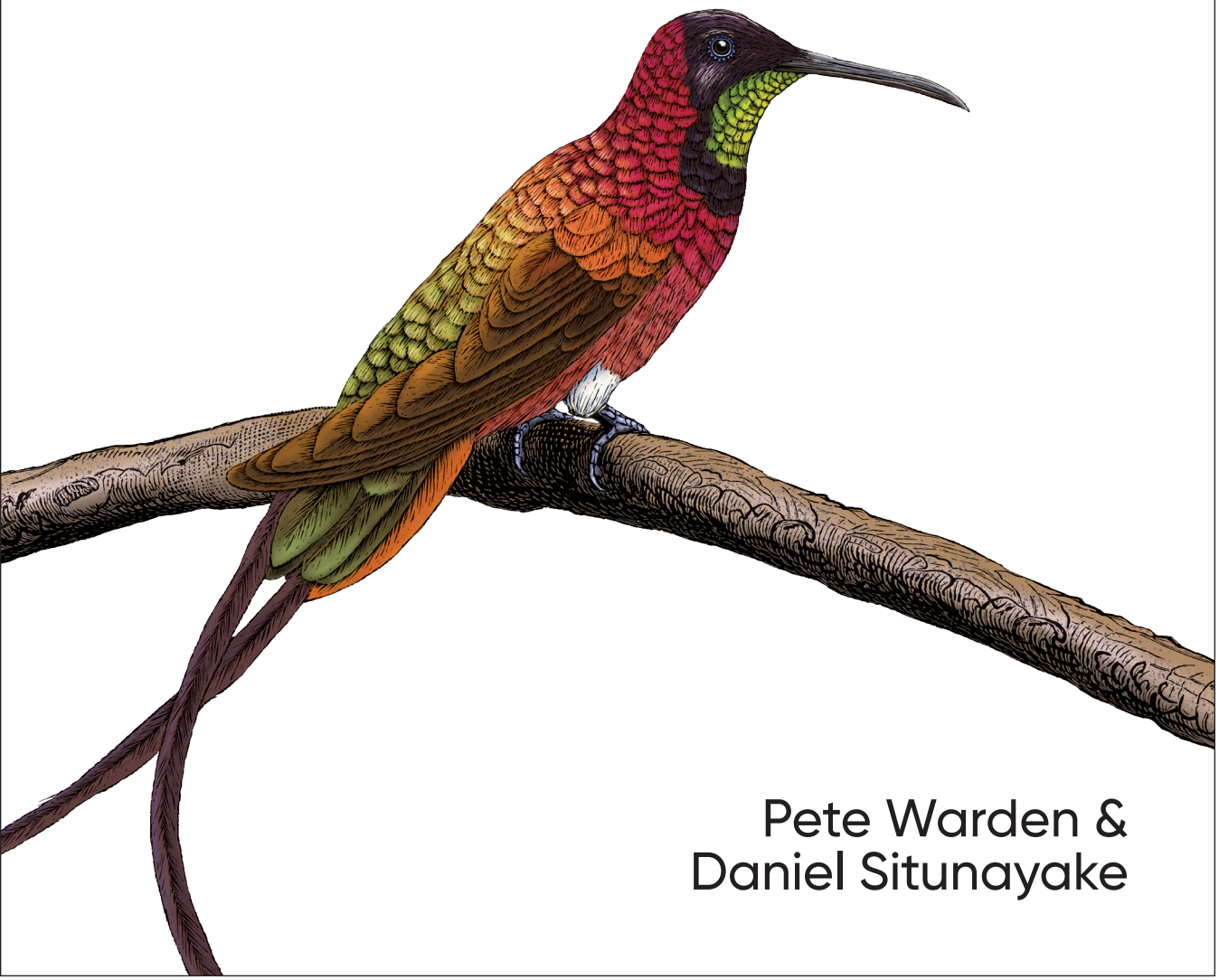
O'REILLY®

TinyML

Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers



Pete Warden &
Daniel Situnayake

O'REILLY®

TinyML

Deep learning networks are getting smaller. Much smaller. The Google Assistant team can detect words with a model just 14 kilobytes in size—small enough to run on a microcontroller. With this practical book you'll enter the field of TinyML, where deep learning and embedded systems combine to make astounding things possible with tiny devices.

Pete Warden and Daniel Situnayake explain how you can train models small enough to fit into any environment. Ideal for software and hardware developers who want to build embedded systems using machine learning, this guide walks you through creating a series of TinyML projects, step-by-step. No machine learning or microcontroller experience is necessary.

- Build a speech recognizer, a camera that detects people, and a magic wand that responds to gestures
- Work with Arduino and ultra-low-power microcontrollers
- Learn the essentials of ML and how to train your own models
- Train models to understand audio, image, and accelerometer data
- Explore TensorFlow Lite for Microcontrollers, Google's toolkit for TinyML
- Debug applications and provide safeguards for privacy and security
- Optimize latency, energy usage, and model and binary size

"This is a must-read book for anyone interested in machine learning on resource-constrained devices. It is a milestone in the development of AI."

—**Massimo Banzi**
Cofounder, Arduino

"This book teaches—through clear, interesting use cases—how to deploy ML on Arm-based microcontrollers."

—**Jem Davies**
VP, Fellow and GM, Machine Learning Group, Arm

Pete Warden is technical lead for mobile and embedded TensorFlow, and a founding member of the TensorFlow team. He was previously CTO and founder of Jetpac, acquired by Google in 2014.

Daniel Situnayake leads developer advocacy for TensorFlow Lite at Google, and helps run the tinyML meetup group. He cofounded Tiny Farms, the first US company using automation to produce insect protein at industrial scale.

MACHINE LEARNING / EMBEDDED DEVICES

US $49.99 CAN $65.99

ISBN: 978-1-492-05204-3

Twitter: @oreillymedia
facebook.com/oreilly

TinyML

Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers

Pete Warden and Daniel Situnayake

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

TinyML
by Pete Warden and Daniel Situnayake

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Mike Loukides
Development Editor: Nicole Taché
Production Editor: Beth Kelly
Copyeditor: Octal Publishing, Inc.
Proofreader: Rachel Head
Indexer: WordCo, Inc.
Interior Designer: David Futato
Illustrator: Rebecca Demarest

December 2019: First Edition

Revision History for the First Edition
2019-12-13: First Release
2020-08-07: Second Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781492052043* for release details.

978-1-492-05204-3

[LSI]

Table of Contents

Preface

Something about electronics has captured my imagination for as long as I can remember. We've learned to dig rocks from the earth, refine them in mysterious ways, and produce a dizzying array of tiny components that we combine—according to arcane laws—to imbue them with some essence of life.

To my eight-year-old mind, a battery, switch, and filament bulb were enchanting enough, let alone the processor inside my family's home computer. And as the years have passed, I've developed some understanding of the principles of electronics and software that make these inventions work. But what has always struck me is the way a system of simple elements can come together to create a subtle and complex thing, and deep learning really takes this to new heights.

One of this book's examples is a deep learning network that, in some sense, understands how to see. It's made up of thousands of virtual "neurons," each of which follows some simple rules and outputs a single number. Alone, each neuron isn't capable of much, but combined, and—through training—given a spark of human knowledge, they can make sense of our complex world.

There's some magic in this idea: simple algorithms running on tiny computers made from sand, metal, and plastic can embody a fragment of human understanding. This is the essence of TinyML, a term that Pete coined and will introduce in Chapter 1. In the pages of this book, you'll find the tools you'll need to build these things yourself.

Thank you for being our reader. This is a complicated subject, but we've tried hard to keep things simple and explain all the concepts that you'll need. We hope you enjoy what we've written, and we're excited to see what you create!

— Daniel Situnayake

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a tip or suggestion.

This element signifies a general note.

This element indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *https://tinymlbook.com/supplemental*.

If you have a technical question or a problem using the code examples, please send email to *bookquestions@oreilly.com*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not

need to contact us for permission unless you're reproducing a significant portion of code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of the example code from this book into your product's documentation does require permission.

We appreciate, but generally do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*TinyML* by Pete Warden and Daniel Situnayake (O'Reilly). Copyright Pete Warden and Daniel Situnayake, 978-1-492-05204-3."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

O'Reilly Online Learning

For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, please visit *http://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/tiny*.

Email *tinyml-book@googlegroups.com* to comment or ask technical questions about this book.

For news and more information about our books and courses, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

We'd like to give special thanks to Nicole Tache for her wonderful editing, Jennifer Wang for her inspirational magic wand example, and Neil Tan for the groundbreaking embedded ML work he did with the uTensor library. We couldn't have written this book without the professional support of Rajat Monga and Sarah Sirajuddin. We'd also like to thank our partners Joanne Ladolcetta and Lauren Ward for their patience.

This book is the result of work from hundreds of people from across the hardware, software, and research world, especially on the TensorFlow team. While we can only mention a few, and apologies to everyone we've missed, we'd like to acknowledge: Mehmet Ali Anil, Alasdair Allan, Raziel Alvarez, Paige Bailey, Massimo Banzi, Raj Batra, Mary Bennion, Jeff Bier, Lukas Biewald, Ian Bratt, Laurence Campbell, Andrew Cavanaugh, Lawrence Chan, Vikas Chandra, Marcus Chang, Tony Chiang, Aakanksha Chowdhery, Rod Crawford, Robert David, Tim Davis, Hongyang Deng, Wolff Dobson, Jared Duke, Jens Elofsson, Johan Euphrosine, Martino Facchin, Limor Fried, Nupur Garg, Nicholas Gillian, Evgeni Gousev, Alessandro Grande, Song Han, Justin Hong, Sara Hooker, Andrew Howard, Magnus Hyttsten, Advait Jain, Nat Jeffries, Michael Jones, Mat Kelcey, Kurt Keutzer, Fredrik Knutsson, Nick Kreeger, Nic Lane, Shuangfeng Li, Mike Liang, Yu-Cheng Ling, Renjie Liu, Mike Loukides, Owen Lyke, Cristian Maglie, Bill Mark, Matthew Mattina, Sandeep Mistry, Amit Mittra, Laurence Moroney, Boris Murmann, Ian Nappier, Meghna Natraj, Ben Nuttall, Dominic Pajak, Dave Patterson, Dario Pennisi, Jahnell Pereira, Raaj Prasad, Frederic Rechtenstein, Vikas Reddi, Rocky Rhodes, David Rim, Kazunori Sato, Nathan Seidle, Andrew Selle, Arpit Shah, Marcus Shawcroft, Zach Shelby, Suharsh Sivakumar, Ravishankar Sivalingam, Rex St. John, Dominic Symes, Olivier Temam, Phillip Torrone, Stephan Uphoff, Eben Upton, Lu Wang, Tiezhen Wang, Paul Whatmough, Tom White, Edd Wilder-James, and Wei Xiao.

CHAPTER 1
Introduction

The goal of this book is to show how any developer with basic experience using a command-line terminal and code editor can get started building their own projects running machine learning (ML) on embedded devices.

When I first joined Google in 2014, I discovered a lot of internal projects that I had no idea existed, but the most exciting was the work that the OK Google team were doing. They were running neural networks that were just 14 kilobytes (KB) in size! They needed to be so small because they were running on the digital signal processors (DSPs) present in most Android phones, continuously listening for the "OK Google" wake words, and these DSPs had only tens of kilobytes of RAM and flash memory. The team had to use the DSPs for this job because the main CPU was powered off to conserve battery, and these specialized chips use only a few milliwatts (mW) of power.

Coming from the image side of deep learning, I'd never seen networks so small, and the idea that you could use such low-power chips to run neural models stuck with me. As I worked on getting TensorFlow and later TensorFlow Lite running on Android and iOS devices, I remained fascinated by the possibilities of working with even simple chips. I learned that there were other pioneering projects in the audio world (like Pixel's Music IQ) for predictive maintenance (like PsiKick) and even in the vision world (Qualcomm's Glance camera module).

It became clear to me that there was a whole new class of products emerging, with the key characteristics that they used ML to make sense of noisy sensor data, could run using a battery or energy harvesting for years, and cost only a dollar or two. One term I heard repeatedly was "peel-and-stick sensors," for devices that required no battery changes and could be applied anywhere in an environment and forgotten. Making these products real required ways to turn raw sensor data into actionable information

locally, on the device itself, since the energy costs of transmitting streams anywhere have proved to be inherently too high to be practical.

This is where the idea of TinyML comes in. Long conversations with colleagues across industry and academia have led to the rough consensus that if you can run a neural network model at an energy cost of below 1 mW, it makes a lot of entirely new applications possible. This might seem like a somewhat arbitrary number, but if you translate it into concrete terms, it means a device running on a coin battery has a lifetime of a year. That results in a product that's small enough to fit into any environment and able to run for a useful amount of time without any human intervention.

I'm going to be jumping straight into using some technical terms to talk about what this book will be covering, but don't worry if some of them are unfamiliar to you; we define their meaning the first time we use them.

At this point, you might be wondering about platforms like the Raspberry Pi, or NVIDIA's Jetson boards. These are fantastic devices, and I use them myself frequently, but even the smallest Pi is similar to a mobile phone's main CPU and so draws hundreds of milliwatts. Keeping one running even for a few days requires a battery similar to a smartphone's, making it difficult to build truly untethered experiences. NVIDIA's Jetson is based on a powerful GPU, and we've seen it use up to 12 watts of power when running at full speed, so it's even more difficult to use without a large external power supply. This is usually not a problem in automotive or robotics applications, since the mechanical parts demand a large power source themselves, but it does make it tough to use these platforms for the kinds of products I'm most interested in, which need to operate without a wired power supply. Happily, when using them the lack of resource constraints means that frameworks like TensorFlow, TensorFlow Lite, and NVIDIA's TensorRT are available, since they're usually based on Linux-capable Arm Cortex-A CPUs, which have hundreds of megabytes of memory. This book will not be focused on describing how to run on those platforms for the reason just mentioned, but if you're interested, there are a lot of resources and documentation available; for example, see TensorFlow Lite's mobile documentation.

Another characteristic I care about is cost. The cheapest Raspberry Pi Zero is $5 for makers, but it is extremely difficult to buy that class of chip in large numbers at that price. Purchases of the Zero are usually restricted by quantity, and while the prices for industrial purchases aren't transparent, it's clear that $5 is definitely unusual. By contrast, the cheapest 32-bit microcontrollers cost much less than a dollar each. This low price has made it possible for manufacturers to replace traditional analog or electromechanical control circuits with software-defined alternatives for everything from toys to washing machines. I'm hoping we can use the ubiquity of microcontrollers in these devices to introduce artificial intelligence as a software update, without

requiring a lot of changes to existing designs. It should also make it possible to get large numbers of smart sensors deployed across environments like buildings or wildlife reserves without the costs outweighing the benefits or funds available.

Embedded Devices

The definition of TinyML as having an energy cost below 1 mW does mean that we need to look to the world of embedded devices for our hardware platforms. Until a few years ago, I wasn't familiar with them myself—they were shrouded in mystery for me. Traditionally they had been 8-bit devices and used obscure and proprietary toolchains, so it seemed very intimidating to get started with any of them. A big step forward came when Arduino introduced a user-friendly integrated development environment (IDE) along with standardized hardware. Since then, 32-bit CPUs have become the standard, largely thanks to Arm's Cortex-M series of chips. When I started to prototype some ML experiments a couple of years ago, I was pleasantly surprised by how relatively straightforward the development process had become.

Embedded devices still come with some tough resource constraints, though. They often have only a few hundred kilobytes of RAM, or sometimes much less than that, and have similar amounts of flash memory for persistent program and data storage. A clock speed of just tens of megahertz is not unusual. They will definitely not have full Linux (since that requires a memory controller and at least one megabyte of RAM), and if there is an operating system, it may well not provide all or any of the POSIX or standard C library functions you expect. Many embedded systems avoid using dynamic memory allocation functions like `new` or `malloc()` because they're designed to be reliable and long-running, and it's extremely difficult to ensure that if you have a heap that can be fragmented. You might also find it tricky to use a debugger or other familiar tools from desktop development, since the interfaces you'll be using to access the chip are very specialized.

There were some nice surprises as I learned embedded development, though. Having a system with no other processes to interrupt your program can make building a mental model of what's happening very simple, and the straightforward nature of a processor without branch prediction or instruction pipelining makes manual assembly optimization a lot easier than on more complex CPUs. I also find a simple joy in seeing LEDs light up on a miniature computer that I can balance on a fingertip, knowing that it's running millions of instructions a second to understand the world around it.

Changing Landscape

It's only recently that we've been able to run ML on microcontrollers at all, and the field is very young, which means hardware, software, and research are all changing extremely quickly. This book is a based on a snapshot of the world as it existed in 2019, which in this area means some parts were out of date before we'd even finished writing the last chapter. We've tried to make sure we're relying on hardware platforms that will be available over the long term, but it's likely that devices will continue to improve and evolve. The TensorFlow Lite software framework that we use has a stable API, and we'll continue to support the examples we give in the text over time, but we also provide web links to the very latest versions of all our sample code and documentation. You can expect to see reference applications covering more use cases than we have in this book being added to the TensorFlow repository, for example. We also aim to focus on skills like debugging, model creation, and developing an understanding of how deep learning works, which will remain useful even as the infrastructure you're using changes.

We want this book to give you the foundation you need to develop embedded ML products to solve problems you care about. Hopefully we'll be able to start you along the road of building some of the exciting new applications I'm certain will be emerging over the next few years in this domain.

Pete Warden

CHAPTER 2
Getting Started

In this chapter, we cover what you need to know to begin building and modifying machine learning applications on low-power devices. All the software is free, and the hardware development kits are available for less than $30, so the biggest challenge is likely to be the unfamiliarity of the development environment. To help with that, throughout the chapter we recommend a well-lit path of tools that we've found work well together.

Who Is This Book Aimed At?

To build a TinyML project, you will need to know a bit about both machine learning and embedded software development. Neither of these are common skills, and very few people are experts on both, so this book will start with the assumption that you have no background in either of these. The only requirements are that you have some familiarity running commands in the terminal (or Command Prompt on Windows), and are able to load a program source file into an editor, make alterations, and save it. Even if that sounds daunting, we walk you through everything we discuss step by step, like a good recipe, including screenshots (and screencasts online) in many cases, so we're hoping to make this as accessible as possible to a wide audience.

We'll show you some practical applications of machine learning on embedded devices, using projects like simple speech recognition, detecting gestures with a motion sensor, and detecting people with a camera sensor. We want to get you comfortable with building these programs yourself, and then extending them to solve problems you care about. For example, you might want to modify the speech recognition to detect barks instead of human speech, or spot dogs instead of people, and we give you ideas on how to tackle those modifications yourself. Our goal is to provide you with the tools you need to start building exciting applications you care about.

What Hardware Do You Need?

You'll need a laptop or desktop computer with a USB port. This will be your main programming environment, where you edit and compile the programs that you run on the embedded device. You'll connect this computer to the embedded device using the USB port and a specialized adapter that will depend on what development hardware you're using. The main computer can be running Windows, Linux, or macOS. For most of the examples we train our machine learning models in the cloud, using Google Colab, so don't worry about having a specially equipped computer.

You will also need an embedded development board to test your programs on. To do something interesting you'll need a microphone, accelerometers, or a camera attached, and you want something small enough to build into a realistic prototype project, along with a battery. This was tough to find when we started this book, so we worked together with the chip manufacturer Ambiq and maker retailer SparkFun to produce the $15 SparkFun Edge board. All of the book's examples will work with this device.

The second revision of the SparkFun Edge board, the SparkFun Edge 2, is due to be released after this book has been published. All of the projects in this book are guaranteed to work with the new board. However, the code and the instructions for deployment will vary slightly from what is printed here. Don't worry—each project chapter links to a *README.md* that contains up-to-date instructions for deploying each example to the SparkFun Edge 2.

We also offer instructions on how to run many of the projects using the Arduino and Mbed development environments. We recommend the Arduino Nano 33 BLE Sense board, and the STM32F746G Discovery kit development board for Mbed, though all of the projects should be adaptable to other devices if you can capture the sensor data in the formats needed. Table 2-1 shows which devices we've included in each project chapter.

Table 2-1. Devices written about for each project

Project name	Chapter	SparkFun Edge	Arduino Nano 33 BLE Sense	STM32F746G Discovery kit
Hello world	Chapter 5	Included	Included	Included
Wake-word detection	Chapter 7	Included	Included	Included
Person detection	Chapter 9	Included	Included	Not included
Magic wand	Chapter 11	Included	Included	Not included

What If the Board I Want to Use Isn't Listed Here?

The source code for the projects in this book is hosted on GitHub, and we continually update it to support additional devices. Each chapter links to a project *README.md* that lists all of the supported devices and has instructions on how to deploy to them, so you can check there to find out if the device you'd like to use is already supported.

If you have some embedded development experience, it's easy to port the examples to new devices even if they're not listed.

None of these projects require any additional electronic components, aside from person detection, which requires a camera module. If you're using the Arduino, you'll need the Arducam Mini 2MP Plus. And you'll need SparkFun's Himax HM01B0 breakout if you're using the SparkFun Edge.

What Software Do You Need?

All of the projects in this book are based around the TensorFlow Lite for Microcontrollers framework. This is a variant of the TensorFlow Lite framework designed to run on embedded devices with only a few tens of kilobytes of memory available. All of the projects are included as examples in the library, and it's open source, so you can find it on GitHub.

Since the code examples in this book are part of an active open source project, they are continually changing and evolving as we add optimizations, fix bugs, and support additional devices. It's likely you'll spot some differences between the code printed in the book and the most recent code in the TensorFlow repository. That said, although the code might drift a little over time, the basic principles you'll learn here will remain the same.

You'll need some kind of editor to examine and modify your code. If you're not sure which one you should use, Microsoft's free VS Code application is a great place to start. It works on macOS, Linux, and Windows, and has a lot of handy features like syntax highlighting and autocomplete. If you already have a favorite editor you can use that, instead; we won't be doing extensive modifications for any of our projects.

You'll also need somewhere to enter commands. On macOS and Linux this is known as the terminal, and you can find it in your Applications folder under that name. On Windows it's known as the Command Prompt, which you can find in your Start menu.

There will also be extra software that you'll need to communicate with your embedded development board, but this will depend on what device you have. If you're using either the SparkFun Edge board or an Mbed device, you'll need to have Python installed for some build scripts, and then you can use GNU Screen on Linux or macOS or Tera Term on Windows to access the debug logging console, showing text output from the embedded device. If you have an Arduino board, everything you need is installed as part of the IDE, so you just need to download the main software package.

What Do We Hope You'll Learn?

The goal of this book is to help more applications in this new space emerge. There is no one "killer app" for TinyML right now, and there might never be, but we know from experience that there are a lot of problems out there in the world that can be solved using the toolbox it offers. We want to familiarize you with the possible solutions. We want to take domain experts from agriculture, space exploration, medicine, consumer goods, and any other areas with addressable issues and give them an understanding of how to solve problems themselves, or at the very least communicate what problems are solvable with these techniques.

With that in mind, we're hoping that when you finish this book you'll have a good overview of what's currently possible using machine learning on embedded systems at the moment, as well as some idea of what's going to be feasible over the next few years. We want you to be able to build and modify some practical examples using time-series data like audio or input from accelerometers, and for low-power vision. We'd like you to have enough understanding of the entire system to be able to at least participate meaningfully in design discussions with specialists about new products and hopefully be able to prototype early versions yourself.

Since we want to see complete products emerge, we approach everything we're discussing from a whole-system perspective. Often hardware vendors will focus on the energy consumption of the particular component they're selling, but not consider how other necessary parts increase the power required. For example, if you have a microcontroller that consumes only 1 mW, but the only camera sensor it works with takes 10 mW to operate, any vision-based product you use it on won't be able to take advantage of the processor's low energy consumption. This means that we won't be doing many deep dives into the underlying workings of the different areas; instead, we focus on what you need to know to use and modify the components involved.

For example, we won't linger on the details of what is happening under the hood when you train a model in TensorFlow, such as how gradients and back-propagation work. Rather, we show you how to run training from scratch to create a model, what common errors you might encounter and how to handle them, and how to customize the process to build models to tackle your own problems with new datasets.