Brought to you by:

arm

Embedded Machine Learning Design

for dummies

A Wiley Brand

Learn why ML is moving to the edge

Configure your ML environment

Navigate the ML ecosystem

Arm Special Edition

About Arm

Arm technology is at the heart of a computing and connectivity revolution that is transforming the way people live and businesses operate. Arm's advanced, energy-efficient processor designs have enabled intelligent computing in more than 130 billion chips and are securely powering products from the sensor to the smartphone to the supercomputer. With more than 1,000 technology partners including the world's largest business and consumer brands, Arm is driving innovation into all areas where compute is happening—inside the chip, the network, and the cloud.

Arm's Project Trillium is driving the AI revolution, redefining device capabilities through a new class of ultra-efficient processors and highly optimized software. Providing a massive efficiency uplift from CPUs, GPUs, DSPs, and accelerators, Project Trillium completes the Arm Heterogenous ML compute platform with the Arm ML processor, and open-source Arm NN software.

To find out more about the fastest route to ML at the edge, please visit **www.arm.com/ai**.

Embedded Machine Learning Design

Arm Special Edition

for dummies®
A Wiley Brand

Embedded Machine Learning Design For Dummies®, Arm Special Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

For general information on our other products and services, or how to create a custom *For Dummies* book for your business or organization, please contact our Business Development Department in the U.S. at 877-409-4177, contact info@dummies.biz, or visit www.wiley.com/go/custompub. For information about licensing the *For Dummies* brand for products or services, contact BrandedRights&Licenses@Wiley.com.

ISBN 978-1-119-55123-2 (pbk); ISBN 978-1-119-55127-0 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

We're proud of this book and of the people who worked on it. Some of the people who helped bring this book to market include the following:

Contributing Writer: Ulrika Jägare

Project Editor: Martin V. Minner

Senior Acquisitions Editor: Katie Mohr

Editorial Manager: Rev Mengle

Business Development Representative: Karen Hattan

Production Editor: Tamilmani Varadharaj

Arm Contributors: Hellen Norman, Mark O'Connor, Tanuj Aurora

Introduction

From Alan Turing's 1950 prediction that "machines will eventually compete with men," through decades of research in labs and think tanks, machine learning (ML) has finally reached its viability point, exploding into the domain of engineering and into people's daily lives.

The technology is now moving quickly. ML is no longer the preserve of distant, cloud-based data centers. A dramatic shift in the capabilities of compute processing power and ML algorithms is driving applications, training, and inference back to the edge of the network — to the smart devices that are already an intrinsic part of everyday life.

About This Book

Embedded Machine Learning Design For Dummies, Arm Special Edition, shows you that adding machine learning to any device is not only possible but relatively easy to do. This book highlights key challenges and explains why it is vital to address them at the earliest stages of planning. The book addresses how to approach your platform configuration and explains why software matters. Finally, the book explores the importance of an ecosystem

perspective in ML development and gives examples of interesting ML solutions at the edge.

Icons Used in This Book

We occasionally use these special icons to focus your attention on important items:

This icon with the proverbial string around the finger reminds you about information that's worth recalling.

Expect to find something useful or helpful by way of suggestions, advice, or observations here.

This icon may be taken in one of two ways: Techies will zero in on the juicy and significant details that follow; others will happily skip ahead to the next paragraph.

IN THIS CHAPTER

- Grasping the basics of ML
- Understanding differentiation and cost reduction

Chapter 1
Realizing Why ML Is Moving to the Edge

Machine learning (ML) represents the greatest inflection point in computing for more than a generation — and it's already having a significant impact across virtually every market. It's leading to dramatic advances in connected car technologies, changing the face of healthcare, and influencing how city infrastructure is controlled. It's also affecting less obvious sectors such as farming, where device-born intelligence is enabling super-efficient watering practices, precisely targeted pest and disease control, and the optimization of crop harvesting.

The potential for ML is so far-reaching, it's hard to imagine a sector that won't be affected. For users, ML promises new levels of insight and convenience — at home, at work, and at leisure. For manufacturers, it offers the chance to make processes far more efficient and to create new business models and services.

Grasping the Basics of ML

The terms *artificial intelligence (AI)* and *ML* are often used interchangeably. However, in data science, the terms are distinct. This book uses the following definitions:

- *AI* is an umbrella term relating to hardware or software that enables a machine to mimic human intelligence. A range of techniques are used to deliver that "intelligence" including ML, computer vision, and natural language processing.
- *ML* is a subset of AI, as shown in Figure 1-1. ML uses statistical techniques to enable programs to "learn" through training, rather than being programmed with rules.

ML systems process training data to progressively improve performance on a task, providing results that improve with experience. Data is taken from the edge — be that an IoT device, edge server, or edge device — and sent to the cloud to be used for training.

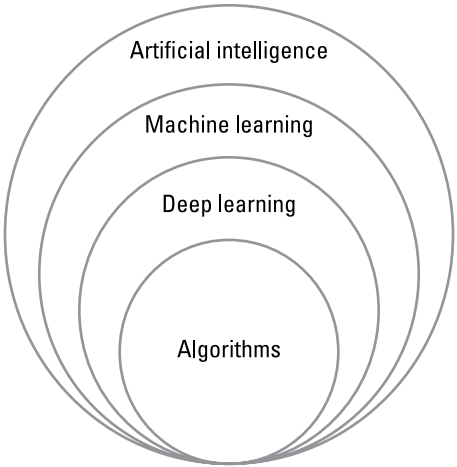


FIGURE 1-1: Machine learning is a subset of artificial intelligence.

Once an ML system is trained, it can analyze new data and categorize it in the context of the training data. This is known as *inference*.

ML is performed in one of two locations:

- **Cloud:** ML training is *typically* performed on remote, power-intensive, and compute-intensive server systems.
- **Edge:** ML inference is *usually* done locally, on the device that will deliver the outputs. The term *edge* may refer to an IoT device, edge server, or edge device.

Differentiation and Cost Reduction

While the first wave of ML focused on cloud computing, the combination of improved techniques for shrinking models to run on low-power hardware and increased compute capabilities on edge devices is opening possibilities for dramatic advantages in differentiation and unit cost reduction:

- **Reduced latency; increased reliability and safety:** Latency in the form of an unresponsive app or a page that won't load is an annoyance for the user, but many time-critical applications — such as automotive systems — simply *cannot* rely on connectivity to the cloud because a delay in response might have serious safety implications and seriously affect vehicle performance.
- **Power and cost:** Transmitting data from a device to a cloud server increases the power cost of performing ML because moving data around a system takes power. Cloud- or network-performed ML also adds a bandwidth tax, which can be significant because ML tends to be data-intensive. By performing as much ML as possible on-device, the cost and complexity burden on the network and cloud infrastructure is reduced.
- **Privacy and security:** Consumers and corporations are increasingly becoming aware of data security.

No one wants their privacy breached, but the risks are amplified when data is constantly shifted to the cloud and back. When processing is done on-device, legislative issues around the storing or transmission of data — and compliance with privacy regulations, such as the European Union's recent General Data Protection Regulation (GDPR) — are minimized.

- **Personalization:** In addition to privacy and security, performing ML on-device can lead to a more personalized compute experience. As more devices become "intelligent," they will need to adapt and provide a contextualized response to their immediate environment — instantaneously. When these devices connect users to the things they care about, AI becomes accessible and personal. Maintaining unique, customized models for every user in the cloud is a significant ongoing expense, so edge devices that can run their *own* customized models will provide a competitive advantage.

Ultimately, ML at the edge delivers a more reliable, responsive, and secure user experience that reduces per-unit cost, personal data risk, and power requirements — and isn't dependent on network connections.

Exploring ML Opportunities

ML is not about a new type of device; it's about *every* device. ML enables devices to contextualize their

immediate environments far better — using data such as vision, sound, heat, and vibration. This innovation is driving new business models, reducing costs, and optimizing performance across a range of parameters.

Although the benefits of ML are exciting, taking the first steps to add ML capability to your product may seem daunting. ML processing requirements vary significantly according to the model and workload; no "one-size-fits-all" solution exists. Almost all models allow accuracy and performance to be traded freely. This flexibility allows a perfect fit between device hardware and the model capability, but it raises additional questions:

- What are the use cases?
- Which neural network (NN) model provides the best performance/accuracy trade-off?
- Which hardware should you choose to complement it? Can lower-capability hardware be used with a reduced-accuracy model? Can the model be tuned to make use of all the available RAM?
- Which tools are available to help a team answer these questions?
- Most importantly: How can these issues be balanced to deliver the best performance at the best unit cost?

IN THIS CHAPTER

- Understanding the components of an ML platform
- Making choices for your ML environment
- Learning from a case study

Chapter 2
Configuring Your ML Environment

Selecting the right solution for your application entails a series of trade-offs: from small, low-power microcontroller units (MCUs) for cost- and power-constrained systems to central processors (CPUs) for greater performance and general-purpose programmability; graphics processors (GPUs) for faster performance with graphics-intensive applications; and neural processors (NPUs) for the most intensive and efficient ML processing. This chapter leads you through the process.

Understanding the Components of an ML Platform

What does your platform need to run ML workloads? Perhaps surprisingly, in hardware terms it may need nothing more than you already have. Many platforms are already efficiently running ML applications on CPUs and GPUs alone. (You can see some examples in Chapter 5.)

For some use cases, higher performance requirements may demand a dedicated ML processor, such as a neural processing unit (NPU). However, adding an NPU to an already crowded mix of CPUs and GPUs running ML applications can create complexity if the software stack for each processor type is different. This issue creates a need for a software layer that hides hardware complexity from the software applications.

Heterogeneous computing refers to systems that use more than one kind of processor or core to achieve better performance and greater efficiency.

A heterogeneous compute platform allows application developers to write ML applications using their favorite NN frameworks — such as Google's TensorFlow or Facebook's Caffe and Caffe2 — and target a variety of processor types from multiple vendors. As shown in Figure 2-1,

the software layer between the application and the hardware handles the translation of the workload to target the applicable and available core types automatically.

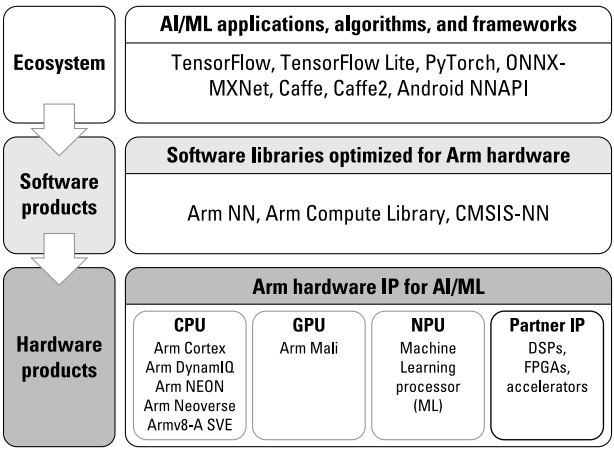


FIGURE 2-1: An example of a heterogeneous ML compute platform with a variety of core types and open-source software.

Open standards can help to reduce complexity and assure future compatibility with solutions from a variety of vendors. One example of this is the Open Neural Network Exchange Format (ONNX), supported by Arm, Microsoft, Facebook, Amazon, and others to provide a common format between training frameworks and runtime engines.

You have many options for running an ML model:

- **Low-power/always-on microcontroller CPUs:** Small models can be run on these cores with a very low power budget. Example: detecting activity or behavior that wakes up the rest of the system to perform more detailed analysis.

  
  TIP

  Sometimes microcontroller CPUs have DSP extensions, known as Digital Signal Controllers, which allow for faster processing, thus providing clean signals for ML without the need for a separate DSP. This feature saves cost, area, and development time, especially for those without a DSP software development background.

- **High-efficiency, general-purpose CPUs:** Depending on the configuration and number of cores, these can comfortably run entire workloads. Example: real-time speech recognition for closed-caption generation.

- **Multimedia and GPU cores:** These often run the same models as a mobile-class core with increased performance or efficiency. If a design already requires a GPU, offloading the ML workload may allow you to reuse this silicon for maximum cost-benefit.

- **Dedicated NPUs:** Dedicated silicon offers the highest efficiency for running general-purpose ML models in a low-power environment. Example: determining which pixels of a real-time HD video feed correspond to a person.

Knowing exactly which components are required in your device demands careful consideration of multiple factors, which the next section discusses.

Making Choices for Your ML Environment

The right hardware for your application, what you need from a dedicated NPU, and how you integrate it into systems and tools varies from case to case. However, the following capabilities are important to consider as part of your architectural design for a constrained environment:

- **Processing:** What type of data ingestion and processing does your edge solution require? This may depend on the type of models you run and the number of models running at the same time. How complex are these models? Are they compute- or bandwidth-constrained?

- **Connectivity:** What can be done locally on the device? What requires a connection to the cloud? What must stay in the cloud?
- **Integration:** Are there any integrations or dependencies that must be managed? What NN frameworks will the developer community be using? How will the models developed in a variety of different frameworks be supported on the system?
- **Power, size, and heat:** Does your edge device have power, size, and thermal constraints?
- **Accuracy:** What is the desired accuracy? Although you might initially think that the higher the accuracy, the better, that isn't always true. After a certain level, you hit a point of diminishing returns. In certain scenarios, a 2 percent increase in accuracy may require a 10x increase in compute and memory requirements, for example. You should clearly understand what level of accuracy is required for any particular use case.
- **Privacy:** What are the privacy and security concerns around your edge solution?
- **Workload:** Will an NPU be mostly idle and only used for work that an otherwise idle CPU or GPU can accomplish? On the other hand, if a CPU or

> GPU *is* capable of running a model, might an NPU be a better solution if those resources are already heavily loaded?

Taking these points into consideration, you can look at the different components that are required. Is a CPU or GPU sufficient? Do you need to add an NPU? How big must that NPU be to service your requirements within the cost constraints? Look at the different core types available in the market and match them to your area and power budgets, as well as your compute requirements.

The following case study offers an example of the complexity that can be managed on an embedded device at the edge.

Case Study: An Edge ML Solution for Asthma Patients

The Amiko Respiro provides smart inhaler technology that helps asthma sufferers breathe more easily. The solution includes ML-powered sensors as add-ons to standard inhalers to improve asthma treatment. These smart sensors must be low-powered, scalable, and cost-efficient enough to work with a patient-facing app on a connected platform.

At the center of Respiro's sensor module is an ultra-low-power Arm Cortex-M processor, enabling:

- Processing that takes place securely on the device with no need to connect to the cloud
- Bluetooth low-power (BTLE) connectivity to a smartphone app
- An efficiency profile that extends device battery life

The solution uses ML to interpret vibration data from the inhaler. The sensor is trained to recognize the patient's breathing pattern and inhalation time and can calculate important parameters such as lung capacity and inhalation technique.

The processor allows the Respiro to run real-time ML algorithms that recognize behavior patterns and interpret data within the sensor module itself. The user doesn't need to wait for back-end infrastructure to process detailed sensor data. When the user presses the trigger, the module instantly recognizes the breath data pattern and provides low-latency, private user feedback.

The Respiro sensor is bundled with an app that the patient installs on a smartphone. The sensor collects inhaler use data without disrupting the medication delivery pathway and sends data and feedback to the app. The sensor also has the flexibility to add new features and easily scale up to deliver further innovative connected healthcare solutions going forward.

IN THIS CHAPTER

- Reducing time to market with off-the-shelf hardware, tools, and simulators
- Differentiating through software
- Training and deploying ML models

Chapter 3
Why Software Really Matters

According to the famous quote by the prominent technologist Mark Andressen, "software is eating the world." It's certainly capable of eating development budgets and delaying product schedules if you get it wrong. However, because software innovation and ML functionality are increasingly becoming key differentiators for many device-makers, if you're building ML into your product, you need a strong grasp on your specific software needs.

Reducing Time to Market with Off-The-Shelf Hardware, Tools, and Simulators

The quickest way to de-risk software and ML model development is to do your prototyping with off-the-shelf hardware that has functionality as close to your final product as possible. That can be through the use of industry-standard CPUs and GPUs or a previous iteration of the product itself.

To achieve what you need, it's advisable to choose components sourced from a large-scale provider — preferably from an ecosystem with a wide range of possible prototype platforms. This means you'll be able to select technology similar to your final design in the areas that matter — such as performance and power or with specific hardware connectivity.

Once a prototype platform is in the hands of the software team, the next step is to create a useful development environment. Depending on your device, this might include:

- An operating system
- Compilers
- Performance libraries

- Debugging and profiling tools
- ML frameworks

Development teams take time to build expertise in these tools and libraries. Building on a platform that's already familiar to your team increases productivity and the capacity for innovation, greatly reducing the risk to your software development.

TIP

Ideally, the tools and the software your team develops should be portable to your final platform. Common standards and architectures pave the way for a smooth and painless integration when early hardware becomes ready for testing, minimizing the chance of last-minute delays and performance surprises.

If your chosen architecture provides simulators and fast modeling tools, this risk is minimized even further. Today's fast models and simulators are capable of bringing up an entire operating system in simulation and running the final application code even before silicon is ready, with cycle simulators providing further correctness and performance predictions. Projects that use simulation effectively can often successfully deploy the same software directly onto the first silicon.

In short, to effectively de-risk software development, select an architecture and off-the-shelf prototyping platform that

- Closely reflects the functionality required in the final product
- Supports a common set of tools, libraries, and frameworks that is familiar to your team
- Provides robust simulation solutions that minimize time spent integrating software and hardware

Differentiating Through Software

A decision for hardware architecture is also a decision for its software stack and the ecosystem supporting it. Large ecosystems frequently have dedicated teams continuously optimizing and improving their software stacks. For example, Arm provides Arm NN, Compute Library, and CMSIS-NN. Over a four-month period, the performance of NNs such as AlexNet, Inception, SqueezeNet, and VGG-16 improved 1.6x to 2.6x on Cortex-A series CPUs and the Mali GPU, as shown in Figure 3-1.

Developing on top of an actively developed and maintained stack ensures you benefit from future performance and security improvements without taking time away from your own development efforts.

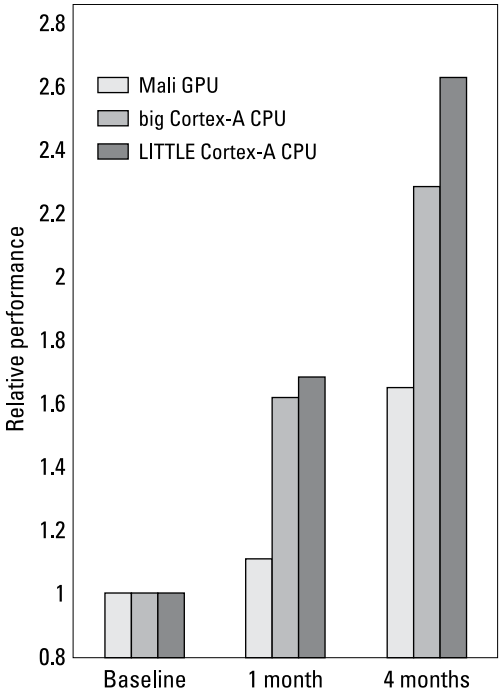


FIGURE 3-1: Software performance gain.

Many designs have the option of trading off software solutions for hardware implementations. In general, software solutions allow your team to continue to innovate on software features as well as ML model architectures and performance training right up until release.

In some cases, the performance gains unlocked upstream in the ecosystem and the ML research community may enable the second version of your product to provide significantly differentiated performance and features, with minimal hardware changes — as long as those features were not hard-wired into the original design.

When choosing a hardware solution, pay close attention to these areas of software performance:

- **An optimized software stack:** This is key to unlocking promised hardware performance figures.
- **Ecosystems:** These must be able to deliver consistent performance and security updates.
- **Your solution**: Consider which parts require a hardware implementation and which could be performed more flexibly in software.

Training and Deploying ML Models

The upside, and downside, of NNs is their flexibility. Accuracy, performance, memory requirements, and hardware compatibility can be traded off against each other to fit your desired performance parameters — but evaluating this trade-off, and training the networks for

optimal deployment, is often not trivial. As with software development, your team will take time to build expertise in these tools. To minimize your risk, look for solutions built upon, and compatible with, open-source standards such as TensorFlow and ONNX.

As model training and optimization techniques improve, intelligence that was once confined to the datacenter — such as speech and image recognition, recommendation, and prediction — is increasingly available on mobile- and embedded-class CPUs and GPUs, in addition to the increasing range of high-performance NPUs.

The world's most popular AI platform today is the smartphone, which includes ML features like predictive text, speech recognition, and computational photography. However, as the benefits of embedded ML development are becoming better understood, these capabilities are expanding and becoming available in an increasing range of edge devices. For example, effective voice keyword search — such as listening for a wake word or simple command — has been effectively demonstrated on microcontrollers.

REMEMBER

ML requires a *training* phase, where the learning happens, and an *inference* phase where that learned is applied. Today, training typically happens on servers or similar devices, but inference is increasingly moving to the edge — to consumer and industrial devices in homes, factories, and places of work.

ML workloads are characterized by demanding requirements for computation and memory bandwidth. However, recent optimization techniques such as quantization, pruning, and model compression make it possible to deploy solutions previously confined to the datacenter onto mobile and embedded devices.

REMEMBER

Making the best use of available compute resources is increasingly complex. Although the largest ML models may run most efficiently on a dedicated ML processor, in many cases it may be more energy-, latency- and cost-efficient to run some networks — or parts of networks — on one or more CPU or GPU cores, depending on the device, its current wake-state, and its workload. The flexibility to easily move models between compute resources is critical.

TIP

Look for a platform that enables developers and data scientists to easily build and run ML applications in a power-efficient environment across CPUs, GPUs, and NPUs, leaving you maximum flexibility in your current and future designs. The software should provide a bridge to existing ML frameworks, such as TensorFlow and PyTorch, while hiding implementation details of the underlying hardware. This allows developers to continue to use their preferred frameworks and tools while the heterogeneous ML platform seamlessly converts the results to run on the underlying hardware.

IN THIS CHAPTER

- Reusing existing assets
- Finding a better product/market fit

Chapter 4
Why Ecosystems Are Important

There's an old saying: "When you marry someone, you marry their family too." When you select an architecture for your device, you're also committing — for better or worse — to an ecosystem of silicon partners, original equipment manufacturers (OEMs), software vendors, and consultants; its array of training and educational materials; and — not least — its recruitment pool of experienced engineers.

Getting the most out of an ecosystem is an important part of minimizing costs and reducing time to market, as this chapter shows.

Reusing Existing Assets

One of the most important assets for the success of your business is the ability to build and capitalize upon your in-house expertise. An ecosystem built around interoperable standards allows solutions and skills to migrate across projects, reducing your risk and time to market.

A standards-based design also futureproofs your software and ML investment: Being able to switch hardware with minimal disruption to your software stack gives you the agility and freedom to move to a better solution.

To this end, single-purpose hardware can be a double-edged sword: Although software development may be initially fast, if the platform doesn't provide for a pipeline of new products with various performance and power profiles, you may have to turn to other non-aligned technology providers for future designs. This technology "divorce" means that expertise gathered on one design does not transfer to the next product — or even an iteration of the current product. Ultimately, initial speed may give way to eventual stagnation.

ML brings its own set of challenges: The research community is moving quickly, and new and better operators, activation functions, and architectures for NNs are being published every day. Whether you choose to run the models on a CPU, a GPU, or an NPU, you can avoid being blindsided by future research or competitor developments by making sure that new and custom ML operators

can be supported in software rather than being limited by the hardware design.

Finding a Better Product/ Market Fit

Very few companies custom design every aspect of a device, so even the largest need the ability to reuse their own components or source compatible external ones.

When you select a large and vibrant ecosystem, you gain access to a wider array of potential components and suppliers, which gives you the best chance to find the perfect product/market fit.

In addition, a large ecosystem can make it easier to find off-the-shelf solutions that meet most — if not all — of your product needs. This helps to minimize the number of components per device, reducing per-unit costs.

On the Shoulders of Giants

Speed to market and differentiation are key elements in the design of any product. Engaging with a standards-based ecosystem is helpful because it means a company can rely on a community of partners rather than inventing every element from scratch.

Nowhere is this more important than in ML. As the speed of new business opportunities and creative real-world solutions increases, new models and algorithms must be adapted more quickly. Open-source blueprints play an important role in allowing companies to extend their proprietary frameworks and tools to bring innovative products to market faster.

ML standardization is an important but challenging task because everything — from use cases to software stacks and hardware processors — is changing so quickly. Even the data poses a problem. However, from the perspective of system architecture, some vendors offer standardization across the ecosystem.

ML requires two kinds of frameworks to work seamlessly together:

- The training framework, such as TensorFlow, is used by data scientists and ML engineers to train new models from data.
- The inference framework, such as Arm NN, loads those models and executes them efficiently on the underlying CPUs, GPUs, and ML-specific IP.

When Arm donated its open-source inference engine, Arm NN, to the Linaro Machine Intelligence initiative, the gift was a rallying point for the entire ecosystem to contribute and develop a single, optimized inference engine. This approach lets everyone in the ecosystem benefit from each other's expertise and optimizations.

IN THIS CHAPTER

- Applying ML solutions at the edge
- Solving problems at the edge

Chapter 5
Ten Examples of ML at the Edge

The potential for applying ML at the edge is boundless. Here are ten inspiring solutions:

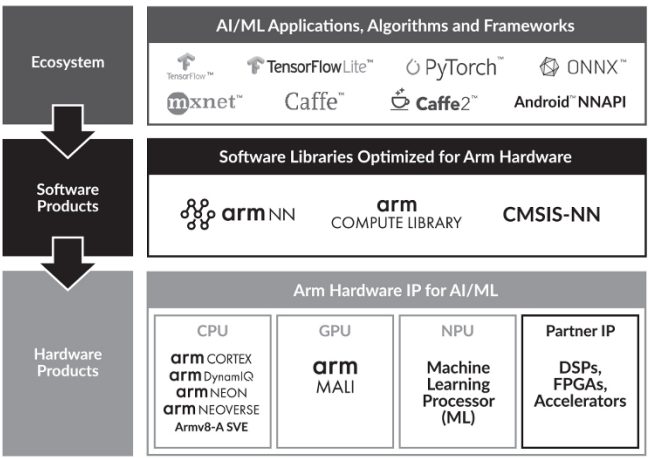
- **Secure, low-power smart home security:** Uses on-device, always-on, motion, person, and sound detection to identify family members or intruders. Starts recording only when it detects motion or sound, sending a notification to the user's smartphone.
- **Hospital staff/visitor tracking:** Alerts receptionists to unknown people or unauthorized access.

Edge recognition means images of visitors and patients are never stored or transmitted.

- **Plant disease detection:** Uses an image recognition smartphone app to detect disease with near 100 percent accuracy — even off-network.
- **Faster produce selection:** Classifies fruit and vegetables by camera, automatically identifying different categories and improving production-line efficiency.
- **Drone avionics:** Recognizes and follows a target while avoiding obstacles, via camera-based vision and movement prediction.
- **No-latency driver assistance:** Helps to reduce collisions using cameras, motion sensors, and GPS to understand and guide driver behavior in real time.
- **Improved human-machine interaction:** Streamlines interactions, boosts productivity, and creates a smoother user experience across devices.
- **On-device translation:** Makes communication possible — however remote the location — and avoids costly roaming charges.
- **Device optimization:** Significantly extends battery life by optimizing operating system scheduling for individual applications.
- **Computational photography:** Tackles tricky problems such as distant objects and low-light conditions to achieve near-perfect images.

arm

Machine Learning Everywhere Compute Happens



Flexible, scalable, power-efficient machine learning solutions for every application.

To find out more visit:

www.arm.com/machine-learning

Intelligence moves to the edge with machine learning

The growth of machine learning represents the biggest inflection point in computing for more than a generation. Increased use of ML will have a massive effect on just about every segment of society, and ML is now running at the edge wherever possible. This book shows you that adding ML to any edge device is not only possible, but relatively easy to do.

Inside...

- Understand ML basics
- Explore edge opportunities
- Configure an ML environment
- Speed development with optimized software libraries
- Achieve consistency across your platform

arm

Go to Dummies.com®
for videos, step-by-step photos, how-to articles, or to shop!

ISBN: 978-1-119-55123-2
Not For Resale

for dummies®
A Wiley Brand

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.