Embedded AI: Principles, Algorithms, and Applications

Table of contents

10 Efficient AI   51

11 Optimizations   52

12 Deployment   53

13 On-Device Learning   54

14 Hardware Acceleration   55

15 MLOps   56

16 AI Sustainability   57

17 Responsible AI   58

18 Generative AI   59

References   60

Preface

In "Embedded AI: Principles, Algorithms, and Applications", we will embark on a critical exploration of the rapidly evolving field of artificial intelligence in the context of embedded systems, originally nurtured from the foundational course, tinyML from CS249r.

The goal of this book is to bring about a collaborative endeavor with insights and contributions from students, practitioners and the wider community, blossoming into a comprehensive guide that delves into the principles governing embedded AI and its myriad applications.

As a living document, this open-source textbook aims to bridge gaps and foster innovation by being globally accessible and continually updated, addressing the pressing need for a centralized resource in this dynamic field. With a rich tapestry of knowledge woven from various expert perspectives, readers can anticipate a guided journey that unveils the intricate dance between cutting-edge algorithms and the principles that ground them, paving the way for the next wave of technological transformation.

The Philosophy Behind the Book

We live in a world where technology perpetually reshapes itself, fostering an ecosystem of open collaboration and knowledge sharing stands as the cornerstone of innovation. This philosophy fuels the creation of "Embedded AI: Principles, Algorithms, and Applications." This is a venture that transcends conventional textbook paradigms to foster a living repository of knowledge. Anchoring its content on principles, algorithms, and applications, the book aims to cultivate a deep-rooted understanding that empowers individuals to navigate the fluid landscape of embedded AI with agility and foresight. By embracing an open approach, we not only democratize learning but also pave avenues for fresh perspectives and iterative enhancements, thus fostering a community where knowledge is not confined but is nurtured to grow, adapt, and illuminate the path of progress in embedded AI technologies globally.

Prerequisites

Venturing into "Embedded AI: Principles, Algorithms, and Applications" does not mandate you to be a maestro in machine learning from the outset. At its core, this resource seeks to nurture learners who bear a fundamental understanding of systems and harbor a curiosity to explore the confluence of disparate, yet interconnected domains: embedded hardware, artificial intelligence, and software. This confluence forms a vibrant nexus where innovations and new knowledge streams emerge, making a basic grounding in system operations a pivotal tool in navigating this dynamic space.

Moreover, the goal of this book is to delve into the synergies created at the intersection of these fields, fostering a learning environment where the boundaries of traditional disciplines blur to give way to a holistic, integrative approach to modern technological innovations. Your interest in unraveling embedded AI technologies and low-level software mechanics would be guiding you through a rich learning experience.

Conventions Used in This Book

Please follow the conventions listed in Conventions

How to Contact Us

Please contact *vj@eecs.harvard.edu*

How to Contribute

Please see instructions at here.

Contributors

Please see Credits.

Acknowledgements

coming soon.

About

Overview

The target audience for this book is primarily **embedded systems engineers** who want to understand and implement machine learning algorithms on resource-constrained devices. However, the book also caters to other professionals with an interest in the field of tiny machine learning (**TinyML**).

Topics Covered

The book covers a wide range of topics related to embedded machine learning, providing a comprehensive understanding of the field. The topics covered include:

1. Overview and Introduction to Embedded Machine Learning
2. Data Engineering
3. Embedded Machine Learning Frameworks
4. Efficient Model Representation and Compression
5. Performance Metrics and Benchmarking of ML Systems
6. Learning on the Edge
7. Hardware Acceleration for Edge ML: GPUs, TPUs, and FPGAs
8. Embedded MLOps
9. Secure and Privacy-Preserving On-Device ML
10. Responsible AI
11. Sustainability at the Edge
12. Generative AI at the Edge

By the end of this book, you will gain a brief introduction to machine learning and IoT. You will learn about real-world deployments of embedded machine learning systems. We hope you will also gain practical experience through hands-on project assignments.

Intended Audience

- **Embedded Systems Engineers:** This book is a valuable resource for engineers working in the field of embedded systems. It provides a solid foundation in TinyML, allowing them to design and implement intelligent applications on microcontrollers and other embedded platforms with limited resources.

- **Computer Science and Electrical Engineering Students:** Students pursuing degrees in computer science and electrical engineering can benefit from this book. It offers an introduction to the concepts, algorithms, and techniques used in TinyML, preparing students to tackle real-world challenges in the emerging field of embedded machine learning.

- **Researchers and Academics:** Researchers and academics in the field of machine learning, computer vision, and signal processing will find this book useful. It offers insights into the unique challenges of deploying machine learning algorithms on low-power, low-memory devices, enabling them to develop new approaches and advance the field of TinyML.

- **Industry Professionals:** Professionals working in industries like IoT, robotics, wearable technology, and smart devices will find this book relevant. It equips them with the knowledge required to integrate machine learning capabilities into their products, enabling intelligent and autonomous behavior.

Key Features

- **Introduction to Machine Learning:** A fundamental understanding of machine learning concepts, including supervised, unsupervised, and reinforcement learning.

- **TinyML Fundamentals:** Exploring the challenges and constraints associated with deploying machine learning on small, low-power devices.

- **Hardware Platforms:** Coverage of popular microcontrollers and development boards specifically designed for TinyML applications, along with their architecture and specifications.

- **Training Models:** Techniques and tools for training machine learning models suitable for embedded systems, including considerations for model size, accuracy, and resource utilization.

- **Optimization Techniques:** Strategies for model compression, quantization, and algorithmic optimization to ensure efficient execution on resource-constrained devices.

- **Real-world Applications:** Practical use cases and examples demonstrating the deployment of TinyML in various domains, such as industrial automation, healthcare, and environmental monitoring.

- **Challenges and Future Trends:** Discussion on the current challenges in TinyML, potential solutions, and emerging trends in the field.

By encompassing these aspects, our aim is to make this book a go-to resource for anyone interested in developing intelligent applications on embedded systems.

Prerequisites

- **Basic Programming Knowledge**: It is *recommended* that readers have some prior experience with programming, preferably in Python. Understanding variables, data types, control structures, and basic programming concepts will facilitate comprehension and engagement with the book.

- **Familiarity with Machine Learning Concepts**: While not *essential*, a basic understanding of machine learning concepts, such as supervised and unsupervised learning, will help readers grasp the material more easily. However, the book provides sufficient explanations to bring readers up to speed if they are new to the field.

- **Python Programming Skills (Optional)**: Readers with some Python programming experience will have an advantage when engaging with the coding portions of the book. Familiarity with libraries such as *NumPy*, *scikit-learn*, and *TensorFlow* will greatly facilitate the implementation and experimentation with machine learning models.

- **Learning Mindset**: The book has been structured to be accessible to a wide audience, including readers with varying levels of technical expertise. It provides a gradual learning curve, allowing readers to start with general knowledge about the field, progress to coding exercises, and potentially advance to deploying models on embedded devices. However, to fully benefit from the book, readers should be willing to *challenge themselves* and engage in practical exercises and projects.

- **Availability of Resources**: To fully explore the practical aspects of TinyML, readers should have access to the necessary resources. These include a computer with Python and relevant libraries installed, as well as *optional access to an embedded development board or microcontroller* for experimenting with deploying machine learning models.

By ensuring that these general requirements are met, readers will have the opportunity to broaden their understanding of TinyML, gain hands-on experience with coding exercises, and even venture into practical implementation on embedded devices, enabling them to *push the boundaries* of their knowledge and skills.

This book is a collaborative effort started by the CS249r Tiny Machine Learning class at Harvard University. We intend for this book to become a community-driven effort to help educators and learners get started with TinyML. This living document will be continually updated as we continue to learn more about TinyML and how to teach it.

Intended Audience

This book is designed specifically for newcomers who wish to explore the fascinating and nascent world of tiny machine learning (tinyML). It provides the basic underpinnings of ML and embedded systems, and moves into more complex and broader topics relevant to both the tinyML and broader research community.

Book Structure

This book is specifically designed to serve both educators and learners in getting started with TinyML. The topics begin with a basic introduction to machine learning (ML) and embedded systems. Following this, readers will be introduced to the ML workflow in the context of tinyML, including data collection, data engineering, model development, model deployment, and then MLOps. Subsequently, special topics are covered such as on-device learning, secure and privacy-preserving ML, responsible AI, sustainability, and generative AI.

Key Takeaways

Users of this book will learn how to train and deploy deep neural network models on resource-constrained microcontrollers and the broader challenges associated with their design, development, deployment, and use.

After completing the course, readers will be empowered with the capabilities to design and implement their own ML-enabled projects, starting from defining a problem to gathering data and training the neural network model and finally deploying it to the device to display inference results or control other hardware appliances based on inference data.

1 Introduction

Welcome to our comprehensive guide to Tiny Machine Learning (TinyML), where we endeavor to bring a fresh perspective to the rapidly emerging field that straddles the domains of electrical engineering, computer science, and applied data science. This book aims to close the gap between complex machine learning abstractions and real-world applications on small devices, providing both theory enthusiasts and practitioners an end-to-end understanding of TinyML.

We begin with an overall introduction to the field of embedd systems and machine learning. We start by elaborating on the key principles of embedded systems, setting the groundwork for embedded machine learning. Then we pivot our attention to deep learning, focusing specifically on deep learning methods given their representation capacity and overall performance in a variety of tasks, especially when applied to small devices.

The book goes on to discuss step-by-step workflows in machine learning, data engineering, pre-processing, and advanced model training techniques. It provides comprehensive analyses of several in-use machine learning frameworks, and how they can be employed effectively to develop efficient AI models.

In a world where efficiency is key, we also discuss TinyML model optimization and deployment strategies. Special focus is given to on-device learning. How do we train a machine learning model on a tiny device while achieving admirable efficiency? What are the current hardware acceleration techniques? And how can we manage the lifecycle of these models? The reader can expect exhaustive answers to these and many more questions in our dedicated chapters.

Importantly, we adopt a forward-looking stance, discussing the sustainability and ecological footprint of AI. We explore the location of TinyML within such debates, and how TinyML may contribute to more sustainable and responsible practices.

Finally, the book ends with a speculative leap into the world of generative AI, outlining its potentials in the TinyML context.

Whether you are an absolute beginner, a professional in the field, or an academic pursuing rigorous research, this book aims to offer a seamless blend of essential theory and practical insight, triggering stimulating conversations around TinyML. Let's embark on this thrilling journey to explore the incredible world of TinyML!

2 Embedded Systems

In the realm of tinyML, the role of embedded systems is akin to that of the foundation stone in a building, offering a sturdy base where intelligent algorithms can operate efficiently and effectively. Embedded systems, characterized by their dedicated functions and real-time computational abilities, become the nexus where data meets computation at a micro-level. These systems are finely-tuned to cater to specific tasks, offering optimized performance, power consumption, and space utilization, which are critical aspects in the deployment of tinyML solutions.

As we delve deeper into this chapter, we will unravel the complex, yet fascinating world of embedded systems, understanding their architecture, functionalities, and the pivotal role they play in facilitating tinyML applications. From exploring the basics of microcontroller units to understanding the interfaces and peripherals that enhance their functionalities, this chapter promises to be a rich resource for grasping the intricacies of embedded systems in the context of tinyML.

2.1 Introduction to Embedded Systems

2.1.1 Definition and Characteristics

Embedded systems are specialized computing systems that do not look like computers. They are dedicated to specific tasks and "embed" as part of a larger device. Unlike general-purpose computers that can run a wide variety of applications, embedded systems perform pre-defined tasks, often with very specific requirements. Since they are task-specific, their design ensures optimized performance and reliability. The characteristics that define these systems are as follows:

1. **Dedicated Functionality**: They are designed to execute a specific function or a set of closely related functions. This focus on specific tasks allows them to be optimized, offering faster performance and reliability.

2. **Real-Time Operation**: Many embedded systems operate in real-time, which means they are required to respond to inputs or changes in the environment immediately or within a predetermined time frame.

3. **Integration with Physical Hardware**: Embedded systems are closely integrated with physical hardware, making them more mechanically inclined compared to general-purpose computing systems.

4. **Long Lifecycle**: These systems typically have a long lifecycle and can continue to function for many years after their initial deployment.

5. **Resource Constraints**: Embedded systems are often resource-constrained, operating with limited computational power and memory. This necessitates the development of efficient algorithms and software.

2.1.2 Historical Background

Embedded systems have a rich history, with their roots tracing back to the 1960s when the first microprocessor, the Figure 2.1, made its debut. This paved the way for the creation of the first embedded system which was used in the Apollo Guidance Computer, the primary navigation system of the Apollo spacecraft. Over the years, the field has evolved dramatically, finding applications in various domains including automotive electronics, consumer electronics, telecommunications, and healthcare, among others.

2.1.3 Importance in tinyML

In the context of tinyML, embedded systems represent a significant frontier. The incorporation of machine learning models directly onto these systems facilitates intelligent decision-making at the edge, reducing latency and enhancing security. Here are several reasons why embedded systems are critical in the tinyML landscape:

1. **Edge Computing**: By bringing computation closer to the data source, embedded systems enhance efficiency and reduce the necessity for constant communication with centralized data centers.

2. **Low Power Consumption**: Embedded systems in tinyML are designed to consume minimal power, a critical requirement for battery-operated devices and IoT applications.

3. **Real-Time Analysis and Decision Making**: Embedded systems can facilitate real-time data analysis, allowing for immediate decision-making based on the insights generated.

4. **Security and Privacy**: Processing data locally on embedded systems ensures better security and privacy, as it reduces the chances of data interception during transmission.

5. **Cost-Effective**: Implementing ML models on embedded systems can be cost-effective, especially in scenarios where data transmission and storage in cloud servers might incur significant costs.
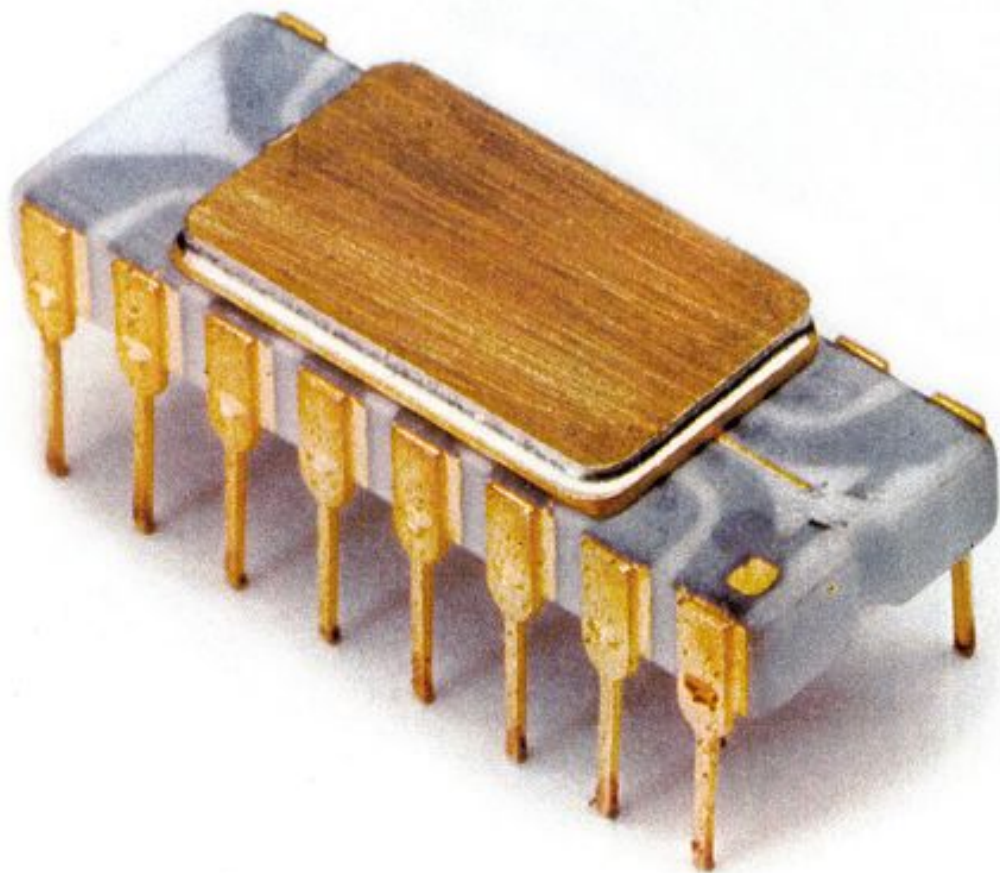
Figure 2.1: Intel 4004

As we venture deeper into this chapter, we will unveil the intricacies that govern the functioning of embedded systems and explore how they form the bedrock upon which tinyML stands, promising a future of integrated, intelligent, and efficient devices and systems.

2.2 Architecture of Embedded Systems

The architecture of embedded systems forms the blueprint that delineates the structure and functioning of these specialized systems. It provides insights into how different components within an embedded system interact and collaborate to achieve specific functionalities. This section dissects the integral components of the architecture - microcontrollers, microprocessors, different memory types and their management, and the intricacies of System on Chip (SoC).

2.2.1 Microcontrollers vs Microprocessors

Understanding the difference between microcontrollers and microprocessors is pivotal to grasping the fundamentals of embedded system architecture. Here, we delve into the characteristics of both:

- **Microcontrollers**

  Microcontrollers are compact, integrated circuits designed to govern specific operations in an embedded system. They house a processor, memory, and input/output peripherals in a single unit as shown in Figure 2.2, facilitating simplicity and ease of operation. Microcontrollers are typically used in products where the computational requirements are not highly demanding, and cost-effectiveness is a priority.

  Characteristics:

  - Single-chip solution
  - On-chip memory and peripherals
  - Low power consumption
  - Ideal for control-oriented applications

- **Microprocessors**

  On the other hand, microprocessors are more complex, forming the central processing unit within a system, lacking the integrated memory and I/O peripherals found in microcontrollers. They are usually found in systems that demand higher computational power and flexibility. These are used in devices where substantial processing power is required, and the tasks are more data-intensive.

  Characteristics:

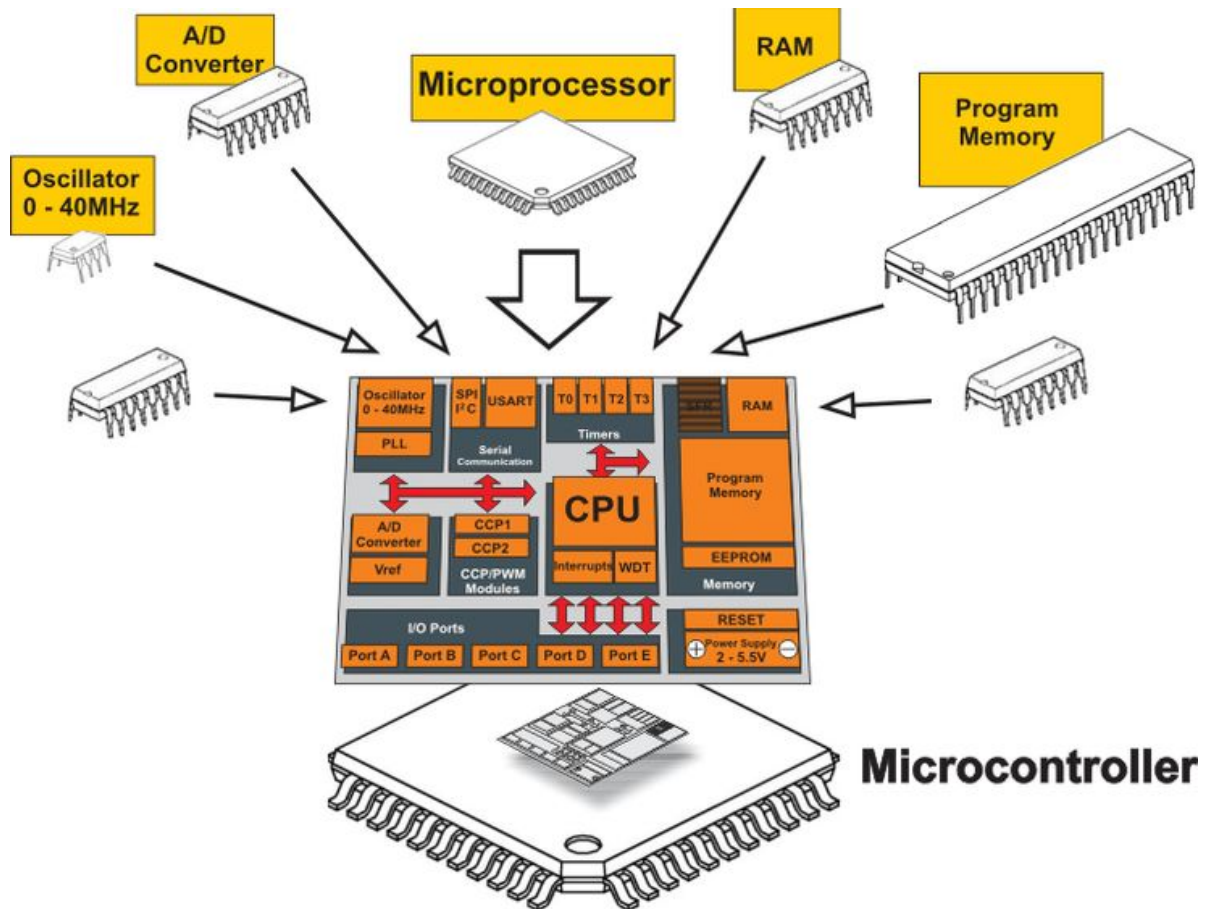 - Requires external components such as memory and I/O peripherals

Figure 2.2: Microcontrollers

– Higher processing power compared to microcontrollers
– More flexible in terms of connectivity with various components
– Ideal for data-intensive applications

2.2.2 Memory Types and Management

Embedded systems leverage various types of memory, each serving distinct purposes. Effective memory management is crucial to optimize performance and resource utilization. Below we discuss different memory types and how they are managed in an embedded system environment:

- **ROM (Read-Only Memory)**: This is non-volatile memory where data is written during manufacturing and remains unchanged throughout the device's life. It stores the firmware and boot-up instructions.

- **RAM (Random Access Memory)**: A volatile memory used to store temporary data generated during the system's operation. It is faster and allows read-write operations, but data is lost once power is turned off.

- **Flash Memory**: A non-volatile memory type that can be electrically erased and reprogrammed. It finds applications in storing firmware or data that needs to persist between reboots.

Memory Management:

- **Static Memory Allocation**: Memory is allocated before runtime, and the allocation does not change during the system's operation.

- **Dynamic Memory Allocation**: Memory is allocated at runtime, allowing flexibility but at the cost of increased complexity and potential memory leaks.

2.2.3 System on Chip (SoC)

Most embedded systems are SoCs. A System on Chip (SoC) represents an advanced integration technology where most of the components required to build a complete system are integrated onto a single chip. It usually contains a microprocessor or microcontroller, memory blocks, peripheral interfaces, and other components required for a fully functioning system. Here's a deeper look at its characteristics and applications:

- **Integration of Multiple Components**: SoCs house multiple components, including CPUs, memory, and peripherals, in a single chip, promoting higher integration levels and minimizing the need for external components.

- **Power Efficiency**: Due to the high level of integration, SoCs are often more power-efficient compared to systems built using separate chips.

- **Cost-Effectiveness**: The integration leads to reduced manufacturing costs, as fewer separate components are required.

- **Applications**: SoCs find applications in a variety of domains, including mobile computing, automotive electronics, and IoT devices, where compact size and power efficiency are prized.

Here are some examples of widely used SoCs that you may recognize given that they have found substantial applications across various domains:

1. **Qualcomm Snapdragon**: Predominantly found in smartphones and tablets, they offer a combination of processing power, graphics, and connectivity solutions.

2. **Apple A-series**: Custom SoCs developed by Apple, utilized in their range of iPhones, iPads, and even in some versions of Apple TV and HomePod. Notable examples include the A14 Bionic and A15 Bionic chips.

3. **Samsung Exynos**: Developed by Samsung, these are utilized extensively in their range of smartphones, tablets, and other electronic devices.

4. **NVIDIA Tegra**: Initially designed for mobile devices, they have found substantial applications in automotive and gaming consoles, like the Nintendo Switch. You can see a picture of it below in Figure 2.3.

5. **Intel Atom**: These are used in a wide variety of systems including netbooks, smartphones, and even embedded systems owing to their power efficiency.

6. **MediaTek Helio**: Popular in budget to mid-range smartphones, these chips offer a good balance of power efficiency and performance.

7. **Broadcom SoCs**: Used extensively in networking equipment, Broadcom offers a range of SoCs with different functionalities including those optimized for wireless communications and data processing.

8. **Texas Instruments (TI) OMAP**: These were popular in smartphones and tablets, offering a range of functionalities including multimedia processing and connectivity.

9. **Xilinx Zynq**: Predominantly used in embedded systems for industrial automation and for applications demanding high levels of data processing, such as advanced driver-assistance systems (ADAS).

10. **Altera SoC FPGA**: Now under Intel, these SoCs integrate FPGA technology with ARM cores, offering flexibility and performance for various applications including automotive and industrial systems.
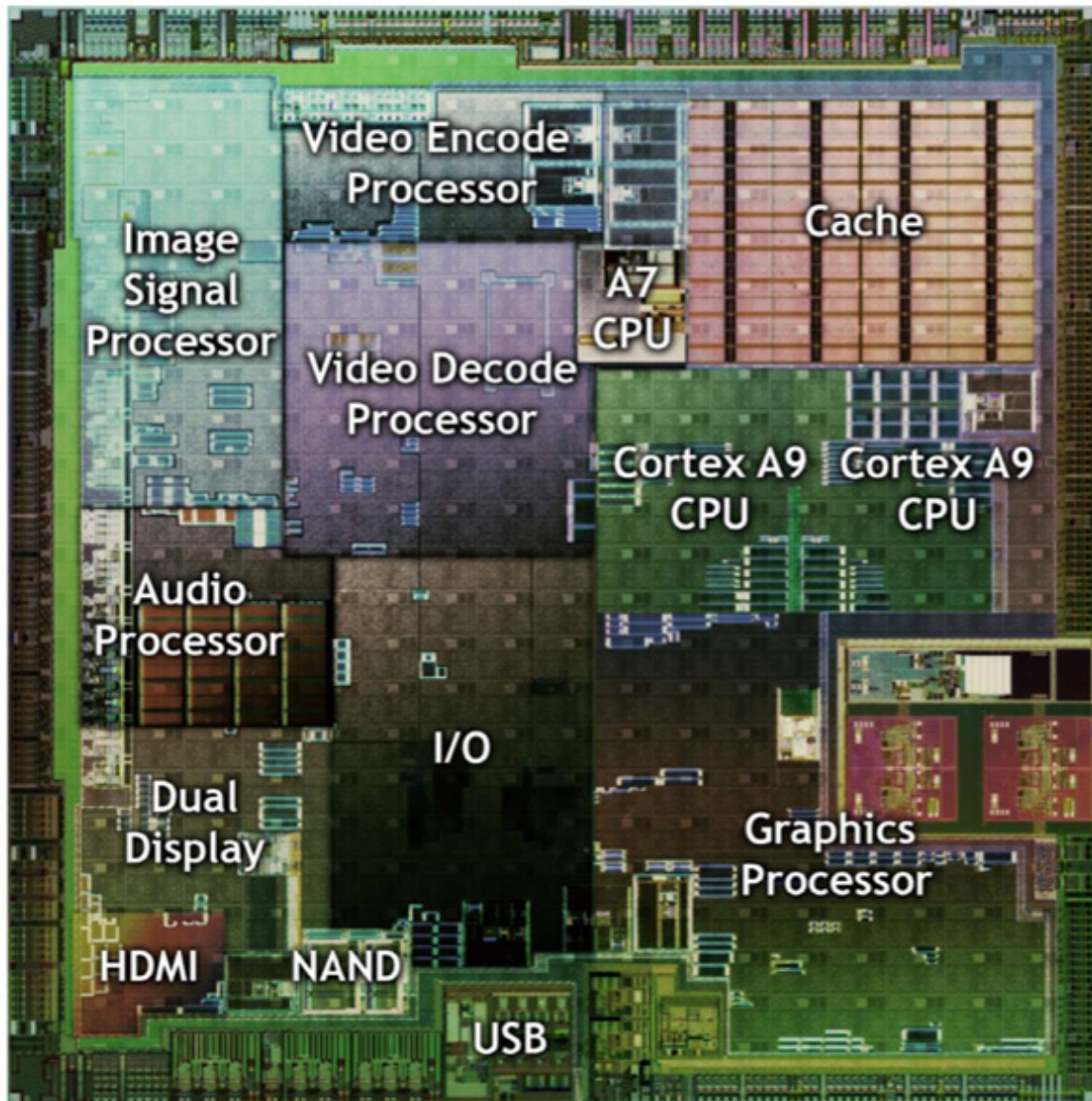
Figure 2.3: NVIDIA's Tegra 2 combines two ARM Cortex-A9 cores with an ARM7 for SoC management tasks.

Each of these SoCs presents a unique set of features and capabilities, catering to the diverse needs of the rapidly evolving technology landscape. They integrate multiple components into a single chip, offering power efficiency, cost-effectiveness, and compact solutions for modern electronic devices.

2.3 Embedded Systems Programming

Embedded systems programming diverges considerably from traditional software development, specifically honed to navigate the limited resources and the real-time requirements frequently associated with embedded hardware. This section will illuminate the nuances of the different programming languages utilized, delve into the intricacies of firmware development, and explore the critical role of Real-time Operating Systems (RTOS) in this specialized field.

2.3.1 Programming Languages: C, C++, Python, etc.

The selection of appropriate programming languages is crucial in embedded systems, often prioritizing direct hardware interaction and optimization of memory usage. Let us explore the specifics of these languages and how they stand apart from those typically utilized in more conventional systems:

- **C**: Traditionally the cornerstone of embedded systems programming, the C language facilitates direct interaction with hardware components, offering capabilities for bit-wise operations and manipulating memory addresses. Its procedural approach and low-level capabilities make it the favored choice for constrained environments, focusing on firmware development.

- **C++**: Building on the foundation laid by C, C++ integrates object-oriented principles, fostering organized and modular code development. Despite its inherent complexity, it is embraced in scenarios where higher-level abstractions do not compromise the granular control provided by C.

- **Python**: While not a classic choice for embedded systems due to its relative memory consumption and runtime delays, Python is finding its place in the embedded domain, especially in systems where resource constraints are less stringent. In recent times, a variant known as MicroPython has emerged, specifically tailored for microcontrollers. MicroPython retains the simplicity and ease of use of Python while being optimized for embedded environments, offering a flexible programming paradigm that facilitates rapid prototyping and development. For instance, the code snipped below shows how we can use MicroPython to interface with the pins on a PyBoard.

```python
import pyb # Package from PyBoard

# turn on an LED
pyb.LED(1).on()

# print some text to the serial console
print('Hello MicroPython!')
```

Comparison with Traditional Systems: In stark contrast to conventional systems, where languages like Java, Python, or JavaScript are celebrated for their development ease and comprehensive libraries, embedded systems are geared towards languages that offer refined control over hardware components and potential optimization opportunities, carefully navigating the limited resources at their disposal.

2.3.2 Firmware Development

The realm of firmware development within embedded systems encompasses crafting programs permanently stored in the non-volatile memory of hardware, thus ensuring persistent operation. Here, we delineate how it distinguishes itself from software development for traditional systems:

1. **Resource Optimization**: The necessity for constant optimization is paramount, enabling the code to function within the confines of restricted memory and processing capacities.

2. **Hardware Interaction**: Firmware typically exhibits a close-knit relationship with hardware, necessitating a profound comprehension of the hardware components and their functionalities.

3. **Lifecycle Management**: Firmware updates are less frequent compared to software updates in traditional systems, mandating rigorous testing procedures to avert failures that could culminate in hardware malfunctions.

4. **Security Concerns**: Given its integral role, firmware is a potential target for security breaches, warranting meticulous scrutiny towards security elements, including secure coding practices and encryption protocols.

2.3.3 Real-time Operating Systems (RTOS)

RTOS serve as the backbone for real-time systems, orchestrating task execution in a predictable, deterministic manner. This is a sharp deviation from the operating systems in mainstream computing environments, as delineated below:

1. **Deterministic Timing**: RTOS are structured to respond to inputs or events within a well-defined timeframe, meeting the critical time-sensitive requisites of many embedded systems.

2. **Task Prioritization**: They facilitate task prioritization, where critical tasks are accorded precedence in processing time allocation over less vital tasks.

3. **Microkernel Architecture**: A substantial number of RTOS leverage a microkernel architecture, epitomizing minimalism and efficiency by focusing only on the necessary functionalities to facilitate their operations.

4. **Memory Management**: Memory management in RTOS is often more streamlined compared to their counterparts in traditional operating systems, aiding in swift response times and operational efficacy.

Examples of RTOS: Notable examples in this category include FreeRTOS, RTEMS, and VxWorks, each offering unique features tailored to meet the diverse requirements of various embedded systems applications.

2.4 Interfaces and Peripherals

Embedded systems interact with the external world through various interfaces and peripherals, which are distinctly streamlined and specialized compared to general-purpose systems. Let's delve into the specifics:

2.4.1 Digital I/O

Digital Input/Output (I/O) interfaces are foundational in embedded systems, allowing them to interact with other devices and components. For example, a digital I/O pin can be used to read a binary signal (0 or 1) from sensors or to control actuators.

In embedded systems, these I/O ports often need to function under strict timing constraints, something which is less prevalent in general-purpose computing systems. Furthermore, these systems are usually programmed to perform specific, optimized operations on digital signals, sometimes needing to work in real time or near-real-time environments.

2.4.2 Analog Interfaces

Analog interfaces in embedded systems are crucial for interacting with a world that largely communicates in analog signals. These interfaces can include components like Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs). ADCs, for instance, can be used to read sensor data from environmental sensors like temperature or humidity

sensors, translating real-world analog data into a digital format that can be processed by the microcontroller.

Compared to general-purpose systems, embedded systems might employ analog interfaces in a more direct and frequent manner, especially in applications involving sensor integrations, which necessitate the conversion of a wide variety of analog signals to digital data for processing and analysis.

If you look closely enough in Figure 2.4, you will see there are indications of I/O pinouts for analog, digital, as well as communication layouts.

Figure 2.4: Nicla Vision pinout

2.4.3 Communication Protocols (SPI, I2C, UART, etc.)

Communication protocols serve as the conduits for facilitating communication between various components within or connected to an embedded system. Let's explore a few widely adopted ones:

- **SPI (Serial Peripheral Interface)**: This is a synchronous serial communication protocol, which is used for short-distance communication primarily in embedded systems. For example, it is often utilized in SD card and TFT display communications.

- **I2C (Inter-Integrated Circuit)**: This is a multi-master, multi-slave, packet switched, single-ended, serial communication bus, which is used widely in embedded systems to connect low-speed peripherals to a motherboard, embedded system, or cellphone. It's known for its simplicity and low pin count.

- **UART (Universal Asynchronous Receiver-Transmitter)**: This communication protocol allows for asynchronous serial communication between devices. It's widely used in embedded systems to transmit data between devices over a serial port, facilitating the transmission of data logs from a sensor node to a computer, for instance.

Compared to general-purpose systems, communication protocols in embedded systems are often more optimized for speed and reliability, as they may be used in critical applications where data transmission integrity is paramount. Moreover, they might be directly integrated into the microcontroller, emphasizing a more harmonized and seamless interaction between components, which is typically not observed in general-purpose systems.

2.5 Power Management in Embedded Systems

When engineering embedded systems, power management emerges as a pivotal focus area, shaping not only the system's efficiency but also its viability in real-world applications. The sheer diversity in the deployment of embedded systems, ranging from handheld devices to industrial machinery, underscores the imperative to optimize power management meticulously. Let's delve into this critical facet of embedded systems:

2.5.1 Power Consumption Considerations

In embedded systems, power consumption is a vital parameter that governs the system's performance and lifespan. Typically, microcontrollers in these systems operate in the range of 1.8V to 5V, with current consumption being in the microampere (A) to milliampere (mA) range during active modes. In sleep or standby modes, the current consumption can plummet to nanoamperes (nA), ensuring battery longevity.

Comparatively, general-purpose computing systems, like personal computers, consume power in the order of tens to hundreds of watts, which is several orders of magnitude higher. This stark contrast delineates the necessity for meticulous power management in embedded systems, where the available power budget is often significantly restrained.

The intricacies of managing power consumption hinge on a variety of factors including the operating voltage, clock frequency, and the specific tasks being performed by the system. Often, engineers find themselves navigating a complex trade-off landscape, balancing power consumption against system performance and responsiveness.

2.5.2 Energy-Efficient Design

Embedding energy efficiency into the design phase is integral to the successful deployment of embedded systems. Engineers often employ techniques such as dynamic voltage and frequency scaling (DVFS), which allows the system to adjust the voltage and frequency dynamically based on the processing requirements, thereby optimizing power consumption.

Additionally, leveraging low-power modes where non-essential peripherals are turned off or clock frequencies are reduced can significantly conserve power. For instance, utilizing deep sleep modes where the system consumes as little as 100 nA can dramatically enhance battery life, especially in battery-powered embedded systems.

In embedded systems, energy-efficient design isn't confined to just power-saving modes and techniques like Dynamic Voltage and Frequency Scaling (DVFS); it extends fundamentally to the architecture of the microcontroller itself, particularly in its instruction set architecture (ISA).

The microcontroller instruction set architecture (ISA) in embedded systems is often highly specialized, stripped of any unnecessary complexities that might add to power consumption. This specialization facilitates executing operations using a smaller number of cycles compared to general-purpose processors, which, in turn, reduces the power consumption per operation. Moreover, these specialized ISAs are crafted to efficiently execute the specific types of tasks that the embedded system is designed to perform, optimizing the execution path and thereby conserving energy.

Furthermore, it's not uncommon to find RISC (Reduced Instruction Set Computer) architectures in embedded systems. These architectures utilize a smaller set of simple instructions compared to Complex Instruction Set Computer (CISC) architectures found in traditional general-purpose systems. This design choice significantly reduces the power consumed per instruction, making these microcontrollers inherently more energy-efficient.

Apart from ISAs, embedded microcontrollers are often integrated with peripherals and components that are tailored to exhibit minimal energy expenditure, further reinforcing the emphasis on energy efficiency. Through careful design, engineers can craft systems that harmoniously integrate performance requirements with power management strategies, crafting solutions that

stand as testimony to innovation and sustainability in the field of embedded systems. This meticulous approach to design, focusing on both macro and micro-level optimizations, forms the bedrock of energy efficiency in embedded systems, differentiating them from their general-purpose counterparts which are often characterized by higher power consumption and a broader range of functionalities.

By focusing on these elements, engineers can forge pathways to create systems that not only fulfill their functional roles but do so with an acumen that reflects a deep understanding of the broader impacts of technology on society and the environment.

2.5.3 Battery Management

Battery management constitutes a vital part of power management strategies in embedded systems. The objective here is to maximize battery life without compromising system performance. Battery-powered embedded systems often employ lithium-ion or lithium-polymer batteries due to their high energy density and rechargeable nature. These batteries usually have a voltage range of 3.7V to 4.2V per cell. For instance, the Nicla Vision uses 3.7V battery as shown in Figure 2.5.
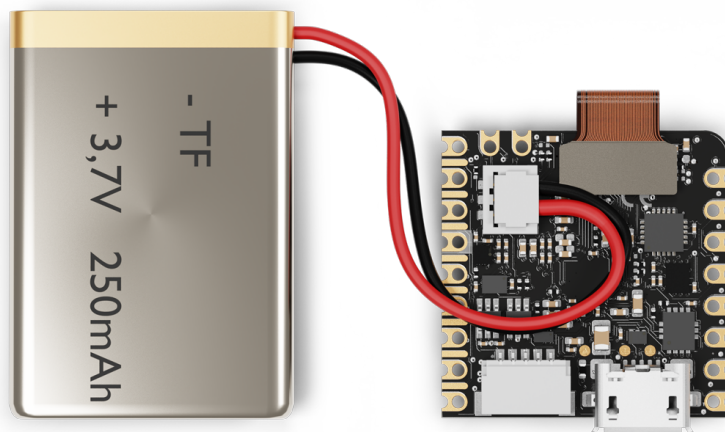

Figure 2.5: Nicla Vision battery.

Engineers need to incorporate strategies like efficient charge management, overvoltage protection, and temperature monitoring to safeguard the battery's health and prolong its lifespan. Moreover, developing systems that can harvest energy from the environment, like solar or vibrational energy, can supplement battery power and create sustainable, long-lasting solutions.

The focus on power management stems from the necessity to optimize resource utilization, extend battery life, and reduce operational costs. In deployments where systems are remote or inaccessible, efficient power management can significantly reduce the need for maintenance interventions, thereby ensuring sustained, uninterrupted operation.

One could say that power management in embedded systems is not just a technical requirement but a critical enabler that can dictate the success or failure of a deployment. Engineers invest significantly in optimizing power management strategies to craft systems that are not only efficient but also sustainable, showcasing a deep-seated commitment to innovation and excellence in the embedded systems domain.

2.6 Real-Time Characteristics

In the intricate fabric of embedded systems, the real-time characteristics stand as defining threads, weaving together components and tasks into a coherent, responsive entity. This facet, which is often unique to embedded systems, holds a critical place in the architecture and operation of these systems, providing them with the agility and precision to interact with their environment in a timely manner. Let's explore the intricacies that underline the real-time characteristics of embedded systems:

2.6.1 Real-time Clocks

Real-time clocks (RTCs) play a pivotal role in embedded systems, providing a precise time reference that governs the operations of the system. These clocks often have battery backups to ensure consistent timekeeping even when the main power source is unavailable. The utilization of RTCs is far more prevalent and critical in embedded systems compared to general-purpose computing systems, where timekeeping, although necessary, often doesn't dictate the system's core functionality.

For instance, in industrial automation systems, RTCs help in coordinating tasks with high precision, ensuring that processes occur in sync and without delay. They find significant applications in systems where time-stamped data logging is necessary, such as in environmental monitoring systems where data accuracy and time correlation are vital.

2.6.2 Timing and Synchronization

Timing and synchronization are hallmarks of embedded systems, where multiple components and processes need to work in harmony. The very essence of a real-time embedded system is dictated by its ability to perform tasks within a defined time frame. These systems usually have stringent timing requirements, demanding synchronization mechanisms that are both robust and precise.

For example, in automotive control systems, the timely and synchronized functioning of various sensors and actuators is non-negotiable to ensure safety and optimal performance. This is a stark contrast to general-purpose systems, where timing, although managed, doesn't often have immediate and critical repercussions.

2.6.3 Task Management and Scheduling

In embedded systems, task management and scheduling are critical to ensuring that the system can respond to real-time events effectively. Task schedulers in these systems might employ strategies such as priority scheduling, where tasks are assigned priority levels, and higher-priority tasks are allowed to pre-empt lower-priority tasks. This is particularly vital in systems where certain operations have a higher criticality.

For instance, in medical devices like pacemakers, the timely delivery of electrical pulses is a critical task, and the scheduling mechanism must prioritize this above all other tasks to ensure the patient's safety. This finely tuned scheduling and task management is quite unique to embedded systems, distinguishing them markedly from the more flexible and less deterministic scheduling observed in general-purpose systems.

2.6.4 Error Handling and Fault Tolerance

To further bolster their real-time characteristics, embedded systems often feature mechanisms for error handling and fault tolerance. These are designed to quickly identify and correct errors, or to maintain system operation even in the face of faults. In aviation control systems, for example, real-time fault tolerance is crucial to maintain flight stability and safety in drones. This level of meticulous error handling is somewhat distinctive to embedded systems compared to general-purpose systems, highlighting the critical nature of many embedded system applications.

The real-time characteristics of embedded systems set them apart, crafting a landscape where precision, synchrony, and timely responses are not just desired but mandatory. These characteristics find resonance in myriad applications, from automotive control systems to industrial automation and healthcare devices, underscoring the embedded systems' role as silent, yet powerful, orchestrators of a technologically harmonized world. Through their real-time attributes, embedded systems are able to deliver solutions that not only meet the functional

requirements but do so with a level of precision and reliability that is both remarkable and indispensable in the contemporary world. ## Security and Reliability

In a world that is ever-increasingly connected and reliant on technology, the topics of security and reliability have vaulted to the forefront of concerns in system design. Particularly in the realm of embedded systems, where these units are often integral parts in critical infrastructures and applications, the stakes are exponentially higher. Let's delve into the vital aspects that uphold the fortress of security and reliability in embedded systems:

2.7 Security and Reliability

2.7.1 Secure Boot and Root of Trust

As embedded systems find themselves at the heart of numerous critical applications, ensuring the authenticity and integrity of the system right from the moment of booting is paramount. The secure boot process is a cornerstone in this security paradigm, allowing the system to only execute code that is verified and trusted. This mechanism is often complemented by a "Root of Trust," an immutable and trusted environment, usually hardware-based, that validates the initial firmware and subsequent software layers during the boot process.

For instance, in financial transactions using Point-of-Sale (POS) terminals, a secure boot process ensures that the firmware is unaltered and secure, thwarting attempts of malicious firmware alterations which can potentially lead to significant data breaches. Similarly, in home automation systems, a robust secure boot process prevents unauthorized access, safeguarding user privacy and data.

2.7.2 Fault Tolerance

Fault tolerance is an indispensable attribute in embedded systems, bestowing the system with the resilience to continue operations even in the presence of faults or failures. This is achieved through various mechanisms like redundancy, where critical components are duplicated to take over in case of a failure, or through advanced error detection and correction techniques.

In applications such as aerospace and aviation, fault tolerance is not just a desirable feature but a mandatory requirement. Aircraft control systems, for instance, employ multiple redundant systems operating in parallel, ensuring continuous operation even in the case of a component failure. This level of fault tolerance ensures a high degree of reliability, making sure that the system can withstand failures without catastrophic consequences, a characteristic quite unique and elevated compared to traditional computing systems.

2.7.3 Safety-Critical Systems

Safety-critical systems are those where a failure could result in loss of life, significant property damage, or environmental harm. These systems require meticulous design to ensure the utmost reliability and safety. Embedded systems in this category often adhere to strict development standards and undergo rigorous testing to validate their reliability and safety characteristics.

For example, in automotive safety systems like Anti-lock Braking Systems (ABS) and Electronic Stability Control (ESC), embedded controllers play a pivotal role. These controllers are developed following stringent standards such as ISO 26262, ensuring that they meet the high reliability and safety requirements necessary to protect lives. In healthcare, devices like pacemakers and infusion pumps fall under this category, where the reliability of embedded systems can literally be a matter of life and death.

The emphasis on security and reliability in embedded systems cannot be overstated and I would state that these are often overlooked topics by most. As these systems intertwine deeper into the fabric of our daily lives and critical infrastructures, the doctrines of security and reliability stand as the beacon guiding the development and deployment of embedded systems. Through mechanisms like secure boot processes and fault tolerance techniques, these systems promise not only functional efficacy but also a shield of trust and safety, offering a robust and secure harbor in a sea of technological advancements and innovations. It's these foundational principles that shape the embedded systems of today, sculpting them into reliable guardians and efficient executors in various critical spheres of modern society.

2.8 Future Trends and Challenges

Arm, the largest manufacturer of microcontrollers, has shipped (either directly or indirectly through partners) a record 8.0 billion chips, taking total shipped to date to more than a quarter of a trillion or 250 billion [1]!

We stand on the cusp of an era of unprecedented growth in the field of embedded systems, it is both exciting and crucial to cast a discerning eye on the possible future trends and challenges that await us. From the burgeoning realms of edge computing to the demands of scalability, the landscape is set to evolve, bringing forth new vistas of opportunities and trials. Let's venture into the dynamic frontier that the future holds for embedded systems:

2.8.1 Edge Computing and IoT

With the proliferation of the Internet of Things (IoT), the role of edge computing is becoming increasingly vital. Edge computing essentially allows data processing at the source, reducing

latency and the load on central data centers. This paradigm shift is expected to redefine embedded systems, imbuing them with greater processing capabilities and intelligence to execute complex tasks locally.

Moreover, with the IoT expected to encompass billions of devices globally, embedded systems will play a central role in facilitating seamless connectivity and interoperability among a diverse array of devices. This ecosystem would foster real-time analytics and decision-making, paving the way for smarter cities, industries, and homes. The challenge here lies in developing systems that are secure, energy-efficient, and capable of handling the surge in data volumes efficiently.

2.8.2 Scalability and Upgradation

As embedded systems continue to evolve, the need for scalability and easy upgradation will become a focal point. Systems will be expected to adapt to changing technologies and user requirements without substantial overhauls. This calls for modular designs and open standards that allow for seamless integration of new features and functionalities.

Furthermore, with the rapid advancements in technology, embedded systems will need to be equipped with mechanisms for remote upgrades and maintenance, ensuring their longevity and relevance in a fast-paced technological landscape. The onus will be on developers and manufacturers to create systems that not only meet the current demands but are also primed for future expansions, thereby ensuring a sustainable and progressive development trajectory.

2.8.3 Market Opportunities

The market dynamics surrounding embedded systems are poised for exciting shifts. As industries increasingly adopt automation and digital transformation, the demand for sophisticated embedded systems is expected to soar. AI and ML are set to integrate deeper into embedded systems, offering unprecedented levels of intelligence and automation.

Simultaneously, there is a burgeoning market for embedded systems in consumer electronics, automotive, healthcare, and industrial applications, presenting vast opportunities for innovation and growth. However, this expansion also brings forth challenges, including increased competition and the need for compliance with evolving regulatory standards. Companies venturing into this space will need to be agile, innovative, and responsive to the changing market dynamics to carve a niche for themselves.

2.8.4 Conclusion

As we look into the horizon, it's evident that the world of embedded systems is on the brink of a transformative phase, marked by innovations, opportunities, and challenges. The future beckons with promises of greater connectivity, intelligence, and efficiency, forging a path where

embedded systems will be at the helm, steering the technological advancements of society. The journey ahead is one of exploration and adaptation, where the marriage of technology and ingenuity will craft a future that is not only technologically enriched but also responsive to the complex and ever-evolving demands of a dynamic world. It is a landscape ripe with potential, beckoning pioneers to venture forth and shape the contours of a promising and vibrant future.

3 Deep Learning Primer

3.1 Overview

3.1.1 Definition and Importance

Deep learning, a subset of machine learning and artificial intelligence (AI), involves algorithms inspired by the structure and function of the human brain, called artificial neural networks. It stands as a cornerstone in the field of AI, spearheading advancements in various domains including computer vision, natural language processing, and autonomous vehicles. Its relevance in embedded AI systems is underscored by its ability to facilitate complex computations and predictions, leveraging the limited resources available in embedded environments.

3.1.2 Brief History of Deep Learning

The concept of deep learning has its roots in the early artificial neural networks. It has witnessed several waves of popularity, starting with the introduction of the Perceptron in the 1950s, followed by the development of backpropagation algorithms in the 1980s.

The term "deep learning" emerged in the 2000s, marked by breakthroughs in computational power and data availability. Key milestones include the successful training of deep networks by Geoffrey Hinton, one of the god fathers of AI, and the resurgence of neural networks as a potent tool for data analysis and modeling.

In recent years, deep learning has witnessed exponential growth, becoming a transformative force across various industries. Figure 3.1 shows that we are currently in the third era of deep learning. From 1952 to 2010, computational growth followed an 18-month doubling pattern. This dramatically accelerated to a 6-month cycle from 2010 to 2022. At the same time, we witnessed the advent of major-scale models between 2015 and 2022; these appeared 2 to 3 orders of magnitude faster and followed a 10-month doubling cycle.

A confluence of factors has fueled this surge, including advancements in computational power, the proliferation of big data, and improvements in algorithmic designs. Firstly, the expansion of computational capabilities, particularly the advent of Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs), has significantly accelerated the training and inference times of deep learning models. These hardware advancements have made it feasible to construct and train more complex, deeper networks than were possible in the earlier years.

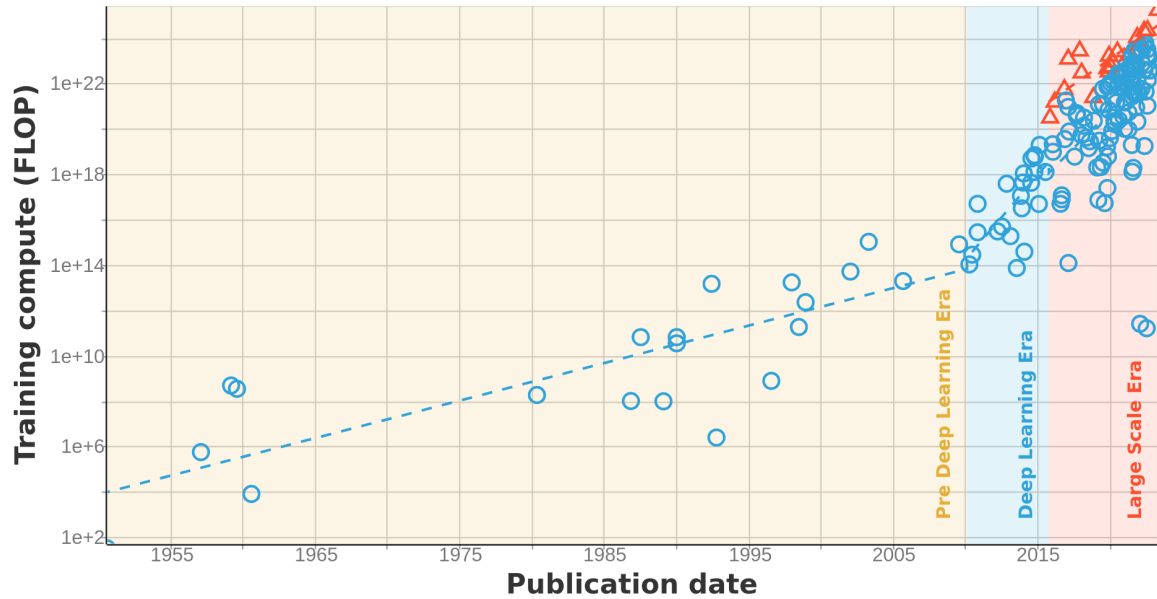**Training compute (FLOP) of milestone Machine Learning systems over time**

Figure 3.1: Growth of deep learning models.

Secondly, the digital revolution has brought forth an abundance of "big" data, providing rich material for deep learning models to learn from and excel in tasks such as image and speech recognition, language translation, and game playing. The availability of large, labeled datasets has been instrumental in the refinement and successful deployment of deep learning applications in real-world scenarios.

Additionally, collaborations and open-source initiatives have fostered a vibrant community of researchers and practitioners, propelling rapid advancements in deep learning techniques. Innovations such as deep reinforcement learning, transfer learning, and generative adversarial networks have expanded the boundaries of what is achievable with deep learning, opening new avenues and opportunities in various fields including healthcare, finance, transportation, and entertainment.

Companies and organizations worldwide are recognizing the transformative potential of deep learning, investing heavily in research and development to harness its power in offering innovative solutions, optimizing operations, and creating new business opportunities. As deep learning continues its upward trajectory, it is poised to revolutionize how we interact with technology, making our lives more convenient, safe, and connected.

3.1.3 Applications of Deep Learning

Deep learning is widely used in many industries today. It is used in finance for things such as stock market prediction, risk assessment, and fraud detection. It is also used in marketing for things such as customer segmentation, personalization, and content optimization. In healthcare, machine learning is used for tasks such as diagnosis, treatment planning, and patient monitoring. It has had a transformational impact on our society.

An example of the transformative impact that machine learning has had on society is how it has saved money and lives. For example, as mentioned earlier, deep learning algorithms can make predictions about stocks, like predicting whether they will go up or down. These predictions guide investment strategies and improve financial decisions. Similarly, deep learning can also make medical predictions to improve patient diagnosis and save lives. The possibilities are endless and the benefits are clear. Machine learning is not only able to make predictions with greater accuracy than humans but it is also able to do so at a much faster pace.

Deep learning has been applied to manufacturing to great effect. By using software to constantly learn from the vast amounts of data collected throughout the manufacturing process, companies are able to increase productivity while reducing wastage through improved efficiency. Companies are benefiting financially from these effects while customers are receiving better quality products at lower prices. Machine learning enables manufacturers to constantly improve their processes to create higher quality goods faster and more efficiently than ever before.

Deep learning has also improved products that we use daily like Netflix recommendations or Google Translate's text translations, but it also allows companies such as Amazon and Uber to save money on customer service costs by quickly identifying unhappy customers.

3.1.4 Relevance to Embedded AI

Embedded AI, which involves integrating AI algorithms directly into hardware devices, naturally benefits from the capabilities of deep learning. The synergy of deep learning algorithms with embedded systems has paved the way for intelligent, autonomous devices capable of sophisticated on-device data processing and analysis. Deep learning facilitates the extraction of intricate patterns and information from input data, making it a vital tool in the development of smart embedded systems, ranging from household appliances to industrial machines. This union aims to foster a new era of smart, interconnected devices that can learn and adapt to user behaviors and environmental conditions, optimizing performance and offering unprecedented levels of convenience and efficiency.

4 Embedded ML

4.1 CloudML

4.2 EdgeML

4.3 TinyML

4.3.1 TinyML for IoT Systems

4.3.2 How does TinyML Work

4.3.3 Resources are Limited, but so is the Competition

4.4 Exercises

5 ML Workflow

5.1 Data Collection

5.2 Pre-Processing

5.3 Training

5.4 Optimization

5.5 Deployment

5.6 Evaluation

5.7 Quiz

6 Data Engineering

6.1 Data Sources

6.2 Training Data

6.3 Training Data Splits

6.4 Data Labeling

6.5 Types of Data

7 Pre-processing

7.1 What is Data Pre-processing?

7.2 What's Involved with Data Pre-processing?

7.3 What's The Importance Of Data Pre-Processing?

8 ML Frameworks

coming soon.

9 Model Training

9.1 Selecting a Training Dataset

9.2 Neural Network Architectures

9.2.1 Multilayer Perceptron (MLP)

9.2.2 Convolutional Neural Networks

9.2.3 Recurrent Neural Networks

9.2.4 Transformers

9.3 Back Propagation

9.4 Convergence

9.5 Overfitting and Underfitting

9.6 Hyperparameters

9.6.1 Epochs

9.6.2 Learning Rate

9.7 Transfer Learning

9.7.1 Optimizer

9.8 Summary

9.9 Quiz

10 Efficient AI

This is an efficient test of a forked repo.

11 Optimizations

11.1 Software Optimizations

11.1.1 Compression

11.1.2 Quantization

11.1.3 Weight Pruning

11.1.4 Knowledge Distillation

11.2 Hardware Optimizations

11.2.1 GPUs

11.2.2 TPUs

11.2.3 NPUs

12 Deployment

13 On-Device Learning

13.1 Federated Learning

13.2 On-Device Training

coming soon.

14 Hardware Acceleration

coming soon.

15 MLOps

16 AI Sustainability

coming soon.

17 Responsible AI

coming soon.

18 Generative AI

coming soon.

References

[1] ARM.com. *The future is being built on Arm: Market diversification continues to drive strong royalty and licensing growth as ecosystem reaches quarter of a trillion chips milestone – Arm®.* https://www.arm.com/company/news/2023/02/arm-announces-q3-fy22-results. (Accessed on 09/16/2023).