



Build Arm Cortex-M voice assistant with Google TensorFlow Lite

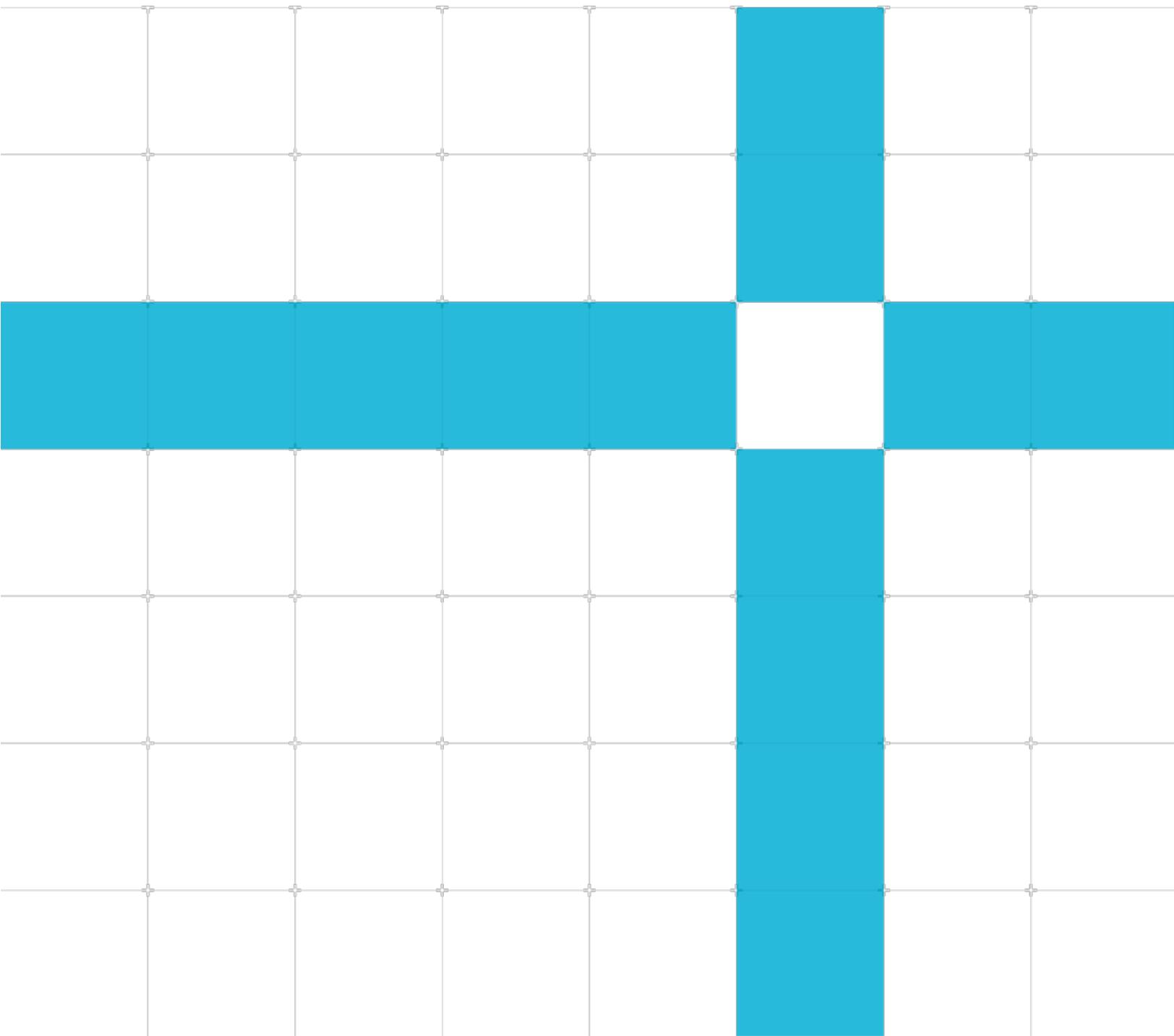
Non-Confidential

Issue 03

Copyright © 2019-2020 Arm Limited (or its affiliates).

ARM062-948681440-3282

All rights reserved.



Build Arm Cortex-M voice assistant with Google TensorFlow Lite

Copyright © 2019-2020 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
01	23 July 2019	Non-Confidential	First release
02	01 May 2020	Non-Confidential	Adds updates to the TensorFlow Lite codebase
03	30 November 2020	Non-Confidential	Adds CMSIS-NN support

Confidential Proprietary Notice

This document is **CONFIDENTIAL** and any use by you is subject to the terms of the agreement between you and Arm or the terms of the agreement between you and the party authorized by Arm to disclose this document to you.

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: **(i)** for the purposes of determining whether implementations infringe any third party patents; **(ii)** for developing technology or products which avoid any of Arm's intellectual property; or **(iii)** as a reference for modifying existing patents or patent applications or creating any continuation, continuation in part, or extension of existing patents or patent applications; or **(iv)** for generating data for publication or disclosure to third parties, which compares the performance or functionality of the Arm technology described in this document with any other products created by you or a third party, without obtaining Arm's prior written consent.

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, third party 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.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. 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 <http://www.arm.com/company/policies/trademarks>.

Copyright © 2019-2020 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20348)

Confidentiality Status

Non-Confidential 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.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement

prevails over and supersedes the conflicting provisions of these terms. 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 <http://www.arm.com/company/policies/trademarks>.

Copyright © 2019-2020 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)

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

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 contact terms@arm.com.

Contents

1 Introduction	7
1.1 Product revision status	7
1.2 Intended audience	7
1.3 Conventions	7
1.3.1 Glossary	7
1.3.2 Typographical conventions	8
1.4 Feedback	9
1.4.1 Feedback on this product	9
1.4.2 Feedback on content	9
2 Overview	10
2.1 About TensorFlow Lite	10
2.1.1 TensorFlow Lite - video	10
3 Getting started	11
3.1 Before you begin	11
3.2 Getting started	11
4 Download and build the sample application	13
4.1 Install Arm toolchain and Mbed CLI	13
4.2 Build and compile micro speech example	13
4.3 CMSIS-NN	15
5 Project structure	17
5.1 Convolutional neural networks	17
5.2 Feature generation with Fast Fourier transform	18
5.3 Recognition and windowing	19
5.3.1 Interpreting the results	19
6 Deploy the sample to your STM32F7	20
6.1 Test keyword spotting	20
7 Retrain the machine learning model	22
7.1 Convert the model	22

7.2 Modify the device code.....	23
8 Troubleshooting.....	26
8.1 Mbed CLI issues or Error: collect2: error: ld returned 1 exit status	26
8.2 Error: Prompt wrapping around line.....	26
8.3 Error: "Requires make version 3.82 or later (current is 3.81)"	26
8.4 Error: -bash: mbed: command not found.....	27
8.5 "sed" errors while generating micro speech project	27
9 Next steps.....	28
10 Supplementary information: model training	29
10.1 Prepare to build TensorFlow	29
10.2 Train the model.....	30
10.3 Freeze the model.....	30
10.4 Convert the model to the TensorFlow Lite format	31
Appendix A Revisions.....	32

1 Introduction

1.1 Product revision status

The r_{xy} identifier indicates the revision status of the product described in this book, for example, r1p2, where:

r_x

Identifies the major revision of the product, for example, r1.

p_y

Identifies the minor revision or modification status of the product, for example, p2.

1.2 Intended audience

This guide is intended for IoT developers who want to learn how to best perform machine learning inferences on an Arm Cortex-M microcontroller.

1.3 Conventions

The following subsections describe conventions used in Arm documents.

1.3.1 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: <https://developer.arm.com/glossary>.

1.3.2 Typographical conventions

Convention	Use
<i>italic</i>	Introduces citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
monospace	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
monospace bold	Denotes language keywords when used outside example code.
monospace <u>underline</u>	Denotes 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	Used in body text for a few terms that have specific technical meanings, that are defined in the Arm® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.
 Caution	This represents a recommendation which, if not followed, might lead to system failure or damage.
 Warning	This represents a requirement for the system that, if not followed, might result in system failure or damage.
 Danger	This represents a requirement for the system that, if not followed, will result in system failure or damage.
 Note	This represents an important piece of information that needs your attention.
 Tip	This represents a useful tip that might make it easier, better or faster to perform a task.
 Remember	This is a reminder of something important that relates to the information you are reading.

1.4 Feedback

Arm welcomes feedback on this product and its documentation.

1.4.1 Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

1.4.2 Feedback on content

If you have comments on content, send an email to errata@arm.com and give:

- The title Build Arm Cortex-M voice assistant with Google TensorFlow Lite.
- The number ARM062-948681440-3282.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.



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

2 Overview



As an IoT developer, you might think of machine learning as a server-side technology. In the traditional view, sensors on your device capture data and send it to the cloud, where Machine Learning (ML) models on hefty machines make sense of it. A network connection is obligatory, and you are going to expect some latency, not to mention hosting costs.

But more and more, developers want to deploy their ML models to the edge, on IoT devices themselves. If you bring ML closer to your sensors, you remove your reliance on a network connection, and you can achieve much lower latency without a round trip to the server.

This is especially exciting for IoT because less network utilization means lower power consumption. Also, you can better guarantee the security and privacy of your users because it does not require you to send data back to the cloud unless you know for sure that it is relevant.

In the following guide, you will learn how you can perform machine learning inference on an Arm Cortex-M microcontroller with [TensorFlow Lite for Microcontrollers](#).

2.1 About TensorFlow Lite

[TensorFlow Lite](#) is a set of tools for running machine learning models on-device. TensorFlow Lite powers billions of mobile app installs, including Google Photos, Gmail, and devices made by Nest and Google Home.

With the launch of TensorFlow Lite for Microcontrollers, developers can run machine learning inference on extremely low-powered devices, like the Cortex-M microcontroller series. Watch the following video to learn more about the announcement:

2.1.1 TensorFlow Lite - video

[TensorFlow Lite \(TF Dev Summit '19\)](#)

3 Getting started

3.1 Before you begin

Here is what you require to complete the guide:

- A computer that supports [Mbed CLI](#)
- An [STM32F7 discovery kit](#) board
- A mini-USB cable
- Python 2.7. Using [pyenv](#) is recommended to manage Python versions.
- For Windows users, install Ubuntu 18.04 LTS in a VirtualBox. Refer to the following videos to set up:
 - [How to install VirtualBox 6.0.10 on Windows 10](#)
 - [How to install Ubuntu 18.04 on VirtualBox in Windows 10](#)

3.2 Getting started

TensorFlow Lite for Microcontrollers supports [several devices](#) out of the box, and is relatively easy to extend to new devices. For this guide, we focus on the [STM32F7 discovery kit](#).



We deploy a sample application that uses the microphone on the STM32F7 and a TensorFlow machine learning model to detect the words “yes” and “no”.

To do this, we show you how to complete the following steps:

1. Download and build the sample application
2. Deploy the sample to your STM32F7
3. Make some code changes to utilize the LCD display on the board

4. Use new trained models to recognize different words

4 Download and build the sample application

4.1 Install Arm toolchain and Mbed CLI

1. Download [Arm cross compilation](#) toolchain. Select the correct toolchain for the OS that your computer is running. For Windows users, if you have already set up the Linux virtual environment, install the toolchain there.
2. To build and deploy the application, we use the [Mbed CLI](#). We recommend that you install Mbed CLI with our installer. If you need more customization, you can perform a manual install. Although this is not recommended.

If you do not already have Mbed CLI installed, download the installer:

Mac installer

3. After Mbed CLI is installed, tell Mbed where to find the Arm embedded toolchain using the following command:

```
mbed config -G GCC_ARM_PATH <path_to_your_arm_toolchain>/bin
```



We recommend running the following commands from inside the Mbed CLI terminal that gets launched with the Mbed CLI Application. This is because it is much quicker to set up, because it resolves all your environment dependencies automatically.



4.2 Build and compile micro speech example

Navigate to the directory where you keep code projects. Run the following command to download TensorFlow Lite source code.

```
git clone https://github.com/tensorflow/tensorflow.git
```

While you wait for the project to download, let us explore the project files on [GitHub](#) and learn how this TensorFlow Lite for Microcontrollers example works.

The code samples audio from the microphone on the STM32F7. The audio is run through a Fast Fourier transform to create a spectrogram. The spectrogram is then fed into a pre-trained machine learning model. The model uses a [convolutional neural network](#) to identify whether the sample represents either the command “yes” or “no”, silence, or an unknown input. We explore how this works in more detail later in the guide.

The micro speech sample application is in the `tensorflow/lite/micro/examples/microspeech` directory.

Here are descriptions of some interesting source files:

- [disco_f746ng/audio_provider.cc](#) captures audio from the microphone on the device.
- [micro_features/micro_features_generator.cc](#): uses a Fast Fourier transform to create a spectrogram from audio.
- [micro_features/tiny_conv_micro_features_model_data.cc](#). This file is the machine learning model itself, represented by a large array of unsigned char values.
- [command_responder.cc](#) is called every time a potential command has been identified.
- [main.cc](#). This file is the entry point for the Mbed program, which runs the machine learning model using TensorFlow Lite for Microcontrollers.

After the project has downloaded, you can run the following commands to navigate into the project directory and build it:

```
cd tensorflow

make -f tensorflow/lite/micro/tools/make/Makefile TARGET=mbed TAGS="CMSIS-NN
disco_f746ng" generate_micro_speech_mbed_project
```

These commands create an Mbed project folder in `tensorflow/lite/micro/tools/make/gen/mbed_cortex-m4/prj/micro_speech/mbed`.

The micro speech source code of the generated Mbed project is in `tensorflow/lite/micro/tools/make/gen/mbed_cortex-m4/prj/micro_speech/mbed/tensorflow/lite/micro/examples/micro_speech`. If you must make further changes to the source code after generating the Mbed project, change the source code in the `micro_speech` folder.

If you encounter the error message "`Tensorflow/lite/micro/tools/make/Makefile:2 *** "Require make version 3.82 or later (current 3.81)"`", please refer to the [Troubleshooting](#) section.

```
cd tensorflow/lite/micro/tools/make/gen/mbed_cortex-m4/prj/micro_speech/mbed

mbed config root .

mbed deploy
```

TensorFlow requires C++ 11, so you must update your profiles to reflect this. Here is a short Python command that does that. Run it from the command line:

```
python -c 'import fileinput, glob;

for filename in glob.glob("mbed-os/tools/profiles/*.json"):

    for line in fileinput.input(filename, inplace=True):

        print line.replace("\-std=gnu++98\"", "\-std=c++11\"", "\-fpermissive\"")'
```

After that setting is updated, you can compile:

```
mbed compile -m DISCO_F746NG -t GCC_ARM
```

4.3 CMSIS-NN

In the example above, we compiled our project with a `TAGS="cmsis-nn"` flag, which enables kernel optimization with CMSIS-NN library. Following are some CMSIS-NN acceleration techniques.

The CMSIS-NN library provides optimized neural network kernel implementations for all Arm Cortex-M processors, ranging from Cortex-M0 to Cortex-M55. The library utilizes the capabilities of the processor, such as DSP and M-Profile Vector (MVE) extensions, to enable the best possible performance.

The STMicroelectronics F746NG Discovery board we use in the guide is powered by Arm Cortex-M7, which supports DSP extensions. That enables the optimized kernels to perform multiple operations in one cycle using SIMD (Single Instruction Multiple Data) instructions. Another optimization technique used by the CMSIS-NN library is loop unrolling. These techniques combined significantly accelerate kernel performance on Arm MCUs.

In the following example, we use the SIMD instruction, SMLAD (Signed Multiply with Addition), together with loop unrolling to perform a matrix multiplication $y = a * b$, where

```
a = [1, 2]
```

and

```
b = [3, 5
     4, 6]
```

a, b are 8-bit values and y is a 32-bit value. With regular C, the code would look something like the following code:

```
for (i=0; i<2; ++i)
    for (j=0; j<2; ++j)
        y[i] += a[j] * b[j][i]
```

However, using loop unrolling and SIMD instructions, the loop looks like the following code:

```
a_operand = a[0] | a[1] << 16 // put a[0], a[1] into one variable
for(i=0; i<2; ++i)
    b_operand = b[0][i] | b[1][i] << 16 // vice versa for b
    y[i] = __SMLAD(a_operand, b_operand, y[i])
```

This code saves cycles due to fewer for-loop checks since `__SMLAD` performs two multiply and accumulate operations in one cycle.

With CMSIS-NN enabled, we observed a 16x performance uplift in the micro speech inference time.

5 Project structure

While the project builds, we can look in more detail at how it works.

5.1 Convolutional neural networks

Convolutional networks are a type of deep neural network. These networks are designed to identify features in multidimensional vectors. The information in these vectors is contained in the relationships between groups of adjacent values.

These networks are usually used to analyze images. An image is a good example of the multidimensional vectors described previously, in which a group of adjacent pixels might represent a shape, a pattern, or a texture. During training, a convolutional network can identify these features and learn what they represent. The network can learn how simple image features, like lines or edges, fit together into more complex features, like an eye, or an ear. The network can also learn, how those features are combined to form an input image, like a photo of a human face. This means that a convolutional network can learn to distinguish between different classes of input image, for example a photo of a person and a photo of a dog.

While they are often applied to images, which are 2D grids of pixels, a convolutional network can be used with any multidimensional vector input. In the example we build in this guide, a convolutional network has been trained on a spectrogram that represents 1 second of audio bucketed into multiple frequencies.

The following image is a visual representation of the audio. The network in our sample has learned which features in this image come together to represent a "yes", and which come together to represent a "no".



Spectrogram for

"yes"(data)

Spectrogram for "no" (data)

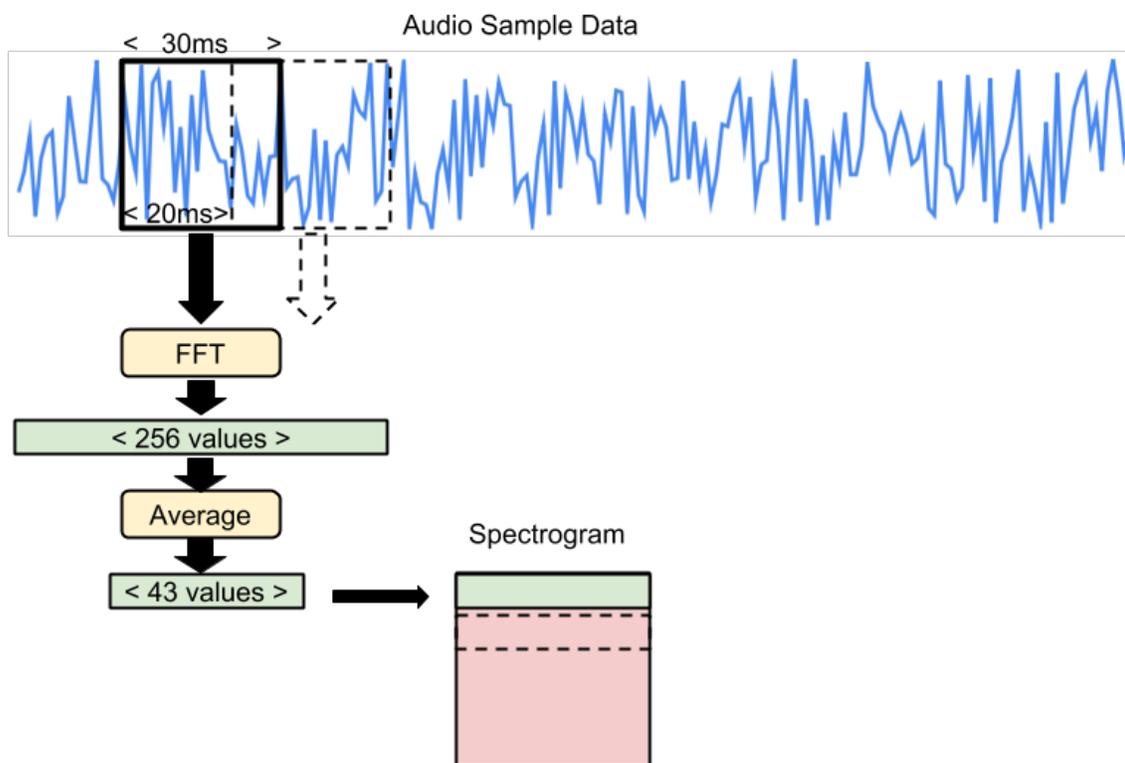
To generate this spectrogram, we use an interesting technique that is described in the next section.

5.2 Feature generation with Fast Fourier transform

In our code, each spectrogram is represented as a 2D array, with 43 columns and 49 rows. Each row represents a 30ms sample of audio that is split into 43 frequency buckets.

To create each row, we run a 30ms slice of audio input through a Fast Fourier transform. Fast Fourier transform analyzes the frequency distribution of audio in the sample and creates an array of 256 frequency buckets, each with a value from 0 to 255. These buckets are averaged together into groups of 6, leaving us with 43 buckets. The code in the file [micro_features/micro_features_generator.cc](#) performs this action.

To build the entire 2D array, we combine the results of running the Fast Fourier transform on 49 consecutive 30ms slices of audio, with each slice overlapping the last by 10ms. The following diagram should make this clearer:



You can see how the 30ms sample window moves forward by 20ms each time until it has covered the full one-second sample. The resulting spectrogram is passed into the convolutional model.

5.3 Recognition and windowing

The process of capturing one second of audio and converting it into a spectrogram leaves us with something that our ML model can interpret. The model outputs a probability score for each category it understands (yes, no, unknown, and silence). The probability score indicates whether the audio is likely to belong to that category.

The model was trained on one-second samples of audio. In the training data, the word “yes” or “no” is spoken at the start of the sample, and the entire word is contained within that one-second. However, when this code is running, there is no guarantee that a user will begin speaking at the very beginning of our one-second sample.

If the user starts saying “yes” at the end of the sample instead of the beginning, the model might not be able to understand the word. This is because the model uses the position of the features within the sample to help predict which word was spoken.

To solve this problem, our code runs inference as often as it can, depending on the speed of the device, and averages all results within a rolling 1000ms window. The code in the file [recognize_commands.cc](#) performs this action. When the average for a given category in a set of predictions goes above the threshold, as defined in [recognize_commands.h](#), we can assume a valid result.

5.3.1 Interpreting the results

The `RespondToCommand` method in [command_responder.cc](#) is called when a command has been recognized. Currently, this results in a line being printed to the serial port. Later in this guide, we modify the code to display the result on the screen.

The arguments passed to `RespondToCommand` contain interesting information about the recognition result. For example, `score` contains a number that indicates the probability that the result is correct. You might want to modify the code so that the display varies based on this information.

6 Deploy the sample to your STM32F7

In the previous section of this guide, we explained the build process for a keyword spotting example application.

Now that the build has completed, we look in this section of the guide at how to deploy the binary to the STM32F7 and test to see if it works.

First, plug in your STM32F7 board via USB. The board should show up on your machine as a USB mass storage device. Copy the binary file that we built earlier to the USB storage.

If you have skipped the previous steps, download the [binary file](#) to proceed.

Use the following command:

```
cp ./BUILD/DISCO_F746NG/GCC_ARM/mbed.bin /Volumes/DIS_F746NG/
```

Depending on your platform, the exact copy command and paths can vary. When you have copied the file, the LEDs on the board should start flashing, and the board eventually reboots with the sample program running.

6.1 Test keyword spotting

The program outputs recognition results to its serial port. To see the output of the program, we must establish a serial connection with the board at 9600 baud.

The USB UART of the board shows up as `/dev/tty.usbmodemXXXXXX`. We can use 'screen' to access the serial console. Although 'screen' is not installed on Linux by default, you can use `apt-get install screen` to install the package.

Run the following command in a separate terminal:

```
screen /dev/tty.usbmodemXXXXXX 9600
```

Try saying the word "yes" several times. You should see some output like the following:

```
Heard yes (208) @116448ms
```

```
Heard unknown (241) @117984ms
```

```
Heard no (201) @124992ms
```

The LCD displays "Heard yes!", as you can see in the following image:



Congratulations! You are now running a machine learning model that can recognize keywords on an Arm Cortex-M7 microcontroller, directly on your STM32F7.

It is easy to change the behavior of our program, but is it difficult to modify the machine learning model itself? The answer is no, and the next section of this guide, [Retrain the machine learning model](#), will show you how.

7 Retrain the machine learning model

The model that we are using for speech recognition was trained on a dataset of one-second spoken commands called the [Speech Commands Dataset](#). The dataset includes examples of the following ten different words:

yes, no, up, down, left, right, on, off, stop, go

While the model in this sample was originally trained to recognize “yes” and “no”, the TensorFlow Lite for Microcontrollers source contains scripts that make it easy to retrain the model to classify any other combination of these words.

We are going to use another pre-trained model to recognize “up” and “down”, instead. If you are interested in the full workflow including the training of the model refer to the [Supplementary information: model training](#) section of this guide.

To build our new ML application, we follow these steps:

1. Download a pretrained model that has been trained and frozen using TensorFlow.
2. Look at how the TensorFlow model gets converted to the TensorFlow Lite format.
3. Convert the TensorFlow Lite model into a C source file.
4. Modify the code and deploy to the device.



Building TensorFlow and training the model each takes a couple of hours on a typical computer. We do not perform this build at this stage. For a full guide on how to do this, refer to the [Supplementary information: model training](#) section in this guide.

7.1 Convert the model

Starting from the trained model to obtain a converted model that can run on the controller itself, we must run a conversion script: the [TensorFlow Lite converter](#). This tool uses clever tricks to make our model as small and efficient as possible, and to convert it to a TensorFlow Lite FlatBuffer. To reduce the size of the model, we used a technique called [quantization](#). All weights and activations in the model get converted from 32-bit floating point format to an 8-bit and fixed-point format, as you can see in the following command:

```
root@dc02b9d87183:/tensorflow_src# bazel run tensorflow/lite/toco:toco -- --input_file=/tmp/tiny_conv.pb --output_file=/tmp/tiny_conv.tflite --input_shapes=1,49,40,1 --input_arrays=Reshape_1 --output_arrays='labels_softmax' --inference_type=QUANTIZED_UINT8 --mean_values=0 --std_values=9.8077
INFO: Options provided by the client:
```

This conversion does not only reduce the size of the network, but also avoids floating point computations that are more computationally expensive.

To save time, we skip this step and instead download the [tiny_conv.tflite](#).

The final step in the process is to convert this model into a C file that we can drop into our Mbed project.

To do this conversion, we use a tool called `xxd`. Issue the following command:

```
xxd -i tiny_conv.tflite > tiny_conv_micro_features_model_data.cc
```

Next, we must update `tiny_conv_micro_features_model_data.cc` so that it is compatible with our code. First, open the file. The top two lines should look similar to the following code, although the exact variable name and hex values may be different:

```
unsigned char tiny_conv_tflite[] = {  
  
    0x18, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x0e, 0x00,
```

You must add the `include` from the following snippet and change the variable declaration without changing the hex values:

```
#include  
"tensorflow/lite/micro/examples/micro_speech/micro_features/tiny_conv_micro_features_model_data.h"  
  
const unsigned char g_tiny_conv_micro_features_model_data[] = {  
  
    0x18, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x0e, 0x00,
```

Next, go to the very bottom of the file and find the unsigned int variable.

```
unsigned int tiny_conv_tflite_len = 18216;
```

Change the declaration to the following code, but do not change the number assigned to it, even if your number is different from the one in this guide.

```
const int g_tiny_conv_micro_features_model_data_len = 18216;
```

Finally, save the file, then copy the `tiny_conv_micro_features_model_data.cc` file into the `tensorflow/tensorflow/lite/micro/tools/make/gen/mbed_cortex-m4/prj/micro_speech/mbed/tensorflow/lite/micro/examples/micro_speech/micro_features` directory.

7.2 Modify the device code

If you build and run your code now, your device should respond to the words “up” and “down”. However, the code was written to assume that the words are “yes” and “no”. Let us update the references and the user interface so that the appropriate words are printed.

First, go to the following directory:

```
tensorflow/lite/micro/examples/micro_speech/
```

and open the file:

```
micro_features/micro_model_settings.cc
```

You see the following category labels:

```
const char* kCategoryLabels[kCategoryCount] = {  
  
    "silence",  
  
    "unknown",  
  
    "yes",  
  
    "no",  
  
};
```

The code uses this array to map the output of the model to the correct value. Because we specified our `wanted_words` as “up, down” in the training script, we should update this array to reflect these words in the same order. Edit the code so it appears as follows:

```
const char* kCategoryLabels[kCategoryCount] = {  
  
    "silence",  
  
    "unknown",  
  
    "up",  
  
    "down",  
  
};
```

Next, we update the code in `command_responder.cc` to reflect these new labels, modifying the if statements and the `DisplayStringAt` call:

```
void RespondToCommand(tflite::ErrorReporter* error_reporter,  
  
                    int32_t current_time, const char* found_command,  
  
                    uint8_t score, bool is_new_command) {  
  
    if (is_new_command) {  
  
        error_reporter->Report("Heard %s (%d) @%dms", found_command, score,  
  
                               current_time);  
  
        if (strcmp(found_command, "up") == 0) {
```

```
    lcd.Clear(0xFF0F9D58);

    lcd.DisplayStringAt(0, LINE(5), (uint8_t *)"Heard up", CENTER_MODE);

} else if(strcmp(found_command, "down") == 0) {

    lcd.Clear(0xFFDB4437);

    lcd.DisplayStringAt(0, LINE(5), (uint8_t *)"Heard down", CENTER_MODE);

} else if(strcmp(found_command, "unknown") == 0) {

    lcd.Clear(0xFFFF4B400);

    lcd.DisplayStringAt(0, LINE(5), (uint8_t *)"Heard unknown", CENTER_MODE);

} else {

    lcd.Clear(0xFF4285F4);

    lcd.DisplayStringAt(0, LINE(5), (uint8_t *)"Heard silence", CENTER_MODE);

}

}

}
```

Now that we have updated the code, go back to the `mbed` directory:

```
cd <path_to_tensorflow>/tensorflow/lite/micro/tools/make/gen/mbed_cortex-  
m4/prj/micro_speech/mbed
```

and run the following command to rebuild the project:

```
mbed compile -m DISCO_F746NG -t GCC_ARM
```

Finally, copy the binary to the USB storage of the device, using the same method that you used earlier. You should now be able to say “up” and “down” to update the display.

8 Troubleshooting

We have found some common errors that users face and have listed them here to help you get started with your application as quickly as possible.

8.1 Mbed CLI issues or Error: collect2: error: ld returned 1 exit status

Purge the cache with the following command:

```
mbd cache purge
```

You probably also have a stale BUILD folder. Clean up your directory and try again:

```
rm -rf BUILD
```

8.2 Error: Prompt wrapping around line

If your terminal wraps your text as shown here:

```
[-m4/prj/micro_speech/mbd $ L/tensorflow/tensorflow/lite/experimental/micro/tools/make/gen/mbd_cortex-  
[-m4/prj/micro_speech/mbd $ lsensorflow/tensorflow/lite/experimental/micro/tools/make/gen/mbd_cortex-
```

In your terminal type:

```
export PS1='\u@\h: '
```

For a more minimalist approach, type:

```
export PS1='> '
```

8.3 Error: "Requires make version 3.82 or later (current is 3.81)"

If you encounter this error, install the brew and make by typing the following code:

```
ruby -e "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

```
brew install make
```



On a Mac, you might have to use `gmake` instead of `make`, to run your commands.

8.4 Error: `-bash: mbed: command not found`

If you encounter this error, try the following fixes.

For Mac:

We recommend using the [installer](#) and running the downloaded Mbed CLI App. This app automatically launches a shell with all the dependencies solved for you.

If installed manually, make sure to follow these [instructions](#).



8.5 "sed" errors while generating micro speech project

If you encounter the following error while in the [Build and compile micro speech example](#) section while generating micro speech project:

```
sed: 1: "tensorflow/lite/micro/t ...": undefined label  
'tensorflow/lite/micro/tools/make/downloads/cmsis/CMSIS/RTOS2/Include/cmsis_os2.h'  
sed: 1: "tensorflow/lite/micro/t ...": undefined label  
'tensorflow/lite/micro/tools/make/downloads/cmsis/CMSIS/RTOS2/Include/os_tick.h'  
sed: 1: "tensorflow/lite/micro/t ...": undefined label  
'tensorflow/lite/micro/tools/make/downloads/cmsis/CMSIS/RTOS2/Source/os_tick_gtim.c'  
sed: 1: "tensorflow/lite/micro/t ...": undefined label  
'tensorflow/lite/micro/tools/make/downloads/cmsis/CMSIS/RTOS2/Source/os_systick.c'
```

add `"LC_CTYPE=C"` tag when compiling the project:

```
LC_CTYPE=C make -f tensorflow/lite/micro/tools/make/Makefile TARGET=mbed  
TAGS="disco_f746ng" generate_micro_speech_mbed_project
```

9 Next steps

Until recently, AI on tiny microcontrollers was deemed impossible. Now, thanks to tools like Mbed and TensorFlow Lite for Microcontrollers, it is not only possible, it is easy and within the reach of every open-source software developer, maker, and start-up.

Soon, more optimized low-level kernels will be available as part of the **CMSIS-NN** open-source project. These kernels allow developers to leverage Single Instruction Multiple Data (SIMD) instructions and receive an uplift in performance. **SIMD instructions** are available in Arm Cortex-M4, Cortex-M7, Cortex-M33, and Cortex-M35P processors.

Now that you have implemented your first machine learning application on a microcontroller, it is time to get creative.

To keep learning about ML with Arm and TensorFlow, here are some additional resources:

- View another in-depth end-to-end **TensorFlow from training to deployment guide**
- Read a white paper on CMSIS-NN: **Machine learning on Arm Cortex-M Microcontrollers**
- Learn how to use the **OpenMV camera for ML** applications
- Explore **Arm NN** for ML on other Arm processors and GPUs

Share your projects using the hashtag #TinyML, and tag @Arm.

10 Supplementary information: model training

10.1 Prepare to build TensorFlow

The code that we used to train our voice model currently depends on some experimental operations that are only available when building TensorFlow from source. We must build TensorFlow.

The easiest way to build TensorFlow from source is to use Docker. Docker is a tool that enables you to run tasks on a virtual machine that is isolated from the rest of your computer, which makes dependency management easier. TensorFlow provides a custom docker image that can be used to build the toolchain from source.

The first step is to follow the instructions to [install Docker](#).

When Docker is installed, run the following command to test that it works:

```
docker run hello-world
```

You should see a message starting with “Hello from Docker!”.

When you have installed Docker, use the following command to install the latest TensorFlow development Docker image. This contains the TensorFlow source:

```
docker pull tensorflow/tensorflow:devel
```

Visit [TensorFlow Docker images](#) for more information.

Now, run the following command to connect to your Docker instance and open a shell:

```
docker run -it -w /tensorflow_src -v $PWD:/mnt tensorflow/tensorflow:devel bash
```

You should now be on the command line of the TensorFlow Docker image, in the directory that contains the TensorFlow source code. You must issue the following commands to fetch the very latest code and install some required Python dependencies:

```
git fetch

git rebase origin master

pip install -U --user pip six numpy wheel setuptools mock tensorflow_estimator

pip install -U --user keras_applications==1.0.6 --no-deps

pip install -U --user keras_preprocessing==1.0.5 --no-deps
```

We must now configure the build. Running the following command from the root of the TensorFlow repo starts configuration. You are asked several questions. Just hit **return** at every prompt to accept the default option.

```
./configure
```

Once configuration is done, we are ready to go.

10.2 Train the model

The following command builds TensorFlow from source and starts training.



Note

The build takes several hours. To save time, you can download the [tiny_conv.pb](#) and skip to the following [Convert the model to the TensorFlow Lite format](#) section.

```
bazel run -c opt --copt=-mavx2 --copt=-mfma tensorflow/examples/speech_commands:train -  
- --model_architecture=tiny_conv --window_stride=20 --preprocess=micro --  
wanted_words="up,down" --silence_percentage=25 --unknown_percentage=25 --quantize=1
```

Notice how the `wanted_words` argument contains the words “up” and “down”. You can add any words that you like from the available ten to this field, separated by commas.

On older CPUs, you can leave out the `--copt` arguments. These arguments are there to accelerate training on chips that support the extensions.

The process takes a couple of hours. While you wait, you can look at a more detailed overview of the [speech model](#) that we are training.

10.3 Freeze the model

We must perform a few extra steps to be able to run the model directly on our microcontroller. With your trained model, you should run the following command to create a single “frozen graph” file that represents the trained model.



Note

We must provide our `wanted_words` argument again.

```
bazel run tensorflow/examples/speech_commands:freeze -- --  
model_qqqarchitecture=tiny_conv --window_stride=20 --preprocess=micro --  
wanted_words="up,down" --quantize=1 --output_file=/tmp/tiny_conv.pb --  
start_checkpoint=/tmp/speech_commands_train/tiny_conv.ckpt-18000
```

You now have a file, `/tmp/tiny_conv.pb`, that represents the model. This is great, but because we deploy the model on a tiny device, we must do everything that we can to make it as small and simple as possible.

10.4 Convert the model to the TensorFlow Lite format

To obtain a converted model that can run on the microcontroller, we must run a conversion script, **TensorFlow Lite converter**. This tool uses clever tricks to make our model as small and efficient as possible and to convert it to a TensorFlow Lite FlatBuffer. To reduce the size of the model, we used a technique called **quantization**. All weights and activations in the model get converted from 32-bit floating point format to an 8-bit and fixed-point format. This conversion does not only reduce the size of the network, but also avoids floating-point computations, that are more computationally expensive.

Run the following command to perform the conversion:

```
bazel run tensorflow/lite/toco:toco -- --input_file=/tmp/tiny_conv.pb --  
output_file=/tmp/tiny_conv.tflite --input_shapes=1,49,40,1 --input_arrays=Reshape_1 --  
output_arrays='labels_softmax' --inference_type=QUANTIZED_UINT8 --mean_values=0 --  
std_values=9.8077
```

You should now have a `/tmp/tiny_conv.tflite` file. We now must copy this file from our Docker instance to the host machine. To do this, run the following command:

```
cp /tmp/tiny_conv.tflite /mnt
```

This command places the file in the directory that you were in when you first ran the command to connect to Docker. For example, if you ran the command from `~/Desktop`, the file is at `~/Desktop/tiny_conv.tflite`.

To leave the Docker instance and get back to your regular command line, type the following:

```
exit
```

Appendix A Revisions

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

Table A-1 Issue 01

Change	Location	Affects
First release	All	All

Table A-2 Differences between issue 01 and issue 02

Change	Location	Affects
Adds updates to the TensorFlow Lite codebase	Download and build the sample application and Project structure	All

Table A-3 Differences between issue 02 and issue 03

Change	Location	Affects
Adds CMSIS-NN support	Download and build the sample application and Troubleshooting	All