M8-4

Edge Machine Learning

Edge Machine Learning:
Use Cases & Motivation

Edge Machine Learning:

Use Cases & Motivation

What is Edge Machine Learning?

- Running ML inference, and increasingly training, directly on edge devices rather than in the cloud

The Edge Computing Spectrum:

- **Cloud**: Centralized data centers
- **Edge Cloud**: Regional computing facilities (CDN points of presence)
- **On-Premise Edge**: Local servers, gateways, base stations
- **Device Edge**: Smartphones, vehicles, cameras, IoT devices

Traditional Cloud-Centric ML

Traditional ML workflow:

- Collect data from edge devices
- Transfer to cloud
- Train models centrally
- Deploy models back to edge or keep in cloud

Limitations:

- High latency for real-time applications
- Bandwidth constraints for data transfer
- Privacy concerns with raw data transmiss
- Dependency on reliable connectivity
- Energy costs of continuous data transmis

Edge ML Paradigm Shift

Edge ML workflow:

- Process data locally where it's generated
- Run inference directly on device
- Only send results or model updates
- Potentially train or fine-tune locally

Evolution:

- **Phase 1**: Cloud-only ML
- **Phase 2**: Cloud training, edge inference
- Phase 3: Edge training and inference (emerging)

Key Driver #1: Latency

Why latency matters:

- Real-time applications: < 10ms often required
- **User experience:** 100ms threshold for "instant" feel
- **Safety-critical systems**: Timely decisions save lives

Cloud round-trip latency:

- Best case: 80-100ms
- Typical: 200-300ms
- With congestion: 500ms+
- With unreliable connectivity: seconds or failure

Edge ML latency:

- On-device: 10-50ms
- Local edge: 5-20ms
- No dependency on network conditions

Key Driver #1: Latency-Critical Applications

Autonomous vehicles:

- Pedestrian detection: ~10ms response time
- Lane keeping: ~20ms response time
- Emergency braking: ~50ms response time

AR/VR:

- Head tracking: <20ms to prevent motion sickness
- Hand tracking: <50ms for natural interaction
- Object recognition: <100ms for seamless experience

Industrial automation:

- Machine safety: <20ms
- Robot control: 1-10ms
- Quality inspection: <100ms

Key Driver #2: Privacy

Data privacy concerns:

- Regulatory constraints (GDPR, HIPAA, CCPA)
- Sensitive personal information
- Competitive business data

Surveillance and monitoring concerns

- Edge ML privacy benefits: Raw data never leaves the device
- Only processed results or model updates shared
- Local processing enables privacy by design
- Selective data sharing under user control

Key Driver #2: Privacy

Privacy-sensitive domains:

- Healthcare (patient monitoring, diagnostics)
- Smart home (voice assistants, cameras)
- Financial services (fraud detection)
- Industrial settings (proprietary processes)

Key Driver #3: Bandwidth

Bandwidth challenges:

- IoT scale: Billions of connected devices
- High-data sensors: Cameras, LiDAR, radar
- Limited network capacity, especially in remote areas
- Costly data transmission (cellular, satellite)

Raw data vs. processed insights:

- HD video: ~6 MB/second
- LiDAR: ~10-100 MB/second
- Processed results: Often < 1 KB/second

Key Driver #3: Bandwidth

Bandwidth savings example:

- Smart camera with object detection
- Raw video: 2 GB/hour
- Object metadata only: ~50 KB/hour
- 40,000× reduction in data transmission

Key Driver #4: Autonomy & Reliability

Connectivity challenges:

- Intermittent network availability
- Remote locations with limited connectivity
- Underground/underwater environments
- Disaster scenarios with infrastructure damage

Benefits of autonomous operation:

- Function during network outages
- Operate in connectivity-challenged environments
- Reduced dependency on cloud infrastructure
- Resilience during peak load or attacks

Key Driver #4: Autonomy & Reliability

Critical applications:

- Disaster response robots
- Remote medical devices
- Off-grid renewable energy systems
- Deep sea and space exploration

Key Driver #5: Energy Efficiency

Energy costs of cloud dependency:

- Continuous data transmission is power-intensive
- Cellular radios consume 3-5× more power than local processing
- Battery life severely impacted by constant cloud communication

Edge ML energy benefits:

- Specialized hardware (NPUs, TPUs) for efficient inference
- Selective communication only when necessary
- Sleep modes between local processing
- Renewable energy integration for edge nodes

Key Driver #5: Energy Efficiency

Example:

- Smartphone wake word detection
- Cloud approach: 3-4 hours battery life
- Edge approach: 1-2 days battery life

Application Domain: Autonomous Systems

Application Domain: Autonomous Systems

Types of autonomous systems:

- Self-driving vehicles (cars, drones, robots)
- Industrial robots and cobots
- Autonomous marine/underwater vehicles
- Space exploration rovers and satellites

Edge ML capabilities:

- Object detection and avoidance
- Simultaneous localization and mapping (SLAM)
- Behavioral prediction and planning S
- ensor fusion and environmental modeling

Industry examples:

- Tesla's Full Self-Driving computer
- Boston Dynamics' robots with onboard vision
- Skydio autonomous drones
- NASA Mars rovers

Application Domain: Smart Devices

Consumer edge devices:

- Smartphones and wearables
- Smart speakers and displays
- AR/VR headsets
- Smart appliances

Edge ML applications:

- Voice recognition and assistants
- Computational photography
- Health monitoring
- Personalized recommendations

Industry examples:

- Apple's Neural Engine for on-device ML
- Google Pixel Visual Core
- Qualcomm AI Engine
- Meta Quest headsets with on-device hand tracking

Application Domain: AR/VR

Augmented and Virtual Reality: ‘

- Head-mounted displays
- Smartphone AR
- Industrial AR headsets
- Mixed reality environments

Edge ML requirements:

- Low-latency tracking (< 20ms)
- Real-time scene understanding
- Gesture and hand tracking
- Spatial mapping and

Industry examples:

- Microsoft HoloLens
- Apple Vision Pro
- Magic Leap
- Snap AR Spectacles

Application Domain: IoT & Advanced Sensor Systems

Specialized Sensor Systems:

- **Thermal cameras**: Heat pattern detection, night vision
- **LiDAR systems**: 3D mapping, object detection
- **Sonar arrays**: Underwater monitoring, depth sensing
- **Camera networks**: Surveillance, traffic monitoring
- **Drones**: Aerial inspection, mapping, delivery
- **BMS (Battery Management Systems):** Power optimization, anomaly detection

Application Domain: IoT & Advanced Sensor Systems

Edge ML applications:

- Real-time anomaly detection
- Environmental classification
- Predictive maintenance
- Autonomous navigation
- Energy optimization
- Target identification and tracking

Industry examples:

- FLIR thermal cameras with on-device person detection
- Ouster LiDAR with edge processing for object classification
- Sonardyne underwater systems with embedded analytics
- DJI drones with onboard obstacle avoidance
- Tesla BMS with predictive failure analysis

Edge-Cloud Spectrum: Decision Factors

Workload placement considerations:

- Latency requirements: Real-time vs. batch processing
- Data sensitivity: Regulated vs. public data
- Bandwidth availability: Limited vs. abundant
- Power constraints: Battery-powered vs. grid-connected
- Computation needs: Simple inference vs. complex training

Hybrid approaches:

- Split computing (model partitioning)
- Hierarchical processing
- Adaptive offloading based on conditions

Edge-Cloud Spectrum: When to Use Edge ML

Edge-first approach when:

- Latency < 100ms required
- Privacy is paramount
- Bandwidth is limited or expensive
- Operation must continue without connectivity
- Battery life is critical

Cloud-first approach when:

- Complex models beyond edge capabilities
- Aggregated data analysis needed
- Historical pattern recognition required
- Massive parallelization needed (training)
- Updates must be instantly global

“Specialized Applications

Specialized Edge Devices: Sensors & Systems

Advanced Sensing Platforms:

Thermal imaging systems:

- Building inspection, wildlife monitoring
- Edge ML: Automated temperature threshold alerts, human/animal detection
- Example: FLIR cameras with onboard person detection and temperature screening

LiDAR systems:

- 3D mapping, autonomous navigation, forestry management
- Edge ML: Point cloud classification, object detection, distance estimation
- Example: Ouster intelligent LiDAR with built-in environmental classification

Specialized Edge Devices: Mobile & Autonomous

Specialized Edge Devices: Mobile & Autonomous

Drones:

- Infrastructure inspection, precision agriculture, search & rescue
- Edge ML: Obstacle avoidance, target identification, path planning
- Example: Skydio drones with on-device navigation AI

Camera systems:

- Security, retail analytics, traffic management
- Edge ML: Face detection, people counting, license plate recognition
- Example: Axis cameras with ARTPEC chips for on-device analytics

Specialized Edge Devices: Industrial & Energy

Industrial & Energy Systems:

Battery Management Systems (BMS):

- Electric vehicles, grid storage, UPS
- Edge ML: State-of-charge prediction, degradation monitoring, thermal management
- Example: Tesla BMS with predictive failure analysis and pack balancing

Industrial Controllers:

- Manufacturing, process control, quality assurance
- Edge ML: Anomaly detection, preventive maintenance, energy optimization
- Example: ABB intelligent controllers with embedded ML for condition monitoring

Summary

- Edge ML addresses fundamental limitations of cloud-centric approaches
- Key drivers: Latency, privacy, bandwidth, autonomy, and energy efficiency
- Wide application domains: Autonomous systems, smart devices, AR/VR, and IoT
- Decision framework helps determine optimal workload placement
- Hybrid approaches often provide the best of both worlds
- Technology advancing rapidly toward more capable edge processing

Edge ML Model Optimization Technique

Edge ML Model Optimization Technique

Why Model Optimization Matters

Edge deployment challenges:

- Limited computational resources
- Restricted memory footprint
- Power/thermal constraints
- Real-time requirements

Optimization objectives:

- Reduce model size (KB/MB)
- Lower computational complexity (FLOPs)
- Decrease memory usage (RAM)
- Maintain accuracy within acceptable

Impact of optimization:

- 5-20× smaller model size
- 2-10× faster inference
- 3-15× lower energy consumption
- Enables previously impossible deployments

Quantization Overview

Reducing numerical precision of model weights and/or activations

Standard precisions:

- FP32: 32-bit floating point (training standard)
- FP16: 16-bit floating point
- BFloat16: 16-bit brain floating point
- INT8: 8-bit integer quantization
- INT4/2/1: Ultra-low precision (emerging)

Benefits:

- Smaller model size (2-4× reduction)
- Faster computation (2-4× speedup)
- Lower power consumption (3-4× reduction)
- Better hardware utilization

Floating Point Formats: FP16 vs BFloat16

Standard FP16 (IEEE 754 Half-Precision):

- 1 sign bit, 5 exponent bits, 10 mantissa bits
- Range: ±65,504
- Precision: ~0.0001
- Good for: Computer graphics, narrow dynamic range tasks

BFloat16 (Brain Floating Point):

- 1 sign bit, 8 exponent bits, 7 mantissa bits
- Range: Same as FP32 ($\pm 3.4 \times 10^{38}$)
- Precision: ~0.01
- Good for: ML training, wide dynamic range tasks

Key advantages of BFloat16:

- Same dynamic range as FP32
- Better numerical stability during training
- Smoother transition from training to inference
- Supported by modern ML accelerators (TPUs, newer GPUs)
- Can often train directly in BFloat16

Post-Training Quantization (PTQ)

Approach:

- Train model in FP32
- Convert to lower precision after training
- Calibrate with representative data

Techniques:

- Dynamic range quantization: Activations quantized on-the-fly
- Full integer quantization: Both weights and activations quantized
- Float16 quantization: Weights and activations in FP16/BFloat16

Post-Training Quantization (PTQ)

Advantages:

- No retraining required
- Simple, fast process (minutes vs. days)
- Minimal code changes

Limitations:

- Accuracy drop (1-5%) for complex models
- Less optimal than training-aware methods
- Requires careful calibration

Quantization-Aware Training (QAT)

Approach:

- Simulate quantization effects during training
- Allows model to adapt to quantization noise
- Fine-tune or train from scratch

Implementation:

- Forward pass: Apply fake quantization
- Backward pass: Use straight-through estimator
- Weights updated in full precision
- Final model converted to actual low precision

Quantization-Aware Training (QAT)

Advantages:

- Minimal accuracy loss (often <1%)
- Better results than PTQ
- Handles challenging models/tasks

Limitations:

- Requires retraining
- Longer development time
- More complex implementation

Pruning Overview

Removing less-important parameters from neural networks

Types of pruning:

- Unstructured pruning: Individual weights set to zero
- Structured pruning: Entire channels/filters/neurons removed
- Semi-structured: Block or pattern-based sparsity

Benefits:

- Reduced model size (5-10× compression)
- Fewer computations (2-5× speedup*)
- Lower memory bandwidth requirements

Accuracy-sparsity trade-off:

- 30-50% sparsity: No accuracy loss
- 50-80% sparsity: Minimal loss (<1%)
- 80-95% sparsity: Moderate loss (1-5%)
- 95%+ sparsity: Significant degradation

Pruning: Unstructured vs. Structured

Unstructured Pruning:

- Individual weights pruned (set to zero)
- Higher theoretical compression rates
- Maintains network architecture
- Requires sparse format support
- Limited practical speedup without special hardware

Structured Pruning:

- Entire structural units removed (filters, channels)
- Lower theoretical compression rates
- Actually changes network architecture
- Compatible with standard deep learning libraries
- Immediate practical speedup

Pruning Techniques

Magnitude-based pruning:

- Remove smallest weights (by absolute value)
- Simple and effective approach
- Can be global or layer-wise

Iterative pruning and fine-tuning:

- Train model to convergence
- Prune weights according to criteria
- Fine-tune remaining weights
- Repeat steps 2-3 until target sparsity

Importance-based pruning:

- Score weights by activation, gradient, or Hessian
- Better accuracy preservation
- More computationally expensive

Dynamic sparse training:

- Prune and grow connections during training
- Maintains sparsity while exploring parameter space
- Often better final accuracy at high sparsity

Lottery Ticket Hypothesis

Large networks contain smaller "winning ticket" subnetworks

Process to find winning tickets:

- Train original network
- Prune low-magnitude weights
- Reset remaining weights to original initialization
- Retrain smaller network

Key findings:

- Winning tickets train faster
- Often achieve comparable or better accuracy
- Can be up to 90-99% smaller than original network
- Provides theoretical foundation for pruning
- Suggests importance of initialization and connectivity patterns

Practical applications:

- Network architecture search & Efficient transfer learning
- Compact model deployment

Knowledge Distillation

Train a smaller model (student) to mimic a larger model (teacher)

Basic approach:

- Train large teacher model to high accuracy
- Use teacher to generate soft targets (probabilities)
- Train smaller student model using both:
  - Ground truth labels (hard targets)
  - Teacher's predictions (soft targets)

Benefits:

- 5-1× model compression
- Preserves accuracy better than direct training
- Transfers "dark knowledge" from teacher
- Can combine with other optimization techniques

Distillation loss:

```
L = α * CrossEntropy(student, hard_labels) +
    (1-α) * KLDivergence(student/T, teacher/T)
```

where T is temperature (typically 2-10)

Advanced Distillation for Edge

Feature distillation:

- Match intermediate layer activations
- Better for very deep → shallow networks
- Enables deeper knowledge transfer

Self-distillation:

- Use the same architecture for teacher and student
- Student initialized with subset of teacher weights
- Useful for structured pruning and width reduction

Advanced Distillation for Edge

Ensemble distillation:

- Multiple teachers → single student
- Combined knowledge from multiple sources
- Higher accuracy than single-teacher approaches

Online distillation:

- Teacher and student trained simultaneously
- No separate teacher training phase
- Faster development cycle

TinyML Techniques

ML for microcontrollers (MCUs) and ultra-low-power devices

Target hardware:

- Memory: 32KB-512KB RAM
- Storage: <1MB flash
- Compute: 16-200MHz MCUs
- Power: milliwatts or microwatts

Core techniques:

- Extreme quantization: INT8/INT4/binary
- Operator fusion: Combine layers to reduce memory transfers
- Memory planning: Reuse buffers, minimize peak usage
- Depthwise separable convolutions: Factorize standard convolutions
- Efficient attention mechanisms: Linear attention variants
- Specialized tiny architectures: MicroNets

TinyML Deployment Example

Person detection on Arduino Nano 33 BLE Sense:

- MCU: ARM Cortex-M4 @64MHz
- Memory: 256KB RAM
- Storage: 1MB flash
- Power: 1.9mA active current

Optimized model:

- MobileNet-based with 250KB model size
- INT8 quantized
- Custom last-mile architecture
- Hand-tuned for target hardware

Performance:

- Inference time: ~250ms
- Power consumption: ~10mW
- Accuracy: ~93% on person detection
- 20+ hours on coin cell battery

ML Optimization Frameworks

TensorFlow Lite

Focus: Mobile and edge deployment of TensorFlow models

Key features:

- Post-training quantization (FP16, INT8)
- Quantization-aware training Pruning via
- TensorFlow Model Optimization Toolkit

Supported optimizations:

- Weight quantization (INT8, FP16)
- Operator fusion Layer optimization
- Hardware acceleration (GPU, DSP,

Deployment targets:

- Android (Java, C++ APIs)
- iOS (Swift, Objective-C)
- Linux devices Microcontrollers

ONNX Runtime

Focus: Cross-platform, cross-framework inference optimization

Key features:

- Framework-agnostic (convert from PyTorch, TensorFlow, etc.)
- Extensive graph optimizations
- Quantization tools (QDQ operators)
- Hardware acceleration (DirectML, CUDA, etc.)

Deployment targets:

- Windows, Linux, Mac
- Mobile platforms, Web (WASM)
- Edge devices (Raspberry Pi, etc.)

Supported optimizations:

- Constant folding
- Node fusion
- Layout optimization
- Redundancy elimination
- Memory planning

CoreML & TVM

CoreML:

- Apple's ML framework for iOS, macOS, watchOS, tvOS
- Optimized for Apple Silicon (CPU, GPU, Neural Engine)
- Support for quantization, pruning, and model compression
- Integration with on-device training APIs
- Automatic model scaling based on device capabilities

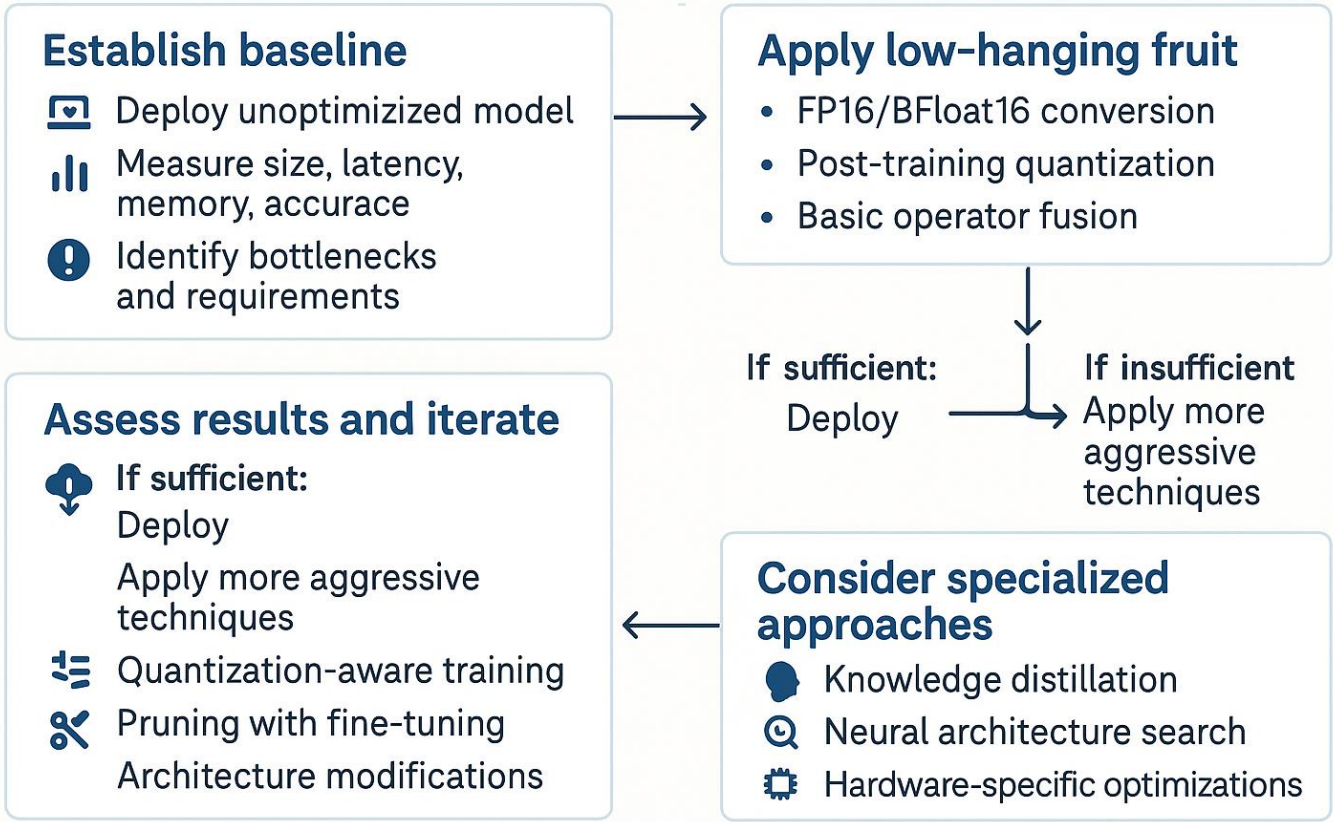
Apache TVM:

- Compiler-based approach to ML optimization
- Target-aware optimization
- Auto-tuning for specific hardware
- Support for custom hardware targets
- Quantization and operator fusion
- Deep learning compiler stack

Optimization Selection Guide

Technique	Size Reduction	Speed Improvement	Accuracy Impact	Development Effort
FP16/BFloat16	2×	1.5-2×	Minimal (<0.5%)	Low
INT8 PTQ	4×	2-3×	Low (1-2%)	Low
INT8 QAT	4×	2-3×	Very Low (<1%)	Medium
Pruning (50%)	2×	1.5×	Low (<1%)	Medium
Pruning (90%)	10×	2-4×	Medium (2-5%)	High
Distillation	5-10×	3-8×	Low-Medium (1-3%)	High
Specialized Arch	10-100×	5-20×	Varies	Very High

Practical Optimization Workflow



Key Takeaways

- Model optimization is essential for edge deployment - not optional
- Multiple complementary techniques: Quantization reduces precision and memory footprint
- Pruning removes unnecessary connections
- Distillation transfers knowledge to smaller models
- TinyML techniques enable MCU deployment
- Optimization is a spectrum: Start with easy wins (BFloat16, FP16, PTQ)
- Progress to more complex techniques as needed
- Combine approaches for greater effect

Key Takeaways

- Framework support is maturing: TensorFlow Lite, ONNX Runtime, CoreML, TVM
- Growing hardware acceleration ecosystem
- Automated optimization workflows emerging
- Always benchmark on target hardware: Theoretical gains ≠ real-world performance
- Hardware-specific optimizations matter
- Focus on end-to-end metrics