



Configure the Arm NN SDK build environment

Version 22.08

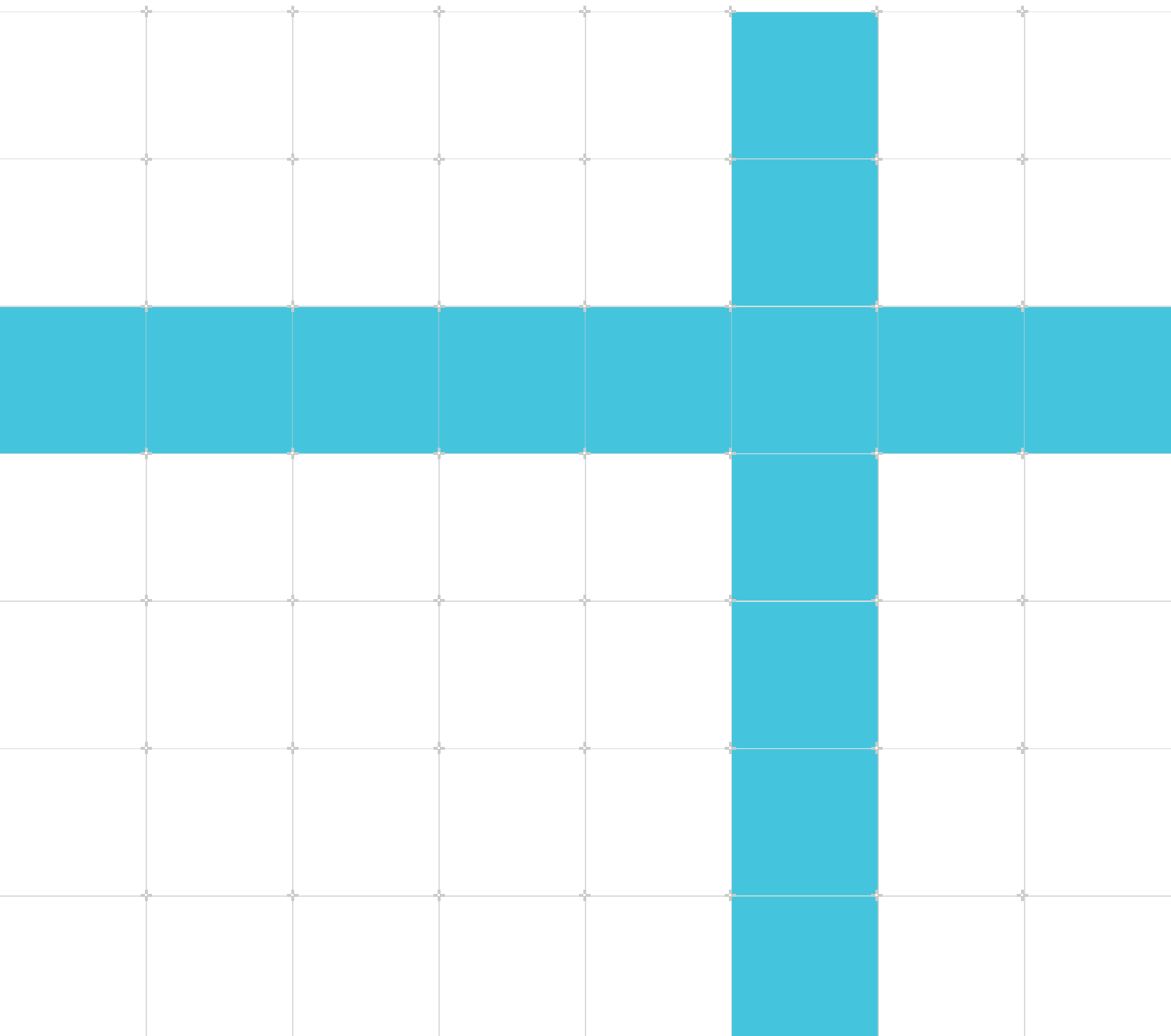
Tutorial

Non-Confidential

Copyright © 2021–2022 Arm Limited (or its affiliates).
All rights reserved.

Issue 01

102644_2208_01_en



Configure the Arm NN SDK build environment Tutorial

Copyright © 2021–2022 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	19 February 2021	Non-Confidential	First release for version 1.01
0100-02	18 May 2021	Non-Confidential	First release for version 1.02
2108-01	20 August 2021	Non-Confidential	First release for version 21.08
2108-02	21 October 2021	Non-Confidential	Second release for version 21.08
2202-01	16 February 2022	Non-Confidential	First release for version 22.02
2208-01	30 August 2022	Non-Confidential	First release for version 22.08

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at <https://www.arm.com/company/policies/trademarks>.

Copyright © 2021–2022 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

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.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on <https://support.developer.arm.com>

To provide feedback on the document, fill the following survey: <https://developer.arm.com/documentation-feedback-survey>.

Inclusive language commitment

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

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Introduction.....	6
1.1 Conventions.....	6
1.2 Useful resources.....	7
1.3 Other information.....	7
2. Overview.....	8
2.1 Before you begin.....	8
3. Download libraries.....	10
4. Build the Arm Compute Library.....	11
5. Build the Google Protocol Buffers library.....	12
6. Generate the build dependencies for TensorFlow Lite.....	13
7. Generate the build dependencies for ONNX.....	14
7.1 Generate the ONNX protobuf source files.....	14
8. Build Arm NN.....	15
9. Test your build.....	17
10. Related information.....	18
11. Next steps.....	19
A. Revisions.....	20

1. Introduction

1.1 Conventions

The following subsections describe conventions used in Arm documents.





Glossary



The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm® Glossary for more information: developer.arm.com/glossary.

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
<i>italic</i>	Citations.
bold	Interface elements, such as menu names. Terms in descriptive lists, where appropriate.
monospace	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
monospace <u>underline</u>	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: <pre>MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2></pre>
SMALL CAPITALS	Terms that have specific technical meanings as defined in the <i>Arm® Glossary</i> . For example, IMPLEMENTATION DEFINED , IMPLEMENTATION SPECIFIC , UNKNOWN , and UNPREDICTABLE .
 Caution	Recommendations. Not following these recommendations might lead to system failure or damage.
 Warning	Requirements for the system. Not following these requirements might result in system failure or damage.
 Danger	Requirements for the system. Not following these requirements will result in system failure or damage.
 Note	An important piece of information that needs your attention.

Convention	Use
 Tip	A useful tip that might make it easier, better or faster to perform a task.
 Remember	A reminder of something important that relates to the information you are reading.

1.2 Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at developer.arm.com/documentation. Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.



Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at <http://www.adobe.com>

1.3 Other information

See the Arm website for other relevant information.

- [Arm® Developer](#).
- [Arm® Documentation](#).
- [Technical Support](#).
- [Arm® Glossary](#).

2. Overview

The steps to build Arm NN in this guide are being replaced by the [Arm NN Build Tool](#) and will be removed soon.



Use the [Arm NN Build Tool](#) as a user-friendly way to build Arm NN and its dependencies.

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.

2.1 Before you begin

The steps to build Arm NN in this guide are being replaced by the [Arm NN Build Tool](#) and will be removed soon.



Use the [Arm NN Build Tool](#) as a user-friendly way to build Arm NN and its dependencies.

This guide requires your platform or board to meet certain requirements.

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 18.04 and SCons 2.5.1 on Debian. Other versions might work

- CMake. Arm tests CMake 3.5.1 on Ubuntu 18.04 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.

3. 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.

Procedure

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. Enter the following commands to download the Arm Compute Library and Arm NN SDKs:

```
$ git clone https://github.com/Arm-software/ComputeLibrary.git  
$ git clone https://github.com/Arm-software/armnn
```



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.

3. Enter 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 a4dfb8d1a71385bd6d122e4f27f86dcebb96712d  
$ 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
```

4. 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.

Procedure

To build the Arm Compute Library on your platform or board:

1. Enter 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 arch=armv7a 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 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
```

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.

Procedure

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
```

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](#).

Procedure

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 all install
```

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.fbs
```

7. Generate the build dependencies for ONNX

The Arm NN ONNX parser requires the ONNX protobuf source files.

7.1 Generate the ONNX protobuf source files

Generate the ONNX protobuf 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

The steps to build Arm NN in this guide are being replaced by the [Arm NN Build Tool](#) and will be removed soon.



Use the [Arm NN Build Tool](#) as a user-friendly way to build Arm NN and its dependencies.

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:

Table 8-1: API parameters

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.
-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:

Table 8-2: ONNX parser parameters

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:

Table 8-3: TensorFlow Lite parser parameters

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 \
-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.

The executables are in the armnn/build directory.

Check that the following binaries are in the armnn/build folder:

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


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
[doctest] doctest version is "2.4.6"
[doctest] run with "--help" for options
=====
[doctest] test cases:      4569 |    4569 passed | 0 failed | 0 skipped
[doctest] assertions: 806003 | 806003 passed | 0 failed |
[doctest] Status: SUCCESS!
```

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](#)
- [OpenCL](#)
- [ONNX](#)
- [Protocol Buffers](#)
- [TensorFlow Lite](#)

Other Arm resources:

- [Arm Community](#) - ask development questions and find articles and blogs on specific topics from Arm experts.
- [Arm NN Github](#) - raise queries or issues associated with the Arm NN how-to guides.
- [Arm NN Product Documentation](#) - find out more about the latest Arm NN features.

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`.

Appendix A Revisions

This appendix describes the technical changes between released issues of this book.

Table A-1: First release for version 1.01

Change	Location
First release	—

Table A-2: First release for version 1.02

Change	Location
Minor change to the Overview	Overview

Table A-3: First release for version 21.08

Change	Location
Removes the Boost library dependency	—

Table A-4: Second release for version 21.08

Change	Location
Adds additional resources	Related information
Updates the SHA in the Tensorflow repository	Download libraries

Table A-5: First release for 22.02

Change	Location
Removes redundant note	Overview
Adds another library	Build Arm NN
Updates the test output example	Test your build
Updates the code to build the FlatBuffers library	Generate the build dependencies for TensorFlow Lite

Table A-6: First release for 22.08

Change	Location
Adds note about the Arm NN Build Tool.	Various