the Build the Google Protocol Buffers library section.

To build the Boost library, take the following steps:

1. Use the following commands to download the Boost library:
```
$ cd $BASEDIR

$ wget
https://dl.bintray.com/boostorg/release/1.64.0/source/boost_1_64_0.tar.bz2

$ tar xf boost_1_64_0.tar.bz2
```
2. Build the Boost library using the instructions in the **Boost getting started guide**. Arm has tested version 1.64, but other versions might work.
3. Include the following flags to build the Boost library:
```
link=static cxxflags=-fPIC --with-filesystem --with-test --with-log --with-
program_options --prefix=path/to/installation/prefix
```
   For example, to build version 1.64 of the Boost library, use the following commands:
```
$ cd $BASEDIR/boost_1_64_0

$ ./bootstrap.sh

$ ./b2 --build-dir=$BASEDIR/boost_1_64_0/build toolset=gcc link=static
cxxflags=-fPIC --with-filesystem --with-test --with-log --with-program_options
install --prefix=$BASEDIR/boost
```

5 Build the Google Protocol Buffers library

You must follow the steps in this section if you are building the environment for ONNX. If you are configuring the Arm NN build environment for TensorFlow Lite only, you can skip this section and move to the Generate the build dependencies for TensorFlow Lite section.

Protocol Buffers (protobuf) is a language-neutral, platform-neutral, and extensible mechanism that Google developed. You can use protobuf to serialize structured data.

Build protobuf using the C++ installation instructions that you can find on the **protobuf GitHub**. To build protobuf, take the following steps:

1. Use the following example code to build protobuf using a C++ installation:

```
git submodule update --init --recursive
./autogen.sh
mkdir aarch64_build
cd aarch64_build

CC=aarch64-linux-gnu-gcc \
CXX=aarch64-linux-gnu-g++ \
../configure --prefix=<path>/google/aarch64_pb_install \
--with-protoc=<path>/google/aarch64_pb_install/bin/protoc
```

   Arm has tested version 3.12.0 of Google protobuf. Other versions might work.

2. Copy the built program and its libraries and documentation to the correct locations using the following command:

```
make install -j16
```

6 Generate the build dependencies for TensorFlow Lite

FlatBuffers is another efficient, cross-platform, serialization library for C++. Google developed FlatBuffers for performance-critical applications. Flatbuffers generate TensorFlow Lite files to serialize their model data. As a result, Arm NN must use FlatBuffers to load and interpret the TensorFlow Lite files.

You require the FlatBuffers library for the Arm NN TensorFlow Lite parsers. To build the FlatBuffers library, use the instructions found in the **FlatBuffers Building Guide**.

The following example code shows you how you can build the FlatBuffers library and interpret the TensorFlow Lite format:

1. Use the following example code to build the FlatBuffers library:

```
$ cd $BASEDIR/flatbuffers-1.12.0

$ mkdir build

$ cd build
$ CXXFLAGS="-fPIC" cmake .. -DFLATBUFFERS_BUILD_FLATC=1 -
DCMAKE_INSTALL_PREFIX:PATH=$BASEDIR/flatbuffers


$ make
```

2. Use the following example code to interpret the TensorFlow Lite format:

```
$ mkdir tflite

$ cd tflite

$ cp $BASEDIR/tensorflow/tensorflow/lite/schema/schema.fbs .

$ $BASEDIR/flatbuffers-1.12.0/build/flatc -c --gen-object-api --reflect-types --
reflect-names schema.fbsS
```

7 Generate the build dependencies for ONNX

7.1 Generate the ONNX protobuf source files

The Arm NN ONNX parser requires the ONNX protobuf source files. Generate these source files based on the ONNX message formats defined in the `onnx.proto` library.

Use the following commands to generate the `onnx.pb.cc` and `onnx.pb.h` source files in the `$BASEDIR/onnx` directory ready for the Arm NN build:

```
$ cd $BASEDIR
$ export ONNX_ML=1 #To clone ONNX with its ML extension
$ git clone --recursive https://github.com/onnx/onnx.git
$ unset ONNX_ML
$ cd onnx
$ git checkout 553df22c67bee5f0fe6599cff60f1afc6748c635
$ export LD_LIBRARY_PATH=$BASEDIR/protobuf-host/lib:$LD_LIBRARY_PATH
$ $BASEDIR/protobuf-host/bin/protoc onnx/onnx.proto --proto_path=. --
proto_path=$BASEDIR/protobuf-host/include --cpp_out $BASEDIR/onnx
```

For more information and instructions, see the **ONNX GitHub**.

8 Build Arm NN

Configure the Arm NN SDK build using CMake. To configure the Arm NN SDK, you must change your directory to the Arm NN directory and enter the required parameters to CMake.

The following table lists the parameters that apply to all APIs:

Parameter	Description
`-DARMCOMPUTE_ROOT`	The location of your Arm Compute Library source files directory
`-DARMCOMPUTE_BUILD_DIR`	The location of your Arm Compute Library build directory
`-DBOOST_ROOT`	The directory you used for Boost. This parameter is the value you used for the prefix flag during Boost build process. Or, if you installed `libboost-all-dev` package in your environment instead of building the Boost library, use the `-DSHARED_BOOST=ON` flag.
`-DARMCOMPUTENEON=1`	Add this argument if you are supporting the Arm Neon architecture. We recommend enabling this option on the Raspberry Pi.
`-DARMCOMPUTECL=1`	Add this argument if you are supporting OpenCL.
`-DARMNNREF=1`	Add this argument if you want to include Arm NN reference support.

The following table lists the parameters for the ONNX parser. You must use these parameters with this parser:

Parameter	Description
`-DTF_GENERATED_SOURCES`	The location of your protobuf generated source files
`-DPROTOBUF_ROOT`	The location of your protobuf install directory
`-DBUILD_ONNX_PARSER=1`	Ensures the ONNX parser builds

The following table lists the parameters for the TensorFlow Lite parser. You must use these parameters with this parser:

Parameter	Description
`-DBUILD_TF_LITE_PARSER=1`	To ensure the TensorFlow Lite parser builds, include this argument.
`-DTF_LITE_GENERATED_PATH`	The location of the TensorFlow Lite schema directory
`-DFLATBUFFERS_ROOT`	The root directory where FlatBuffers is installed.
`-DFLATC_DIR`	The path to the schema compiler

For example, the following commands build the TensorFlow Lite parser:

```
$ cd $BASEDIR/armnn

$ mkdir build

$ cd build

$ cmake .. -DARMCOMPUTE_ROOT=$BASEDIR/ComputeLibrary \ -
DARMCOMPUTE_BUILD_DIR=$BASEDIR/ComputeLibrary/build \ -DSHARED_BOOT=ON \\ -
DTF_GENERATED_SOURCES=$BASEDIR/tensorflow-protobuf \ -DBUILD_TF_LITE_PARSER=1 \ -
DTF_LITE_GENERATED_PATH=$BASEDIR/tensorflow/tensorflow/lite/schema \ -
DFLATBUFFERS_ROOT=$BASEDIR/flatbuffers \ -DFLATC_DIR=$BASEDIR/flatbuffers-1.12.0/build
\ -DARMCOMPUTENEON=1 \ -DARMNNREF=1

$ make
```

If you want to include standalone sample dynamic backend tests, add the following arguments to enable the tests and dynamic backend path to the CMake command:

```
-DSAMPLE_DYNAMIC_BACKEND=1 -DDYNAMIC_BACKEND_PATHS=<the location of the sample dynamic backend>
```

Also, after building Arm NN, build the standalone sample dynamic backend using the guide in the following path, `$BASEDIR/armnn/src/dynamic/README.md#standalone-dynamic-backend-build`.

The executables are in the `armnn/build` directory.

Check that following binaries are in the `armnn/build` folder for all parsers:

- `libarmnn.so`
- `libarmnnBasePipeServer.so`
- `libtimelineDecoderJson.so`
- `libtimelineDecoder.so`

Check that the following binaries are in the `armnn/build` folder for the following specific parsers:

ONNX parser

`libarmnnOnnxParser.so`

TensorFlow Lite

`libarmnnTfLiteParser.so`

TensorFlow

`libarmnnTfParser.so`

9 Test your build

To check that your build of the Arm NN SDK is working correctly, you can run the unit tests. To do this check, change to the Arm NN build directory and enter `./UnitTests`.

If the tests are successful, the output from the tests ends with `*** No errors detected`.

For example:

```
$ ./UnitTests
Running 4154 test cases...
*** No errors detected
```

If some of the tests are unsuccessful, go back through the steps and check that all the commands have been entered correctly.

10 Related information

Here are some resources related to material in this guide:

- **Arm Compute Library**
- **Arm NN**
- Information on the **Armv7-A** and **Armv8-A** CPUs, and the **Arm Mali GPU**.
- **Arm Neon**
- **Boost**
- **OpenCL**
- **ONNX**
- **Protocol Buffers**
- **TensorFlow Lite**

11 Next steps

Now that you have built your environment and parsers for Arm NN, you are ready to begin programming with Arm NN and using it with your models.

Arm also provides Python bindings for our parsers and Arm NN. We call the result PyArmNN. Therefore, you have the convenience of writing your application in either C++ using the Arm NN library or Python using PyArmNN. You can find tutorials on how to use our parsers in our Arm NN documentation. The latest version can be found in the wiki section of the Arm NN Github repository. If you would like a further challenge, you can follow the Accelerating ML Inference on Raspberry Pi With PyArmNN tutorial to learn how to classify an image as Fire or Non-Fire.

Arm NN also provides the `armnnDelegate` library for accelerating certain TensorFlow Lite operators on Arm hardware. The library performs this acceleration by providing the TensorFlow Lite interpreter with an alternative implementation of the operators via its delegation mechanism. This library is our recommended way to accelerate TensorFlow Lite models.

Arm NN also provides a very basic example of how to use the Arm NN SDK API at `$BASEDIR/armnn/samples/SimpleSample.cpp`.