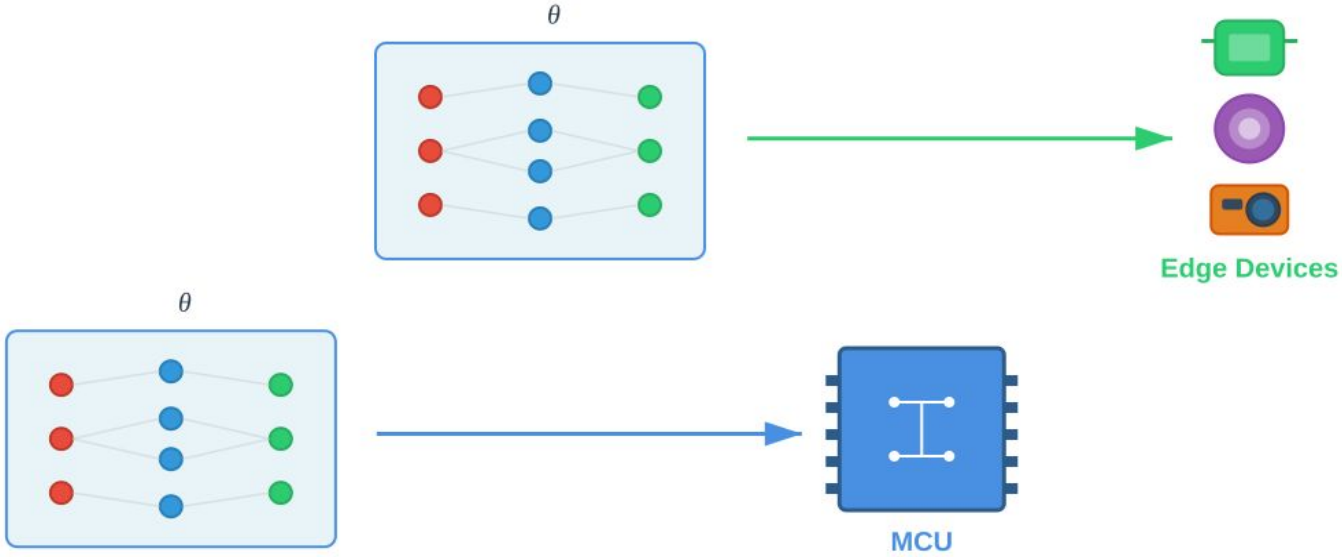
Edge AI & TinyML



Bringing Intelligence to the Edge

Presented by

Dr. Mohammed Alnemari

FALL2025

Dr. Mohammed Alnemari

- ASSISTANT PROFESSOR IN COMPUTER ENGINEERING SPECIALIZING IN ARTIFICIAL INTELLIGENCE IN EMBEDDED SYSTEMS
- ASSOCIATE RESEARCHER AT THE ARTIFICIAL INTELLIGENCE AND SENSING TECHNOLOGIES RESEARCH CENTER.
- AI CONSULTING AND ADVANCED TECHNOLOGIES

Educational Background:

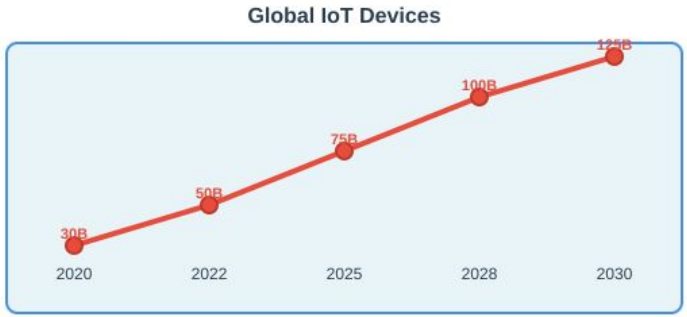
1. Bachelor's in Computer Engineering from Taif University (Saudi Arabia)
2. Bridge Program at the University of Pennsylvania (USA)
3. Master's degree from the University of California, Irvine (USA)
4. Visiting Master's student at National Taiwan University (Taiwan)
5. Visiting PhD student at Tokyo Institute of Technology (Japan)
6. PhD from the University of California, Irvine (USA)
7. Visiting Researcher at the University of Southern California (USA)

RESEARCH INTERESTS: EDGE AI, TinyML , NEURAL NETWORK COMPRESSION, EFFICIENT NEURAL NETWORKS, EMBEDDED MACHINE LEARNING

TABLE OF CONTENT

WHY EDGE AI?

The Data Explosion Challenge



Three Critical Limitations of Cloud Computing

1. Latency

- Network transmission delays
- Unsuitable for real-time
- T = T_transmit + T_compute

2. Privacy

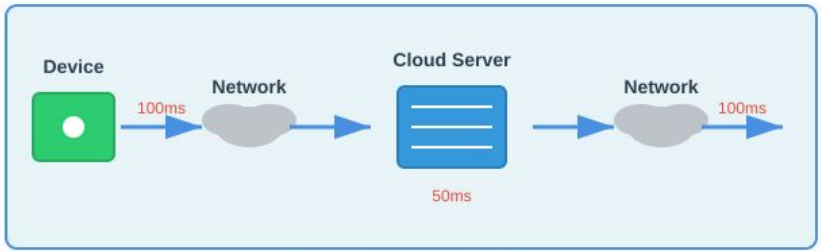
- Sensitive data exposure
- Regulatory compliance
- Data sovereignty concerns

3. Bandwidth

- Network congestion
- High operational costs
- Limited remote connectivity

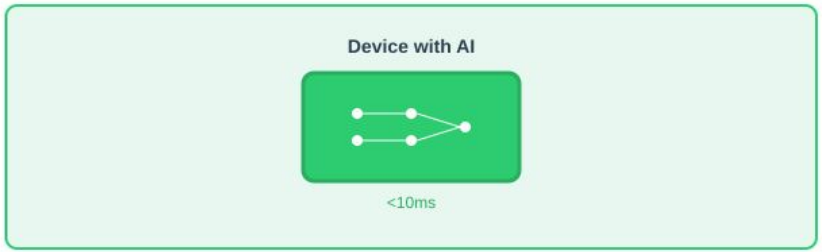
FROM CLOUD AI → EDGE AI

Cloud Inference Architecture



Total Latency: ~250ms

Edge Inference Architecture



Total Latency: <10ms

Mathematical Cost

$$T = T_{transmit} + T_{compute}$$

Where:

$T_{transmit}$ = Data_size / Bandwidth

$T_{compute}$ = Operations / FLOPS

Critical for Control Loops

Latency > 100ms can be catastrophic:

- Autonomous vehicles
- Industrial robotics
- Medical devices
- Energy grids

EDGE AI APPLICATIONS

Smart Home

- Voice assistants
- Security cameras
- Smart thermostats

Wearables

- Health monitoring
- Fitness tracking
- Fall detection

Agriculture

- Crop monitoring
- Soil analysis
- Irrigation control

Industrial IoT

- Predictive maintenance
- Quality control
- Process optimization

Autonomous Systems

- Self-driving vehicles
- Drones
- Navigation systems

Energy Systems

- Smart grids
- Solar monitoring
- Battery management

WHAT IS EDGE AI?

CORE CONCEPTS

Definition

Edge AI refers to artificial intelligence algorithms running locally on embedded or IoT devices at the "edge" of the network, rather than in centralized cloud servers.

Key Requirements

Low Power

- Battery-operated devices
- Energy harvesting compatible
- µW to mW range

Small Memory

- Limited RAM (<256 KB)
- Limited Flash (<1 MB)
- Efficient storage

Real-Time

- <10ms inference
- Deterministic behavior
- No network dependency

Edge AI Architecture

WHAT IS TINYML?

CORE CONCEPTS

Definition

TinyML is the deployment of machine learning models on microcontrollers (MCUs) and ultra-low-power embedded systems.

Typical MCU Specifications

Component	Specification
Processor	32-bit ARM Cortex-M4
Clock Speed	64-80 MHz
Flash Memory	512 KB - 1 MB
RAM	128-256 KB
Power Consumption	<1 mW (sleep), 50-100 mW (active)
Cost	$2-$10

TinyML Capabilities: Audio keyword spotting • Gesture recognition • Anomaly detection • Image classification • Sensor fusion

Community & Standards: IEEE TinyML community • MLPerf Tiny benchmark • TinyML Foundation

TINYML VS EDGE AI VS CLOUD AI

Comparative Analysis

Feature	Cloud AI	Edge AI	TinyML
Hardware	GPU/TPU Servers	Edge TPU, Mobile GPU	Microcontrollers
Memory	GBs to TBs	GBs	KBs to MBs
Power	kW	Watts	mW to µW
Latency	100-500ms	10-50ms	<10ms
Connectivity	Required	Optional	None
Privacy	Low (data sent)	Medium	High (local)
Model Size	Billions of params	Millions of params	Thousands of params
Cost per device	Subscription	$50-$200	$2-$10
Update Ability	Easy	Moderate	Challenging
Applications	Complex NLP, Vision	Autonomous vehicles	Sensors, wearables

CHALLENGES IN BRINGING AI TO THE EDGE

Memory Power Learning

TECHNICAL FOUNDATIONS

1. Limited Compute and Memory Resources

Challenge:

- Modern DNNs: millions to billions of parameters
- MCU Flash: typically <1 MB
- MCU RAM: typically <256 KB

Mathematical Constraint:

$$\text{Model\_size} \leq \text{Flash\_available}$$
$$\text{Activation\_size} \leq \text{RAM\_available}$$

Example:

- MobileNetV2: ~14 MB (uncompressed)
- Target MCU: 1 MB Flash
- **Compression ratio needed: 14×**

2. Power Consumption Constraints

Energy Budget:

- Coin cell battery: ~600 mAh at 3V = 6.5 kJ
- Target lifetime: 1 year = 31,536,000 seconds
- Available power: ~200 µW continuous

Energy per Inference:

$$E_{inference} = P_{compute} \times t_{inference}$$

For 100 inferences/day:

$$E_{daily} = 100 \times E_{inference}$$
$$E_{daily} \leq E_{budget\_daily}$$

Typical Power Budget:

Active: 10-100 mW | Sleep: <10 µW | Average: <1 mW

3. On-Device Learning Limitations

Challenges:

- Backpropagation requires storing activations
- Gradient computation memory-intensive
- Limited training data on device
- Energy cost of training 1000× higher than inference

Solutions:

✓ Transfer learning
(fine-tune pre-trained models)

✓ Federated learning
(collaborative without data sharing)

✓ Online learning
(incremental updates)

Challenge Interconnections:

All constraints are interdependent

OBJECTIVES TO ACHIEVE

OPTIMIZATION GOALS

Minimize number of the weights and activations in the networks.

Minimize the FLOP operation (floating point operation per second).

Minimize the time of the inference.

Minimize the energy requirements.

Minimize the memory complexity

THE ECOSYSTEM OF THE SOLUTION

TOOLS AND ECOSYSTEM

Algorithm

Compression techniques to achieve compact, efficient, and accurate models for edge deployment.

System

Compilers and IR that optimize operations for hardware efficiency. Graph optimization, kernel fusion, memory management.

Hardware

Domain-specific accelerators designed for efficient neural network inference at the edge.

Compression Algorithms

1. Quantization
Reduce precision (FP32→INT8) to minimize model size and accelerate inference.

2. Pruning
Remove redundant weights to create sparse networks with minimal accuracy loss.

3. Knowledge Distillation
Train small student model from large teacher to transfer learned knowledge.

4. Tensor Decomposition
Factorize weight tensors into low-rank components for parameter reduction.

System-Level Tools

TVM
Deep Learning Compiler

MLIR
Multi-Level IR

ONNX Runtime
Inference Optimizer

TFLite Micro
MCU Inference

Custom Hardware

Note: Frameworks like PyTorch/TensorFlow provide high-level APIs, while system tools optimize for hardware deployment.

TINYML HARDWARE PLATFORMS

HANDS-ON UNDERSTANDING

Popular Development Boards

Arduino Nano 33 BLE Sense

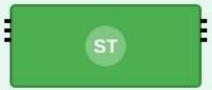
- Cortex-M4 @ 64 MHz
- 1 MB Flash, 256 KB RAM
- IMU, Mic, Temp, Humidity

Raspberry Pi Pico

- Dual Cortex-M0+ @ 133 MHz
- 2 MB Flash, 264 KB RAM
- Low cost ($4)

ESP32

- Dual-core Tensilica @ 240 MHz
- 4 MB Flash, 520 KB RAM
- WiFi + Bluetooth



STM32 Nucleo

- Cortex-M4 @ 80 MHz
- 1 MB Flash, 128 KB RAM
- Arduino-compatible

Sony Spresense

- 6-core Cortex-M4F @ 156 MHz
- 8 MB Flash, 1.5 MB RAM
- GPS, Audio processing

Nvidia Jetson Nano

- Quad-core ARM A57 @ 1.43 GHz
- 128-core Maxwell GPU
- Edge AI (not TinyML)

Selection Criteria: Power budget • Memory requirements • Sensor integration • Connectivity • Cost • Development ecosystem

NEURAL NETWORK COMPRESSION

TECHNICAL FOUNDATIONS

Challenge: Modern neural networks are too large for edge devices.
Solution: Compress models while maintaining accuracy.

Five Main Compression Techniques

1. Quantization

- Reduce precision: FP32 → INT8
- 4× smaller model, 4× faster
- Minimal accuracy loss (<1%)

2. Pruning

- Remove unimportant connections
- Structured vs. unstructured
- Up to 90% sparsity possible

3. Knowledge Distillation

- Train small from large model
- Transfer "soft" predictions
- Better than from scratch

5. Tensor Decomposition

- Low-rank factorization
- Tucker, CP decomposition
- Reduce parameter count

4. Efficient Architectures

- MobileNet, EfficientNet
- Depthwise separable conv.
- Efficient from start

QUANTIZATION DEEP DIVE

TECHNICAL FOUNDATIONS

What is Quantization?

Mapping high-precision values (FP32) to low-precision values (INT8, INT4)

Quantization Formula

$$q = \text{round}(r / S) + Z$$

S = scale, Z = zero-point, r = real value, q = quantized

Precision Comparison

FP32 (32-bit Floating Point)

INT8 (8-bit Integer)

Benefits & Trade-offs

✓ Benefits

- 4× reduction in model size
- 2-4× faster inference
- Lower memory bandwidth

⚠ Trade-offs

- Accuracy loss (typically <1%)
- Requires calibration data
- Hardware support needed

Types: Post-Training Quantization | Quantization-Aware Training | Dynamic Range

Popular Frameworks: TensorFlow Lite • PyTorch Mobile • ONNX Runtime • TFLite Micro

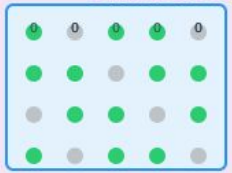
PRUNING DEEP DIVE

TECHNICAL FOUNDATIONS

Pruning removes redundant or less important weights/neurons from neural networks
to create sparse models while maintaining accuracy.

Two Main Types of Pruning

Unstructured Pruning



- Remove individual weights
- Irregular sparsity pattern
- Requires sparse matrix libraries

Structured Pruning

- Remove entire filters/channels
- Regular sparsity pattern
- Hardware-friendly (no special libs)

Typical Pruning Process

1. Train Model → 2. Identify Weights → 3. Prune Weights → 4. Fine-tune → 5. Repeat/Deploy

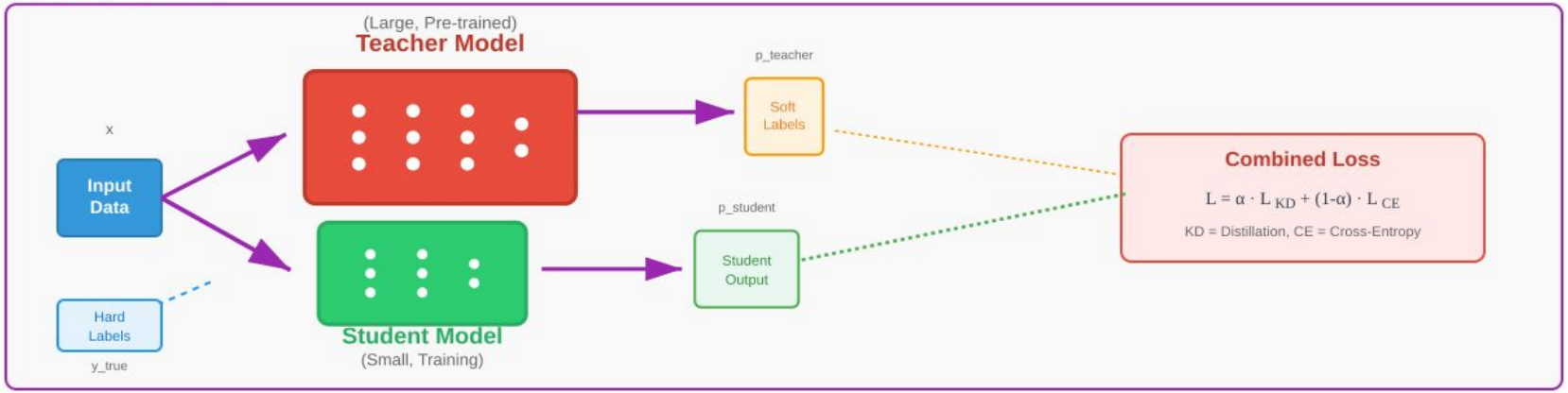
Common Criteria: Magnitude-based • Gradient-based • Second-order • Random

KNOWLEDGE DISTILLATION

TECHNICAL FOUNDATIONS

Knowledge Distillation transfers knowledge from a large "teacher" model
to a smaller "student" model, achieving better performance than training from scratch.

Architecture



Temperature Softmax

$$p_i = \exp(z_i/T) / \Sigma \exp(z_j/T)$$

T = Temperature (higher = softer probabilities)

Typical: T = 3-5 during training, T = 1 at inference

✓ Key Advantages

- Student learns from teacher's soft predictions
- Better generalization than training alone
- Can compress 10-100× with minimal accuracy loss

TENSOR DECOMPOSITION

TECHNICAL FOUNDATIONS

Tensor Decomposition decomposes high-dimensional weight tensors into products of smaller tensors, reducing parameters and computation.

Main Decomposition Methods

CP Decomposition

(CANDECOMP/PARAFAC)

- Sum of rank-1 tensors
- R = rank (hyperparameter)
- Compression: IJK → R(I+J+K)

Tucker Decomposition

- Core tensor + factor matrices
- More flexible than CP
- Compression: IJK → $R_1R_2R_3+IR_1+JR_2+KR_3$

SVD (Matrix Case)

Singular Value Decomposition

- Low-rank approximation
- Keep top r singular values
- Compression: mn → r(m+n+1)

Practical Example: Conv Layer Decomposition

Conv2D: [C_out × C_in × K × K] tensor → Decomposed into smaller filters

Example: 256×256×3×3 (589K params) → 256×64 + 64×256×3×3 (163K params) ≈ 72% reduction

MODEL DEPLOYMENT FRAMEWORKS

TINYML ECOSYSTEM

Popular Deployment Frameworks

TensorFlow Lite Micro

Features:

- C++17 runtime, no OS required
- Memory optimized (<20 KB)
- 90+ operators supported
- Op-level profiling built-in

Best for:
Production deployments, ARM Cortex-M

Edge Impulse Studio

Features:

- End-to-end ML platform
- Data collection & labeling
- AutoML capabilities
- Deploy to 100+ boards

Best for:
Rapid prototyping, audio/vision tasks

PyTorch Mobile / ONNX

Features:

- PyTorch ecosystem integration
- ONNX interoperability
- Mobile & edge optimized
- Growing MCU support

Best for:
Research projects, cross-platform

Typical Deployment Pipeline

1. Training
Python/TensorFlow
Model (.h5/.pt)

→

2. Optimization
Quantization
Pruning/Compression

→

3. Conversion
TFLite Converter
.tflite / .onnx

→

4. Code Gen
C Array
model_data.cc

→

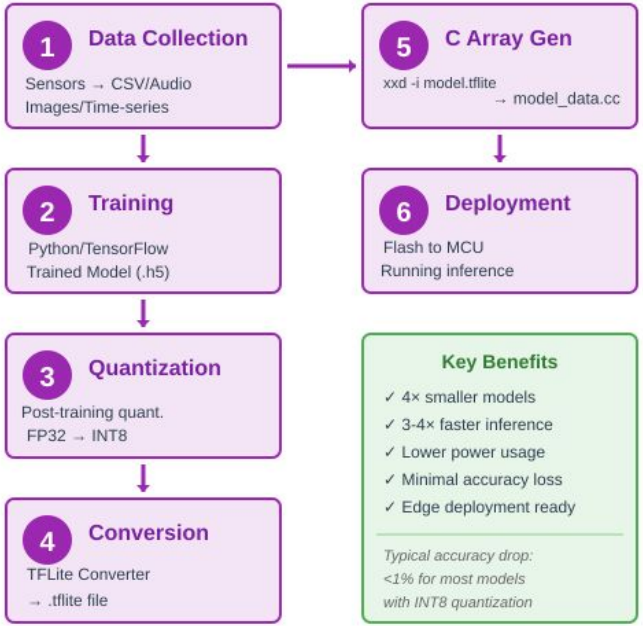
5. Deployment
Flash to MCU
Running inference

Key Considerations: Ease of use • Hardware support • Model format • Community • Documentation • Performance optimization

FROM MODEL TO MICROCONTROLLER

HANDS-ON UNDERSTANDING

Complete Workflow



Code Example: Conversion to TFLite

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

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

# Generate C array
$ xxd -i model.tflite > model_data.cc

# Result: C array ready for MCU
unsigned char model_tflite[] = {
0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, ...
};
unsigned int model_tflite_len = 10240;
```

Model size reduced: FP32 (40KB) → INT8 (10KB) → 4× compression
Inference speed: ~8ms on Cortex-M4 @ 64MHz
Memory footprint: 12KB RAM, 10KB Flash

Note: Always profile your model on target hardware to verify performance meets requirements

CASE STUDY: GESTURE RECOGNITION

REAL-WORLD APPLICATION

Problem Statement

Objective: Recognize hand gestures using accelerometer data on MCU

Gestures:

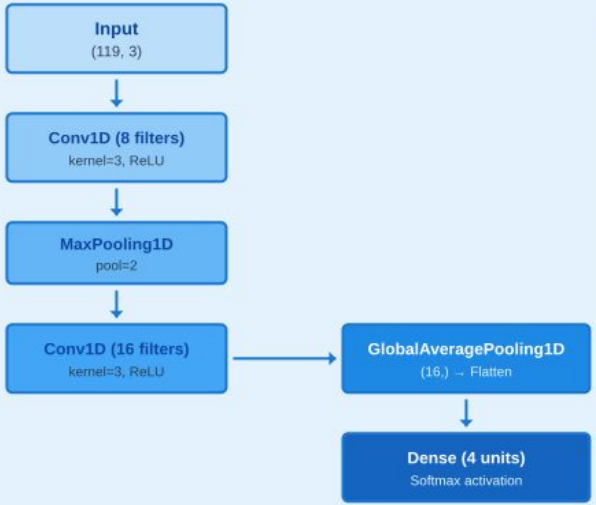
- Wave (side to side motion)
- Fist (punch motion)
- Open hand (spread fingers)
- Idle (no motion)

Dataset

Sensor: 3-axis accelerometer (LSM9DS1)
Sampling rate: 119 Hz
Window size: 1 second (119 samples)
Features: ax, ay, az × 119 = 357 inputs
Samples: 1000 per gesture class

Sample data visualization:

Model Architecture



Model Statistics

Total parameters: ~2,500
FP32 size: 10 KB
INT8 size: 2.5 KB

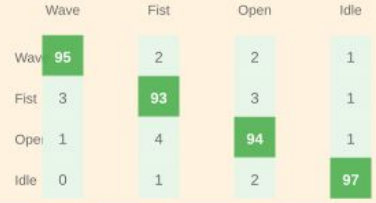
Trainable layers: 4
Input shape: (119, 3)
Output shape: (4,)

✓ Lightweight
✓ Fast inference
✓ MCU-friendly

Performance Metrics

Metric	Value
Accuracy	94.5%
Inference time	8 ms
Memory (RAM)	12 KB
Model size (Flash)	10 KB
Power consumption	45 mW

Confusion Matrix (simplified)



CODE SNIPPET: PYTHON → C++

FROM TRAINING TO DEPLOYMENT

Python Training Code

```python
import tensorflow as tf

# Define model
model = tf.keras.Sequential([
    tf.keras.layers.Conv1D(8,3, activation='relu', input_shape=(119,3)),
    tf.keras.layers.MaxPooling1D(2),
    tf.keras.layers.Conv1D(16,3, activation='relu'),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(4, activation='softmax')])

# Compile
model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])

# Train
model.fit(X_train, y_train,
epochs=50, validation_split=0.2)

# Convert to TFLite
converter = tf.lite.TFLiteConverter.from_keras_model(model)
```

C++ Inference Code

```cpp
// Include TFLite Micro
#include "tensorflow/lite/micro/
micro_interpreter.h"
#include "model_data.h"

// Allocate memory
constexpr int kTensorArenaSize =
20 * 1024;
uint8_t tensor_arena[kTensorArenaSize];

// Setup interpreter
tflite::MicroInterpreter interpreter(
  model, resolver,
  tensor_arena, kTensorArenaSize);

// Allocate tensors
interpreter.AllocateTensors();

// Get input tensor
TfLiteTensor* input =
interpreter.input(0);

// Copy sensor data
for (int i = 0; i < 357; i++) {
  input->data.int8[i] =
  quantize(sensor_data[i]);
}

// Run inference
interpreter.Invoke();

// Get output and process result
TfLiteTensor* output = interpreter.output(0);
int gesture = argmax(output->data.int8, 4);
```

Key Differences:

DATASETS AND BENCHMARKS

STANDARD EVALUATION TOOLS

Popular TinyML Datasets

Dataset	Task	Samples	Size	Input Type
Visual Wake Words	Object detection	115K images	~5 GB	96×96 RGB
Speech Commands	Keyword spotting	105K audio	~2 GB	1-sec clips
Google Commands	Voice recognition	65K commands	~1.4 GB	Audio
UCI HAR	Activity recognition	10K sequences	25 MB	IMU
Anomaly Detection	Industrial	1K normal + anomalies	100 MB	Sensor
Person Detection	Vision	5K images	500 MB	96×96 grayscale

MLPerf Tiny Benchmark

Standard benchmarks for TinyML performance:

Task	Model	Metric	Result
Keyword Spotting	DS-CNN	Accuracy	90.5%
Visual Wake Words	MobileNet	mAP	85.3%
Anomaly Detection	Autoencoder	AUC	0.92
Image Classification	ResNet-8	Accuracy	88.7%

Platforms tested:

Arduino Nano 33 • STM32F746 • ESP32 • NXP i.MX RT

Metrics Reported

ms — Latency (ms)
Inference time per sample

KB — Memory (KB)
RAM + Flash footprint

mJ — Energy (mJ)
Energy consumption per inference

% — Accuracy (%)
Model accuracy on test set

Standardized benchmarks enable fair comparison across platforms

OPTIMIZATION TECHNIQUES

ADVANCED COMPRESSION METHODS

Weight Clustering

Concept: Group similar weights into clusters

```
Original weights:
[0.23, 0.24, 0.87, 0.89, 1.02, 0.98]
Clustered (2 clusters):
[0.24, 0.24, 0.88, 0.88, 1.00, 1.00]
```

Benefits:

- ✓ Reduces unique values
- ✓ Improves compression
- ✓ Minimal accuracy loss

Compression: 2-4×

Typical accuracy drop:
< 0.5%

Mixed Precision

Concept: Different layers use different precision

```
Layer 1 (sensitive):        INT16
Layers 2-5:                 INT8
Layer 6 (sensitive):        INT16
Average: ~9.5 bits per weight
```

Benefits:

- ✓ Balance accuracy vs size
- ✓ Preserve critical layers
- ✓ Optimize memory usage

Sparse Matrix Compression

Structured Pruning Example:

```
Dense matrix (4×4):
[0.2 0.1 0.8 0.3]
[0.0 0.9 0.0 0.4]
[0.7 0.0 0.0 0.6]
[0.0 0.5 0.2 0.0]
Compressed (CSR format):
Values: [0.2, 0.1, 0.8, 0.3, 0.9, 0.4, ...]
Indices: [0, 1, 2, 3, 1, 3, 0, 3, ...]
Pointers: [0, 4, 6, 8, 10]
```

Size reduction: 40% → 62.5% savings

Neural Architecture Search

Automated model design for constraints:

```
Objective:        Maximize accuracy
Subject to:
  • Latency < 10 ms
  • Model size < 100 KB
  • RAM < 20 KB
```

Approaches:

- Reinforcement learning
- Evolutionary algorithms
- Gradient-based NAS

Result:

Custom architectures
optimized for hardware

PROFILING AND POWER MEASUREMENT

INTERACTIVE PERFORMANCE ANALYSIS

Profiling Tools

Energy Trace
(Texas Instruments)

Features:
- Real-time power monitoring
- Energy measurement

Resolution:
4 kHz sampling, nA

STM CubeMonitor
(STMicroelectronics)

Features:
- Variable monitoring
- Power analysis

Use cases:
Debug optimization

Edge Impulse Profiler

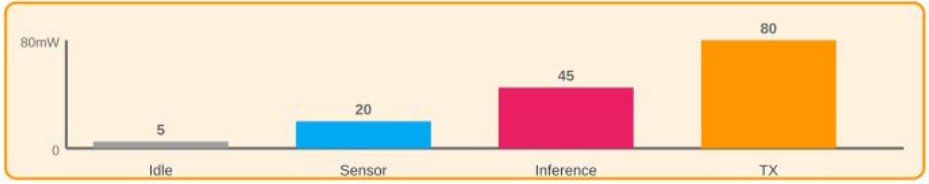
Features:
- On-device profiling
- Layer-wise breakdown

Metrics:
Time, RAM, Flash

Example Profiling Output

Layer	Time (ms)	RAM (KB)	Flash (KB)	Energy (µJ)
Conv1D_1	2.3	4.2	1.8	115
MaxPool	0.5	2.1	0.1	25
Conv1D_2	4.1	6.8	5.2	205
Dense	1.2	0.8	2.1	60
Total	**8.1**	**13.9**	**9.2**	**405**

Power Consumption Comparison



Energy Optimization Strategy

1. Reduce inference frequency
2. Use interrupt-driven sensing
3. Optimize model size
4. Sleep between operations

ML COMPILATION FRAMEWORKS

OPTIMIZATION AND CODE GENERATION

MLIR
Multi-Level IR

MLIR

Multi-Level Intermediate Representation

Features:

- Reusable compiler infrastructure
- Multiple abstraction levels
- Dialect system for extensibility
- Progressive lowering

Used by: TensorFlow, PyTorch, IREE

TVM

Apache TVM

Tensor Virtual Machine

Features:

- End-to-end compiler stack
- Auto-tuning for hardware
- Cross-platform deployment
- Supports ARM, x86, GPUs

Best for: Edge devices, mobile, IoT

XLA

XLA

Accelerated Linear Algebra

Features:

- JIT compilation
- Fusion optimization
- Memory optimization
- TensorFlow backend

Focus: Performance

IREE
ML Runtime

IREE

Intermediate Repr. Execution Engine

Features:

- Built on MLIR
- Ahead-of-time compilation
- Multi-backend support
- Efficient deployment

Target: Mobile, embedded, servers

Glow

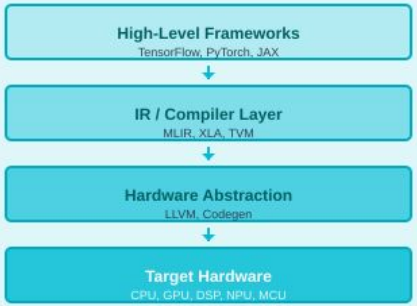
Graph Lowering Compiler

Features:

- Neural network compiler
- Hardware-agnostic
- Graph optimization
- PyTorch integration

Focus: PyTorch models

Compilation Stack



ML HARDWARE ACCELERATORS

SPECIALIZED PROCESSORS FOR EDGE AI

Neural Processing Units (NPUs)

Architecture:

- Dedicated neural network processing
- Parallel MAC (multiply-accumulate) units
- On-chip memory for weights
- Low precision (INT8/INT4) support

Examples:

- Google Edge TPU: 4 TOPS @ 2W
- Apple Neural Engine: 15.8 TOPS
- Qualcomm Hexagon NPU: 15 TOPS

Power Efficiency: 1-5 TOPS/Watt

Digital Signal Processors (DSPs)

Architecture:

- Optimized for signal processing
- Vector instructions (SIMD)
- Hardware loops, zero-overhead
- Fixed-point arithmetic

Examples:

- Cadence Tensilica HiFi: Audio AI
- TI C66x: Industrial, automotive
- ARM Ethos-U: Cortex-M ML extension

Best for: Audio, sensor fusion, control

MCU AI Accelerators

Architecture:

- Cortex-M with ML extensions
- Helium vector extension (M55/M85)
- Integrated MAC accelerators
- Ultra-low power modes

Examples:

- ARM Cortex-M55: 4× ML performance
- NXP i.MX RT: 1 GHz + NPU
- Renesas RA8: Helium + FPU

Power: 10-100 mW during inference

FPGA-based Accelerators

Architecture:

- Reconfigurable logic
- Custom datapath design
- Flexible precision (1-32 bits)
- Pipelined architecture

Examples:

- Lattice sensAI: Ultra-low power
- Intel Cyclone V: Mid-range
- Xilinx Zynq: ARM + FPGA

Best for: Custom algorithms, prototyping

ETHICS AND PRIVACY

LOCAL AI FOR DATA SOVEREIGNTY

Privacy Advantages

Data Stays Local

- ✓ No cloud transmission
- ✓ No third-party access
- ✓ GDPR/HIPAA compliant
- ✓ User control

Security Benefits

- ✓ Reduced attack surface
- ✓ No network vulnerabilities
- ✓ Tamper-evident hardware
- ✓ Encrypted storage

Best Practices

- ✓ Open-source models when possible
- ✓ Document data sources
- ✓ Test across diverse conditions
- ✓ Provide user control
- ✓ Regular audits and validation
- ✓ Clear consent mechanisms

Trade-offs

Aspect	Cloud AI	Edge AI	TinyML
Privacy	Low	Medium	High
Updateability	Easy	Moderate	Hard
Debugging	Easy	Moderate	Difficult

Ethical Considerations

1. Transparency

- How is data processed?
- What decisions are automated?
- Can users understand/audit?
- Explainability requirements

2. Fairness

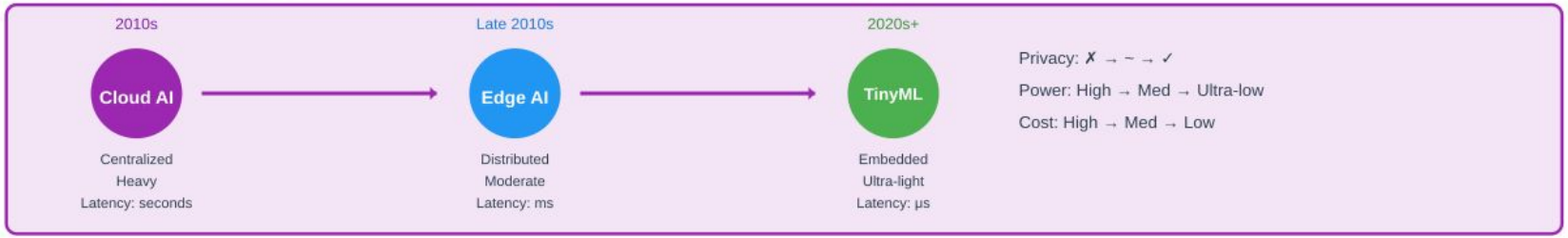
- Bias in training data
- Generalization across groups
- Equitable access to technology
- Inclusive design principles

3. Safety

- Reliability of edge models
- Failure modes and recovery
- Human override mechanisms
- Critical system safeguards

SUMMARY

THE EVOLUTION OF AI



Key Takeaways

1 Efficiency is Everything

- Power constraints drive innovation
- Memory optimization critical
- Compression techniques essential
- 100× reduction while maintaining accuracy
- <1 mW average power achievable

2 Edge AI Enables Real-Time

- Sub-10ms inference possible
- No network dependency
- Deterministic behavior
- Always-on capability
- Offline operation guaranteed

3 Sustainability Through TinyML

- Ultra-low power = battery/solar viable
- Distributed processing reduces load
- Democratizes AI access
- Years of battery life possible
- Reduced carbon footprint

4 Privacy by Design

- Data stays on device
- User sovereignty maintained
- Regulatory compliance built-in
- No cloud data breaches
- GDPR/HIPAA friendly

QUESTIONS?

Dr. Mohammed Alnemari

malnemar@uci.edu

Thank you for your attention!

Collaboration Welcome

Research partnerships
Industry projects

Key Resources

TinyML Foundation: tinyml.org
Edge Impulse: edgeimpulse.com

Academic Opportunities

Student supervision
Research collaboration