Embedded AI - Part 1: Foundations and Optimization

1. Introduction to Embedded AI Hardware & Software

1.1 Latest Prototyping Devices

Arduino Nano 33 BLE Sense

- Microcontroller with built-in sensors (IMU, microphone, temperature, humidity, pressure, light, gesture)
- ARM Cortex-M4 processor
- Bluetooth Low Energy connectivity
- Perfect for TinyML applications

Raspberry Pi AI Kit

- More powerful than microcontrollers
- Runs full operating systems (Linux-based)
- Can handle complex AI models
- Better for prototyping before deployment to smaller devices

Pairbone Controller (likely "Bare-bone Controller")

- Minimal microcontroller setup
- No operating system, runs directly on hardware
- Used for understanding fundamental concepts

1.2 Operating Systems & AI-ML Frameworks

Operating Systems:

- **RTOS (Real-Time Operating System)**: FreeRTOS, Mbed OS - for time-critical tasks
- **Linux-based**: Raspberry Pi OS - for more powerful edge devices
- **Bare Metal**: No OS, direct hardware programming

AI-ML Frameworks:

- **TensorFlow Lite Micro**: Optimized for microcontrollers
- **Edge Impulse**: End-to-end platform for embedded ML
- **CMSIS-NN**: ARM's neural network library
- **PyTorch Mobile**: For edge devices

---

2. Why Embedded AI?

2.1 The Need for Embedded Systems

Embedded Systems are specialized computing systems designed to perform dedicated functions within larger systems.

Examples:

- Washing machine controllers
- Car engine management systems
- Smart thermostats
- Fitness trackers

Why Embedded?

- **Cost-effective**: Single-purpose, cheaper than general computers
- **Power-efficient**: Low power consumption
- **Real-time**: Guaranteed response times
- **Reliable**: Designed for specific tasks
- **Compact**: Small form factor

2.2 Why Embedded AI? (Edge AI)

Edge AI = Running AI/ML models directly on edge devices (not in the cloud)

Key Advantages:

1. **Low Latency**: Instant response
    - Example: Self-driving car needs millisecond decisions, can't wait for cloud
2. **Privacy**: Data stays on device
    - Example: Voice assistants processing wake words locally
3. **Bandwidth Efficiency**: No constant data transmission
    - Example: Smart camera analyzing video locally, only sending alerts
4. **Offline Operation**: Works without internet
    - Example: Industrial sensors in remote locations
5. **Cost Reduction**: Lower cloud computing costs

- Example: Millions of IoT devices would be expensive to run in cloud

6. **Real-time Processing**: Immediate action
    - Example: Predictive maintenance on machinery

Comparison Table:

Feature	Cloud AI	Edge AI
Latency	100-500ms	<10ms
Privacy	Data sent to cloud	Data stays local
Internet	Required	Not required
Power	Device: Low, Cloud: High	Device handles all
Scalability	Easy	Device-dependent

3. Bare Metal Programming

Bare Metal = Programming directly on hardware without an operating system

Super Loop Architecture:

```
Initialize Hardware
WHILE (forever) {
   Read Sensors
   Process Data
   Control Actuators
   Communicate
}
```

Example:

```
c
```

```
void main() {
    setup_hardware();

    while(1) {  // Super loop
        temperature = read_sensor();
        if(temperature > 30) {
            turn_on_fan();
        }
        send_data_to_display();
        delay(100);
    }
}
```

Advantages:

- Maximum control over hardware
- Predictable timing
- Minimal overhead

Disadvantages:

- Hard to manage complex tasks
- No multitasking

---

4. How Microprocessors & Microcontrollers are Manufactured

4.1 Manufacturing Process

1. **Design Phase**: Circuit design using HDL (Hardware Description Language)
2. **Fabrication**: Silicon wafer processing
    - Photolithography: Pattern circuits on silicon
    - Doping: Adding impurities to create transistors
    - Layering: Multiple layers of circuits
3. **Testing**: Each chip tested for defects
4. **Packaging**: Chip encased in protective material with pins

4.2 Customized Hardware

ASIC (Application-Specific Integrated Circuit)

- Custom chips designed for specific tasks
- Example: Google's TPU for AI, Apple's Neural Engine

FPGA (Field-Programmable Gate Array)

- Reconfigurable hardware
- Can be programmed after manufacturing
- Used for prototyping custom hardware

Why Customization?

- Better performance for specific tasks
- Lower power consumption
- Optimized for AI operations (matrix multiplications)

---

5. Edge Impulse Platform

Edge Impulse is an end-to-end development platform for embedded machine learning.

5.1 Workflow

1. **Data Collection**
    - Collect sensor data (audio, motion, images)
    - Label data (classify into categories)
    - Example: Collect "running" vs "walking" accelerometer data
2. **Model Training**
    - Choose processing blocks (spectral analysis, image preprocessing)
    - Select learning block (neural network architecture)
    - Train model in the cloud
3. **Testing & Validation**
    - Test on validation dataset
    - Live classification on device
4. **Deployment**
    - Export optimized model
    - Deploy to target device (Arduino, Raspberry Pi, etc.)

5.2 Classification Matrix (Confusion Matrix)

Shows how well your model performs.

Example 1: 2-Class Classification (Sitting vs Standing)

```
            Predicted
            Sitting  Standing
Actual Sitting   90       10
       Standing  15       85
```

Interpretation:

- **Accuracy**: (90+85)/(90+10+15+85) = 175/200 = 87.5%
- **Precision (Sitting)**: 90/(90+15) = 85.7%
- **Recall (Sitting)**: 90/(90+10) = 90%

Example 2: 3-Class Classification (Walking, Running, Idle)

```
            Predicted
            Walk  Run  Idle
Actual Walk   80    5    15
       Run     3   92     5
       Idle   10    8    82
```

Interpretation:

- Model confuses "Walk" with "Idle" sometimes (15 cases)
- "Run" is classified best (92% correct)
- Overall Accuracy: (80+92+82)/300 = 84.7%

---

6. TinyML (Tiny Machine Learning)

TinyML = Machine learning on microcontrollers and edge devices with <1MB memory and <1mW power

6.1 Characteristics

- **Ultra-low power**: Runs on batteries for months/years
- **Small memory**: Models fit in kilobytes
- **Low latency**: Real-time inference
- **Cost-effective**: Uses cheap microcontrollers

6.2 Applications

- **Predictive maintenance**: Vibration analysis on motors
- **Keyword spotting**: "Hey Google" detection
- **Gesture recognition**: Controlling devices with hand movements
- **Anomaly detection**: Detecting unusual patterns in sensor data

6.3 Can All Microcontrollers Run AI?

NO, not all microcontrollers can run AI. Here's why:

Requirements for AI on MCU:

1. **Sufficient Memory**
   - Need RAM for model and intermediate calculations
   - Need Flash for storing model weights
   - Minimum: 256KB Flash, 64KB RAM
2. **Processing Power**
   - Need to perform many calculations quickly
   - ARM Cortex-M4 or better preferred
3. **Hardware Accelerators** (optional but helpful)
   - DSP (Digital Signal Processing) instructions
   - FPU (Floating Point Unit)
   - Neural network accelerators

Examples:

Microcontroller	Can Run AI?	Why?
ATtiny85	NO	Only 8KB Flash, 512B RAM
Arduino Uno (ATmega328)	Limited	32KB Flash, 2KB RAM - only very tiny models
Arduino Nano 33 BLE	YES	1MB Flash, 256KB RAM, Cortex-M4
ESP32	YES	4MB Flash, 520KB RAM, dual-core
STM32H7	YES	2MB Flash, 1MB RAM, Cortex-M7, AI accelerators

How Do Capable MCUs Run AI?

1. **Model Optimization**: Compress neural networks
2. **Quantization**: Use 8-bit integers instead of 32-bit floats
3. **Pruning**: Remove unnecessary connections
4. **Efficient Libraries**: CMSIS-NN, TensorFlow Lite Micro
5. **Hardware Acceleration**: Use DSP/FPU instructions

---

7. Signal Processing Fundamentals

7.1 Sampling

Sampling = Converting continuous analog signals to discrete digital values

Key Concept: Nyquist-Shannon Sampling Theorem

- **Rule**: Sampling frequency must be **at least 2× the maximum frequency** in the signal
- Better practice: **2.2× to 2.5× max frequency** (to avoid aliasing)

Example 1:

- Human voice: Max frequency ≈ 4 kHz
- Required sampling rate: 2.2 × 4,000 = 8,800 Hz (often use 8 kHz or 16 kHz)

Example 2:

- Audio CD quality: Max frequency = 20 kHz
- Sampling rate: 2.2 × 20,000 = 44,000 Hz (CDs use 44.1 kHz)

7.2 Quantization

Quantization = Converting continuous amplitude values to discrete levels

Simple Explanation: Imagine a thermometer that can show ANY temperature (analog), but you can only write down whole numbers (digital). That's quantization.

Bit Resolution:

- **N bits** = **2^N levels**

Examples:

2-bit Digitization:

- 2^2 = 4 levels
- If voltage range is 0-3V: 0V, 1V, 2V, 3V
- Resolution: 3V/4 = 0.75V per step

3-bit Resolution:

- 2^3 = 8 levels
- If voltage range is 0-3.5V: 0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5V
- Resolution: 3.5V/8 = 0.4375V per step

Arduino Resolution:

- **10-bit ADC** (Analog-to-Digital Converter)
- 2^10 = 1,024 levels
- Voltage range: 0-5V
- Resolution: 5V/1024 = 0.00488V ≈ 4.88mV per step

Brain-Computer Interface (BCI) Requirements:

- EEG signals: Very small (microvolts)
- Need **16-bit to 24-bit ADC**
- 24-bit = 16,777,216 levels for precise measurement
- Higher resolution captures tiny brain signals

7.3 Sensor Mapping to Device

How Sensors Connect:

Sensors output **voltage** that represents physical quantities.

Example: Temperature Sensor (LM35)

- Output: 10mV per °C
- At 25°C: Output = 250mV = 0.25V
- Arduino reads this voltage and converts to temperature

Mapping Process:

1. Sensor produces voltage (analog signal)
2. ADC converts voltage to digital number

3. Software maps number to meaningful value

Arduino Example:

```c
int sensorValue = analogRead(A0);  // Read ADC (0-1023)
float voltage = sensorValue * (5.0/1023.0);  // Convert to voltage
float temperature = voltage * 100;  // For LM35: 10mV/°C
```

7.4 Data Rate Calculation

Formula: Data Rate (bps) = Sampling Frequency × Bit Resolution × Number of Channels

Example Problem 1:

- Audio signal: Max frequency = 4 kHz
- Sampling frequency = 2.2 × 4,000 = 8,800 Hz
- Quantization: 8-bit
- Channels: Mono (1 channel)
- **Data Rate** = 8,800 × 8 × 1 = **70,400 bps = 70.4 kbps**

Example Problem 2:

- Stereo music: Max frequency = 20 kHz
- Sampling frequency = 44,100 Hz
- Quantization: 16-bit
- Channels: Stereo (2 channels)
- **Data Rate** = 44,100 × 16 × 2 = **1,411,200 bps = 1.41 Mbps**

Example Problem 3:

- Given: Sampling rate = 1 kHz, 12-bit resolution, 3 sensors
- **Data Rate** = 1,000 × 12 × 3 = **36,000 bps = 36 kbps**
- **Data per second** = 36,000 bits / 8 = 4,500 bytes/second
- **Data per hour** = 4,500 × 3,600 = 16.2 MB/hour

7.5 Fourier Transform & FFT

Fourier Transform:

- Converts time-domain signals to frequency-domain

- Shows which frequencies are present in a signal

Simple Analogy: A musical chord contains multiple notes. Fourier Transform separates the chord back into individual notes.

FFT (Fast Fourier Transform):

- Efficient algorithm for computing Fourier Transform
- Crucial for audio processing, vibration analysis
- Used in TinyML for feature extraction

Example Use:

- Audio signal → FFT → Frequency spectrum → Neural network → Classification

7.6 Signal Reconstruction

Reconstruction = Converting digital signal back to analog

Process:

1. Digital values → DAC (Digital-to-Analog Converter)
2. DAC outputs stepped voltage
3. Low-pass filter smooths the steps
4. Result: Reconstructed analog signal

Quality depends on:

- Sampling rate (higher = better)
- Bit resolution (higher = smoother)
- Filter quality

---

8. Binary Encoding

Binary Encoding = Representing information in binary (0s and 1s)

8.1 Basic Concepts

1 bit = 0 or 1 **1 byte** = 8 bits **1 kilobyte** = 1,024 bytes

8.2 Representing Numbers

Unsigned Integer (8-bit):

- Range: 0 to 255
- Example: 01101101 = 64+32+8+4+1 = 109

Signed Integer (8-bit, Two's Complement):

- Range: -128 to 127
- Example: 11111111 = -1

8.3 Fixed-Point vs Floating-Point

Fixed-Point:

- Used in embedded systems
- Faster, less memory
- Example: 16-bit number, 8 bits integer, 8 bits fraction

Floating-Point:

- Standard in AI (32-bit float)
- More flexible, larger range
- Slower on microcontrollers without FPU

---

9. ARM Processor Evolution

9.1 Historical Timeline

Traditional ARM Cortex-M Family:

1. **Cortex-M0/M0+** (2009)
    - Ultra-low power
    - 32-bit RISC
    - For simple control tasks
2. **Cortex-M3** (2004)
    - Better performance
    - 3-stage pipeline
    - For general embedded applications
3. **Cortex-M4** (2010)
    - Added DSP instructions

- Optional FPU (Floating Point Unit)
- Good for signal processing
- **First practical for TinyML**

4. **Cortex-M7** (2014)
    - High performance
    - 6-stage pipeline
    - Double-precision FPU
    - For complex applications

9.2 Game Changer: Cortex-M55 (2020)

Why Game Changer?

- **First M-series with Helium technology**
- **M-Profile Vector Extension (MVE)**
- **Up to 15× better ML performance** than M4
- **AI-optimized instructions**

Features:

- 128-bit vector processing
- Custom instructions for neural networks
- Works with Ethos-U55/U65 NPUs

Use Cases:

- Keyword spotting
- Vibration analysis
- Sensor fusion
- Image classification

9.3 Dedicated AI Chips: Ethos-U NPUs

Ethos-U55 & Ethos-U65 = Neural Processing Units

What are NPUs?

- Specialized hardware for neural network operations
- Like a GPU but for embedded systems

Comparison:

Feature	Cortex-M55	Ethos-U55	Ethos-U65
Type	CPU with vector	Micro-NPU	Micro-NPU
Performance	Baseline	32-256 MAC/cycle	128-512 MAC/cycle
Power	Low	Ultra-low	Low
Use Case	General ML	Tiny models	Larger models

MAC (Multiply-Accumulate) = Basic operation in neural networks

Integration:

> Cortex-M55 CPU + Ethos-U55 NPU = Complete AI System
> - CPU: Runs application code, preprocessing
> - NPU: Accelerates neural network inference
> - Result: 50-100× faster ML than M4 alone

Use Cases:

- **Predictive Maintenance**: Analyze motor vibrations (M55 + U55)
- **Smart Home**: Voice control with keyword spotting (M55 + U55)
- **Wearables**: Gesture recognition (M55 + U55)
- **Vision**: Object detection on cameras (M55 + U65)

---

10. ARM Ecosystem

10.1 Complete ARM Ecosystem for AI

Hardware:

- **Cortex-M Processors**: M4, M7, M55
- **Ethos NPUs**: U55, U65
- **Mali GPUs**: For higher-end devices

Software:

- **CMSIS (Cortex Microcontroller Software Interface Standard)**

- **CMSIS-NN**: Neural network kernels
- **Mbed OS**: Operating system
- **Arm NN**: Neural network framework

Tools:

- **Arm Compiler**: Optimized code generation
- **Vela Optimizer**: Optimizes models for Ethos NPUs
- **Arm Development Studio**: IDE

Workflow:

```
TensorFlow Model
  ↓
TensorFlow Lite Converter
  ↓
TFLite Model
  ↓
Vela Optimizer (for Ethos NPU)
  ↓
Optimized Model
  ↓
Deploy to ARM Cortex-M + CMSIS-NN
  ↓
Run Inference
```

---

11. TensorFlow Lite (TF Lite)

TensorFlow Lite = Lightweight version of TensorFlow for mobile and embedded devices

11.1 Why TF Lite?

- **Small model size**: Uses quantization and compression
- **Fast inference**: Optimized operators
- **Cross-platform**: Android, iOS, Linux, MCUs
- **Hardware acceleration**: GPU, NPU support

11.2 TF Lite for Microcontrollers (TF Lite Micro)

Key Features:

- No operating system required
- <100 KB footprint
- Runs on Cortex-M microcontrollers
- Integer-only arithmetic (int8)

11.3 Conversion Process

```python
# 1. Train model in TensorFlow
model = create_and_train_model()

# 2. Convert to TF Lite
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()

# 3. Quantize to int8 (for Micro)
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
tflite_quantized_model = converter.convert()

# 4. Save model
with open('model.tflite', 'wb') as f:
    f.write(tflite_quantized_model)
```

---

12. Vela Optimizer

Vela = Compiler that optimizes TensorFlow Lite models for Ethos-U NPUs

12.1 What Does Vela Do?

1. **Layer Fusion**: Combines operations to reduce memory access
2. **Memory Optimization**: Minimizes SRAM usage
3. **Scheduling**: Determines best execution order
4. **Quantization**: Ensures proper int8 format

12.2 Usage

```bash
```

```
vela model.tflite \
  --accelerator-config=ethos-u55-256 \
  --optimise Performance \
  --output-dir output/
```

Result: Optimized `.tflite` file that runs on Ethos-U NPU

12.3 Performance Gains

- **Without Vela**: Model runs on Cortex-M CPU only
- **With Vela**: Model runs on Ethos-U NPU
- **Speed-up**: 10-50× faster inference
- **Power savings**: 50-80% lower energy per inference

---

13. CMSIS-NN

CMSIS-NN = ARM's library of optimized neural network functions

13.1 What is CMSIS-NN?

- **Collection of kernels** for neural network operations
- **Highly optimized** for ARM Cortex-M processors
- **Used by TF Lite Micro** as backend

13.2 Key Functions

Convolution:

```c
arm_convolve_HWC_q7_basic()  // 2D convolution for CNNs
```

Fully Connected:

```c
arm_fully_connected_q7()  // Dense layer
```

Pooling:

```c
```

```
arm_maxpool_q7_HWC()  // Max pooling
```

Activation:

```
c

arm_relu_q7()  // ReLU activation
```

13.3 How CMSIS-NN Works

Without CMSIS-NN (Naive Implementation):

```
c

// Simple convolution - SLOW
for each output pixel:
   for each filter:
      for each input channel:
         sum += input * weight
```

With CMSIS-NN (Optimized):

- Uses SIMD (Single Instruction Multiple Data)
- Processes 4 values at once (for int8)
- Loop unrolling
- Efficient memory access patterns
- **Result**: 4-5× faster

Example:

```
c

// CMSIS-NN processes 4 bytes simultaneously
uint32_t packed_input = *((uint32_t*)&input[i]);
// Now can multiply 4 values in one instruction
```

---

14. Why Cortex-M Processors for AI?

14.1 Advantages

1. **Power Efficiency**
   - 0.1-100 mW typical power

- Can run on batteries for years

2. **Cost**
    - $0.50 - $10 per chip
    - Much cheaper than GPUs or high-end CPUs
3. **Real-Time Capabilities**
    - Deterministic timing
    - Interrupt handling
    - Critical for control systems
4. **Wide Availability**
    - Billions of devices use ARM Cortex-M
    - Large ecosystem and support
5. **Scalability**
    - M0 for simple tasks
    - M4/M7 for signal processing
    - M55 + Ethos-U for advanced AI

14.2 Comparison

Processor Type	Power	Cost	AI Capability	Use Case
Cortex-M0	1-10mW	$0.50	None	Simple control
Cortex-M4	10-50mW	$2-5	Basic TinyML	Audio, sensors
Cortex-M55	20-100mW	$5-10	Advanced TinyML	Complex ML
Raspberry Pi	2-4W	$35+	Full ML	Prototyping
Jetson Nano	5-10W	$100+	Deep Learning	Vision, robotics

---

15. Why Optimization Matters

15.1 Constraints in Embedded Systems

Power Constraints:

- Battery-powered devices

- Example: Smartwatch must last days, not hours
- **AI inference should use <1% of battery per day**

Cycle Constraints:

- Limited processing speed (10-200 MHz typical)
- Must complete inference within time budget
- Example: Keyword detection must respond in <100ms

Memory Constraints:

- Flash: 256 KB - 2 MB (for model storage)
- RAM: 64 KB - 512 KB (for activations)
- **Model must fit in available memory**

15.2 Optimization Strategies

Four main categories:

1. **Model Optimization**
2. **Library Optimization**
3. **Hardware Design Optimization**
4. **Software Environment Optimization**

---

16. Optimization Techniques

16.1 Model Optimization

A. Quantization

- **Float32 → Int8**: 75% size reduction
- **Accuracy loss**: 1-2% typical
- **Speed-up**: 2-4×

Example:

- Original model: 2 MB (float32)
- Quantized model: 500 KB (int8)
- Can now fit on microcontroller!

B. Pruning

- Remove unnecessary connections
- Set small weights to zero
- **Result**: 30-50% fewer parameters

C. Knowledge Distillation

- Train small model to mimic large model
- Smaller model learns from teacher's outputs

D. Neural Architecture Search (NAS)

- Automatically find efficient architectures
- Examples: MobileNet, EfficientNet

16.2 Library Optimization

CMSIS-NN Optimization Techniques:

1. **SIMD Instructions**
    - Process multiple data points simultaneously
    - Example: 4 int8 multiplications in one cycle
2. **Loop Unrolling**
    - Reduce loop overhead
    - Trade code size for speed
3. **Memory Access Optimization**
    - Minimize cache misses
    - Efficient data layout

Performance Impact:

- Naive implementation: 100 ms
- With CMSIS-NN: 20-25 ms
- **Speed-up: 4-5×**

16.3 Hardware Design Optimization

1. Use Hardware Accelerators

- FPU for floating-point operations
- DSP for signal processing
- NPU (Ethos-U) for neural networks

2. Memory Hierarchy

- Fast SRAM close to processor
- Slower Flash for storage
- Cache for frequently accessed data

3. Custom Hardware

- ASICs for specific AI tasks
- Example: Google's Edge TPU

16.4 Software Environment Optimization

A. Compiler Settings

Optimization Levels:

```c
// -O0: No optimization (debug)
// -O1: Basic optimization
// -O2: Moderate optimization (recommended)
// -O3: Aggressive optimization
// -Os: Optimize for size
```

Example:

```bash
arm-none-eabi-gcc -O3 -mcpu=cortex-m4 -mfpu=fpv4-sp-d16 main.c
```

Flags:

- `-O3`: Maximum speed optimization
- `-mcpu=cortex-m4`: Target Cortex-M4
- `-mfpu=fpv4-sp-d16`: Use hardware FPU

Performance Impact:

- `-O0`: 200 ms inference

- `-O2`: 100 ms inference
- `-O3`: 80 ms inference

B. Code Optimizations

1. Use Fixed-Point Instead of Float

```c
// Slow (float)
float result = a * b + c;

// Fast (fixed-point)
int32_t result = (a * b) >> 16 + c;  // Assumes 16-bit fraction
```

2. Reduce Function Calls

```c
// Slow
for(int i=0; i<1000; i++) {
    result += expensive_function(i);
}

// Fast - inline or macro
#define FAST_CALC(x) ((x) * 2 + 1)
for(int i=0; i<1000; i++) {
    result += FAST_CALC(i);
}
```

---

17. Optimization by Processor Type

17.1 Cortex-M4 Optimizations

Focus: DSP instructions, FPU usage

Techniques:

1. Enable FPU in compiler flags
2. Use CMSIS-DSP functions
3. Process data in batches (SIMD)

Example:

```c
// Use CMSIS-DSP for vector operations
arm_dot_prod_q15(vectorA, vectorB, length, &result);
// Much faster than manual loop
```

17.2 Cortex-M55 Optimizations

Focus: Helium vector extensions

Techniques:

1. Use MVE (M-Profile Vector Extension)
2. Process 128-bit vectors
3. Utilize CMSIS-NN optimized for M55

Performance:

- M4: 100 ms inference
- M55: 7-10 ms inference
- **Speed-up: 10-15×**

17.3 Cortex-M55 + Ethos-U55 Optimizations

Focus: Offload NN operations to NPU

Techniques:

1. Use Vela to optimize model
2. Partition workload: CPU for preprocessing, NPU for inference
3. Pipeline operations

Performance:

- M55 alone: 10 ms
- M55 + U55: 2-3 ms
- **Speed-up: 3-5× over M55 alone, 30-50× over M4**

---

18. Model Design Optimization

18.1 Memory Layout Optimization

Problem: Activations require a lot of RAM

Solutions:

1. In-Place Operations

- Reuse memory buffers
- Overwrite input with output when possible

2. Layer-by-Layer Execution

- Don't keep all activations in memory
- Compute one layer, discard, compute next

Example:

> Without optimization:
> Input: 10 KB
> Layer 1 output: 20 KB
> Layer 2 output: 15 KB
> Total RAM: 45 KB
>
> With optimization:
> Max RAM: 20 KB (largest layer output)
> Savings: 55% reduction

18.2 Channel Alignment

8-byte Alignment for efficient SIMD access

Example:

> Bad: 13 channels → not aligned
> Good: 16 channels → aligned to 8-byte boundary
>
> Result: 20-30% faster memory access

18.3 Convolution Types

Standard Convolution:

- Most accurate

- Most computationally expensive

Depthwise Separable Convolution (MobileNet):

- Separate spatial and channel convolutions
- **9×