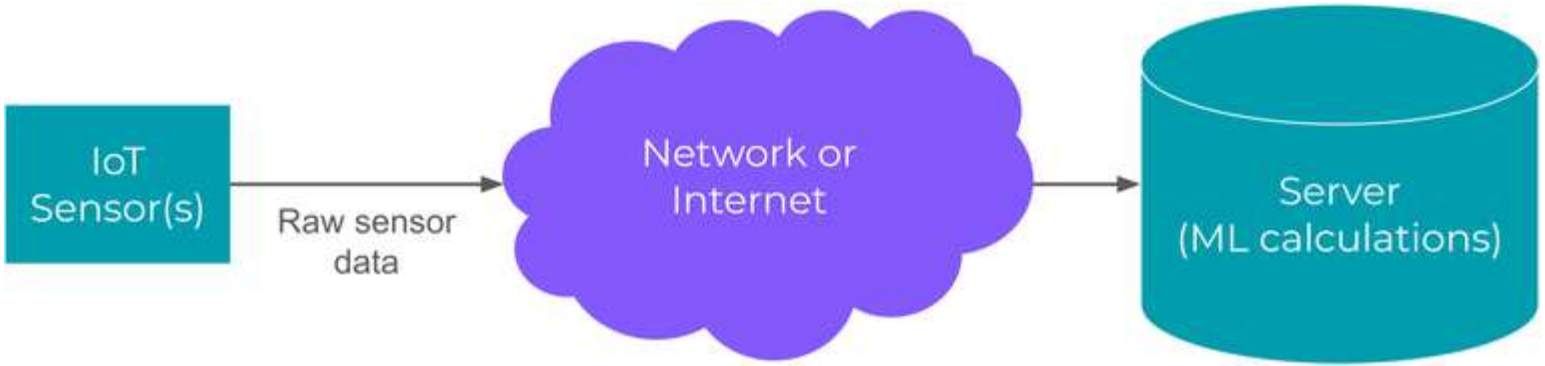
Deployment of AI Models on IoT Edge Devices

L08 (13th June, 2025)

Presented by:
Parul Kukrety

Recap



Recap: ML vs DL on Edge

Aspect	Machine Learning (ML)	Deep Learning (DL)
Model Complexity	Simple models (e.g., decision trees, SVMs)	Complex neural networks (e.g., CNNs, RNNs)
Resource Needs	Low power, memory (e.g., microcontrollers)	High power, needs GPUs/TPUs or optimization
Data Requirements	Works with smaller, structured datasets	Needs large datasets for training
Latency	Low (fast inference)	Higher (slower without optimization)
Accuracy	Moderate (good for simple tasks)	High (ideal for complex tasks)
Edge Use Cases	Smart thermostats, IoT anomaly detection	Facial recognition, autonomous vehicles
Optimization	Lightweight, minimal optimization needed	Requires quantization, compression (e.g., TensorFlow Lite)

Recap: IoT Analytics and Implementation

- Data Collection:
  - Deploy sensors (e.g., temperature, vibration) on IoT devices.
  - Ensure reliable connectivity (MQTT, CoAP).
- Data Preprocessing:
  - Clean noise, handle missing values, Normalize/scale sensor data for consistency.
- Analytics Processing:
  - Edge: Real-time anomaly detection (e.g., engine failure alerts).
  - Cloud: Batch analysis for predictive models (e.g., Random Forest).
- Model Deployment:
  - Use ML frameworks (TensorFlow, PyTorch) for predictive analytics.
  - Stream data via Apache Kafka for real-time insights.
- Actionable Outcomes:
  - Trigger alerts for maintenance (e.g., high vibration).
  - Optimize operations (e.g., adjust engine parameters).
- Monitoring & Updates:
  - Continuously retrain models with new data.
  - Validate predictions (e.g., precision, recall).

What are Edge Devices?

“Devices at the edge of the network”

- Edge devices are the hardware that collect and process data locally, then transmit to the cloud or other systems.
- use edge computing to collect, process, and analyse data locally.
- These devices are positioned at the edge of the network, close to where data is generated.
- This removes the need to send the data back to a centralised server or cloud for processing.
- examples?

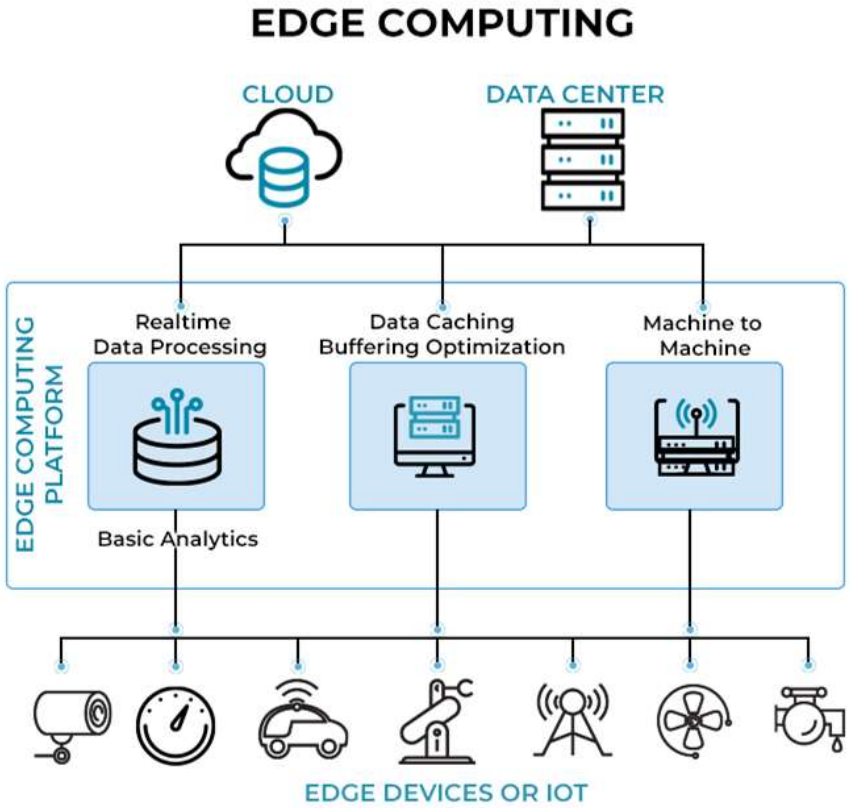
Edge Devices Examples

Traditional Edge Devices: Routers, switches, Firewalls, Multiplexers→ No self processing capabilities.

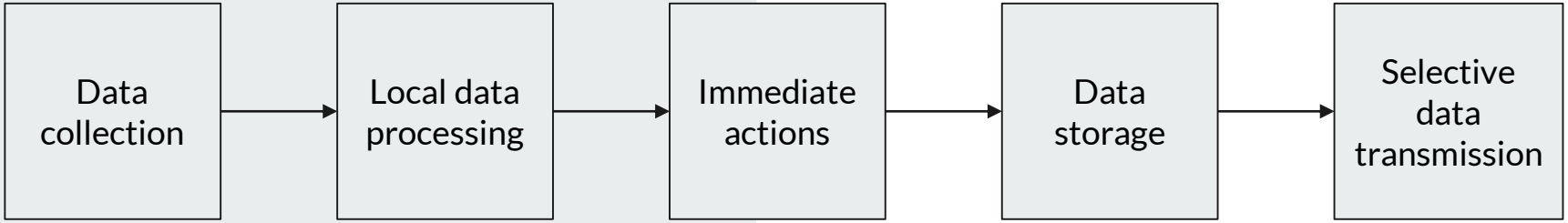
Intelligent Edge Devices: Smart- Sensors, cameras, wearables, IoT Gateways → Smart enough to perform computing tasks.

Working of Edge Devices

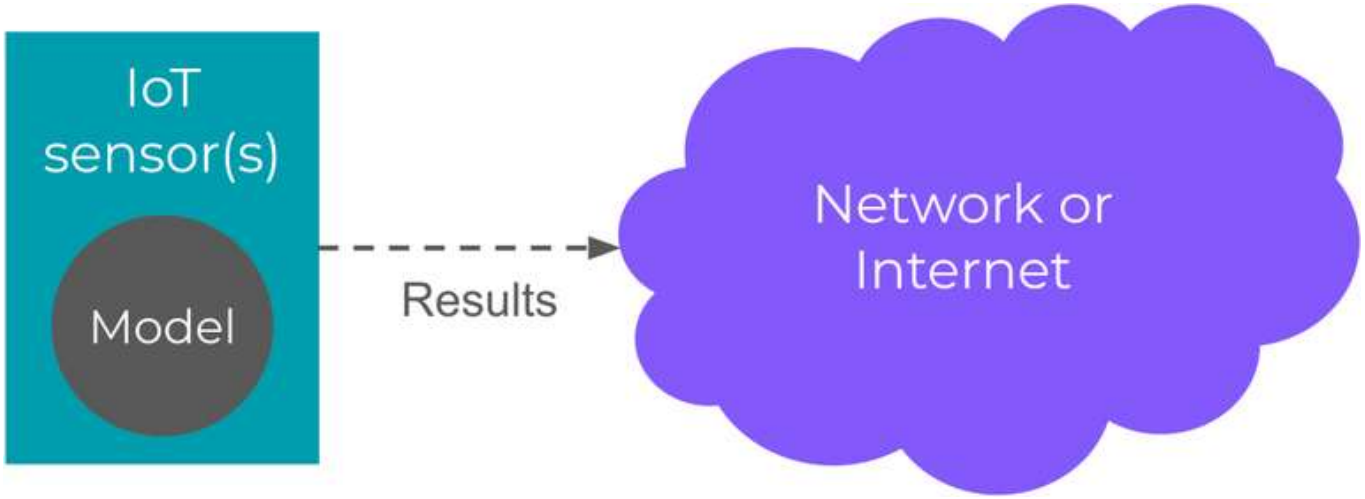
Works near the source of the generated data.



Working of Edge Devices

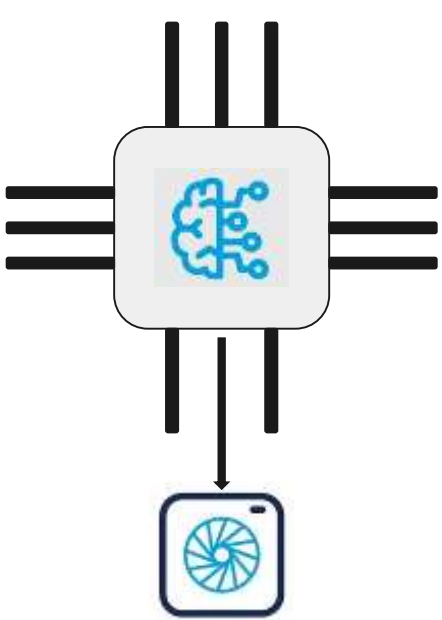


Edge ML



What is Edge ML?

- Running machine learning algorithms on computing devices at the periphery of a network.
- Make decisions and inferences as close as possible to the originating datas source.
- **“Edge AI”**

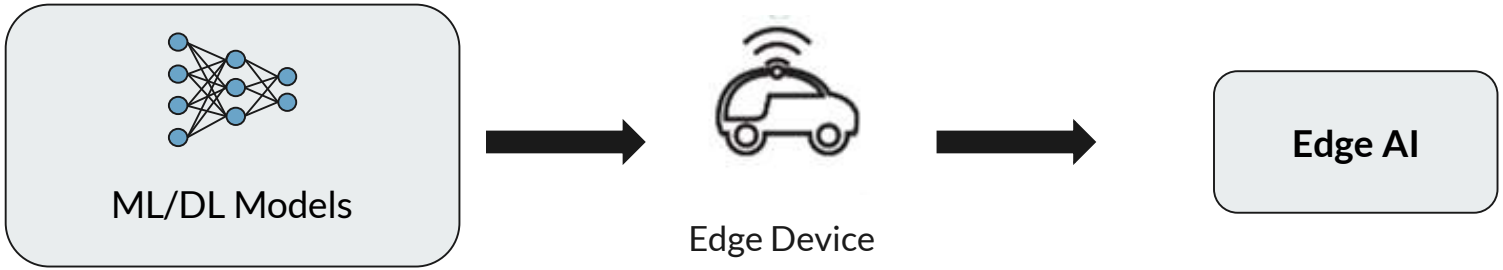


Reduced Latency

Privacy and Security

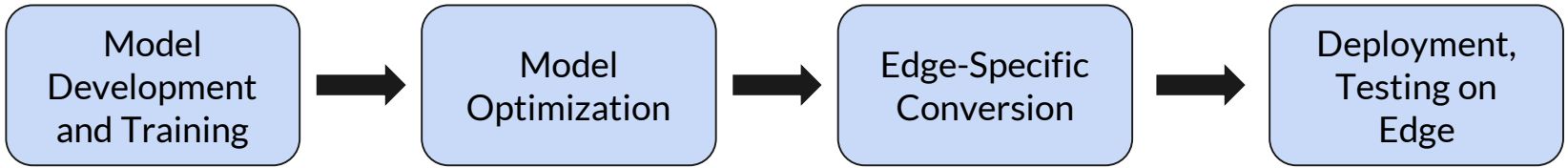
Power efficiency

What is model deployment?



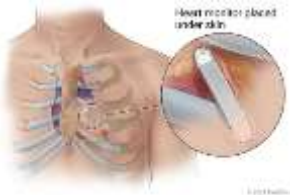
What is model deployment?

- Adapting ML/DL models to run on resource-constrained edge devices,
- Ensuring low latency, efficiency, and reliability.



Target Devices and Specifications

- Smartphones
- Smartwatches
- IoT sensors
- Embedded medical devices



Devices and Specifications

Device	Memory	Computation Power	Energy Consumption
Smartwatch	~0.5–1 GB RAM	100–500 MHz (ARM Cortex-A)	Low (~1–2W during use)
Drone	1–2 GB RAM	1–2 GHz (Quad-core ARM CPUs)	High (~20–50W during flight)
Smart Camera	256 MB – 2 GB RAM	500 MHz – 1.5 GHz (SoC/ASIC)	Medium (~5–10W)
Embedded Heart Monitor	~256 KB – 1 MB RAM	10–50 MHz (MCU)	Very Low (~0.1–0.5W)

Applications of model deployment on Edge

- Real-time object detection in cameras (Anomaly Detection using smart cameras)
- Smart HVAC systems (to identify number of people in a room)
- Security sensors (listening to sound signatures of glass breaking)
- Smart Agriculture (identifying irrigation requirements)
- Wildlife tracking
- Portable medical devices (Identifying diseases from the images)
- And many more.

Model Deployment Pipelines

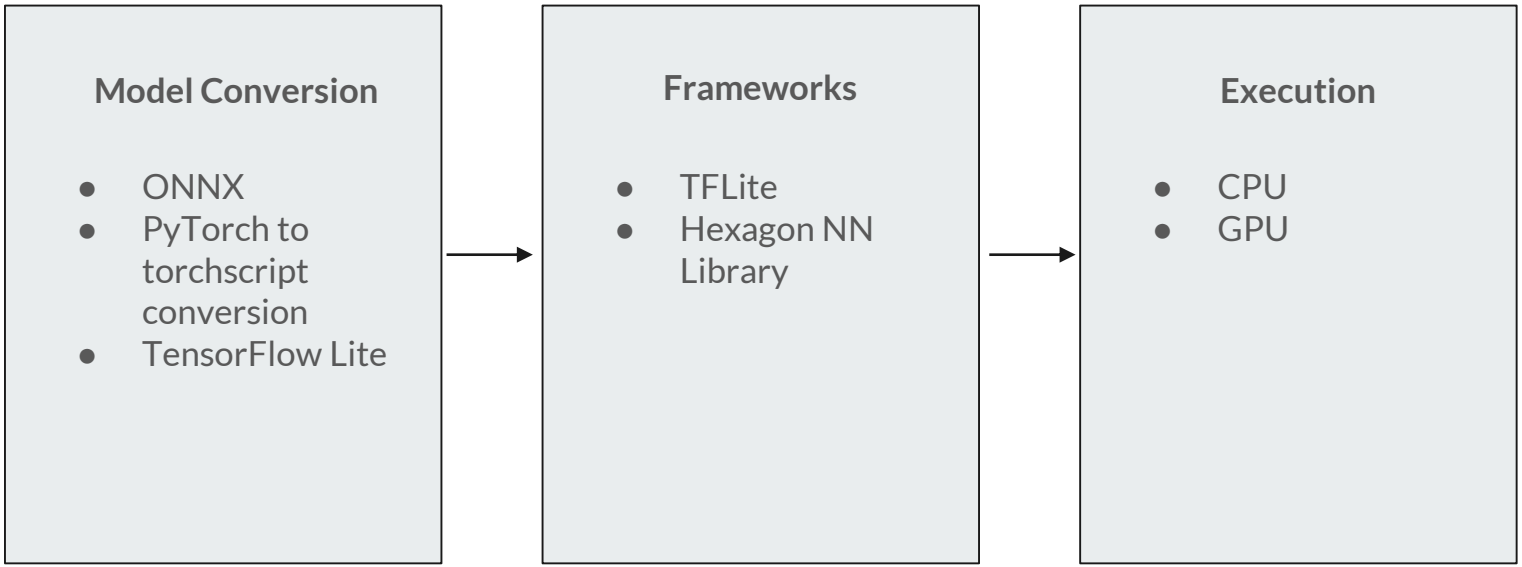
"A structured sequence of steps to move a trained model model from development to production."

Ensures model is optimized, packaged, and integrated into the target system (mobile application, cloud server, edge device).

Key components:

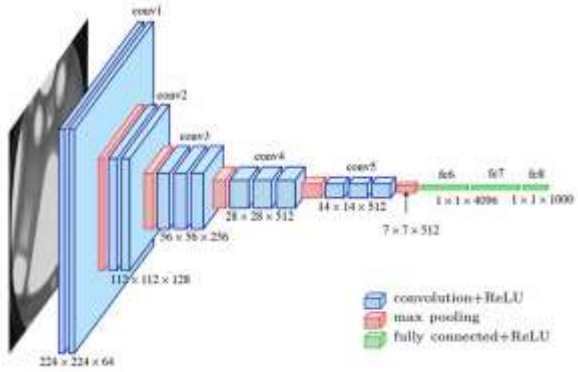
- Model export
- Optimization and compression
- Testing and validation
- Packaging
- Deployment
- Monitoring and maintenance.

Model Deployment



Problem Faced

- Large models in small world.
- They can be slow to respond.
- Using large models drains battery, which is not ideal for mobile or wearable gadgets.
- Not all devices are powerful enough to handle full-size AI models.



VGG16-138 Mil Params

Requires GPU

Trained on ImageNet

High Accuracy

High Cost

343.3 Mil FLOPS

Size > 500 MB

Smart Watch

RAM	**~0.5–1 GB RAM**
Power Consumption	**~1–2W during use**

Memory Overflow

Model too **big** for flash storage

Compute Timeout

Optimization and Compression

Model Development and Training:

- Develop ML (e.g., decision trees) or DL (e.g., CNNs) models in frameworks like TensorFlow, PyTorch, or scikit-learn.
- Train on powerful systems (cloud or desktop) with large datasets.
- ML: Simpler models, less data-intensive.
- DL: Complex models, requires large datasets and compute power.

Model Optimization:

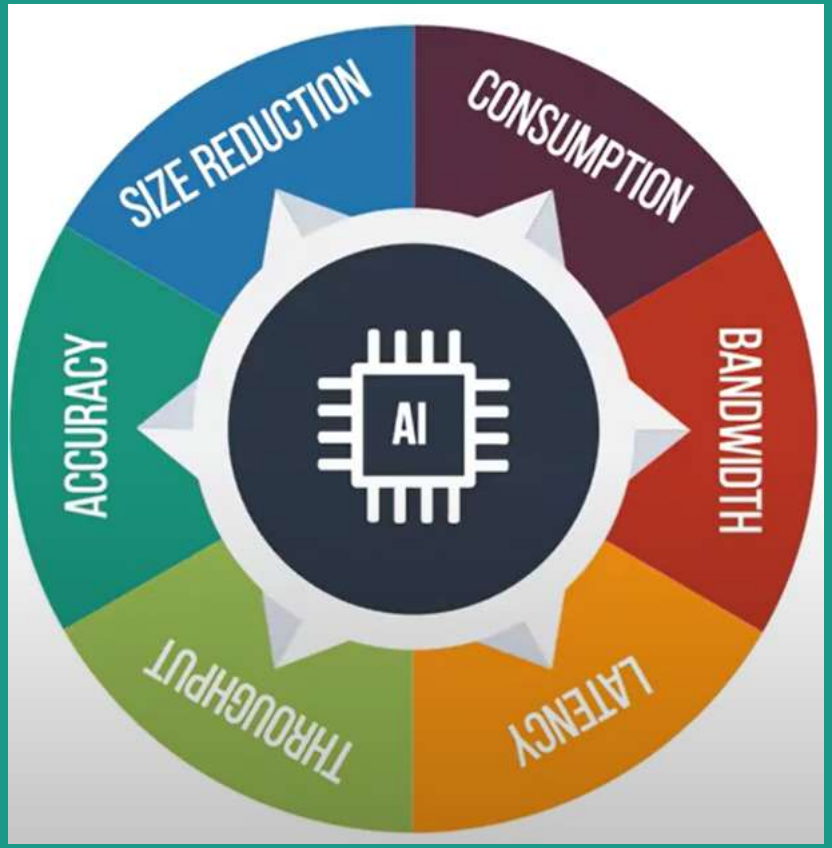
- **Quantization**: Reduce model precision (e.g., 32-bit to 8-bit) to lower memory and compute needs.
- **Pruning**: Remove redundant neurons/weights to shrink DL models.
- ML: Often naturally lightweight; minimal optimization needed.
- DL: Requires heavy optimization for edge (e.g., TensorFlow Lite).

Model Compression: The What and The Why?

We shrink models:

- Reduce size and computation
- Maintain accuracy (or trade off gracefully)
- Enable **real-time**, **on-device intelligence**
- Fewer and/or smaller parameters
- Uses less RAM during runtime
- Faster predictions from the model- lower energy consumption during inference.
- Reducing size typically improves latency and power efficiency.

Key Metrics



Some existing methods

Pruning

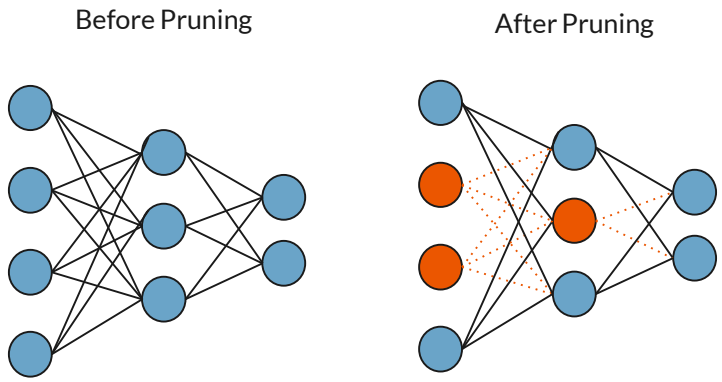
Quantization

Knowledge Distillation

Low-rank factorization

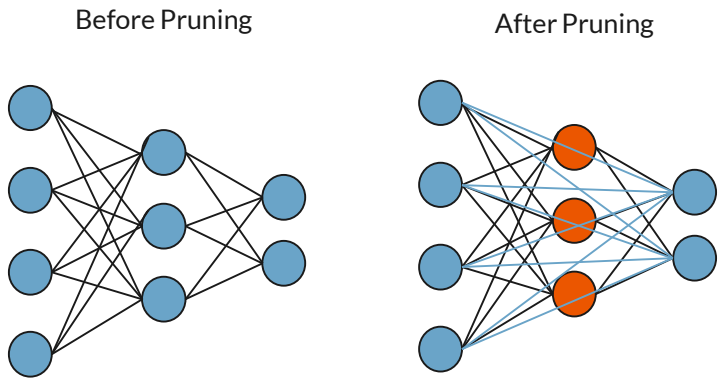
Pruning

" The process of eliminating unnecessary parameters or connections in a neural network, improving efficiency without compromising on performance "



- **Removes unimportant weights or neurons**
- **Retains essential model functionality**
- **Reduces model size and computation**
- **Speeds up inference on edge devices**

Structured Pruning

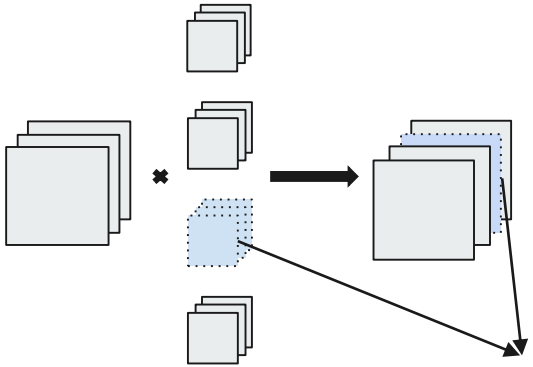


- **Removes entire filters, channels, or neurons**
- **Prunes at higher structural levels (e.g., layers, blocks)**
- **Simplifies model architecture**
- **Reduces computation by simplifying the model's weight graph**

Types of Structured Pruning

Neuron or Filter Pruning

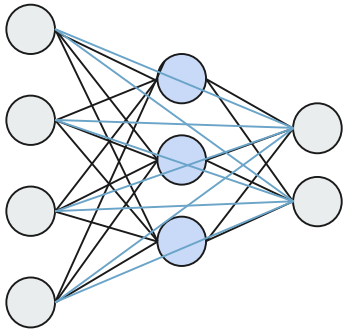
- Removes full filters (Conv) or neurons (Dense).



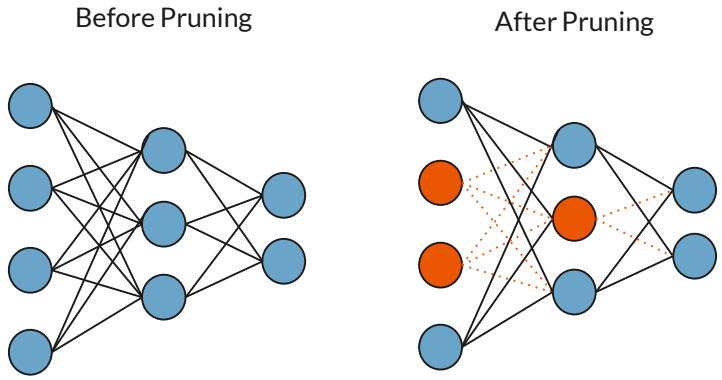
pruned

Layer Pruning

- Deletes redundant or shallow-impact layers.
- Drastically reduces depth and size.

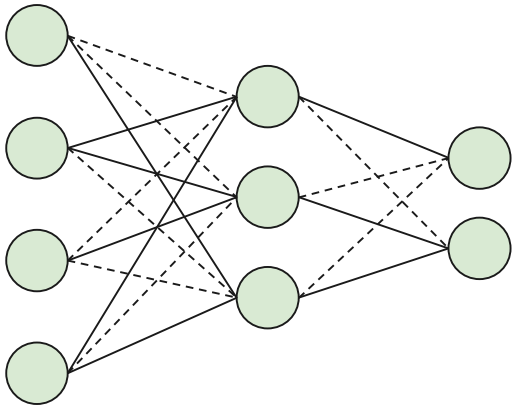


Unstructured Pruning



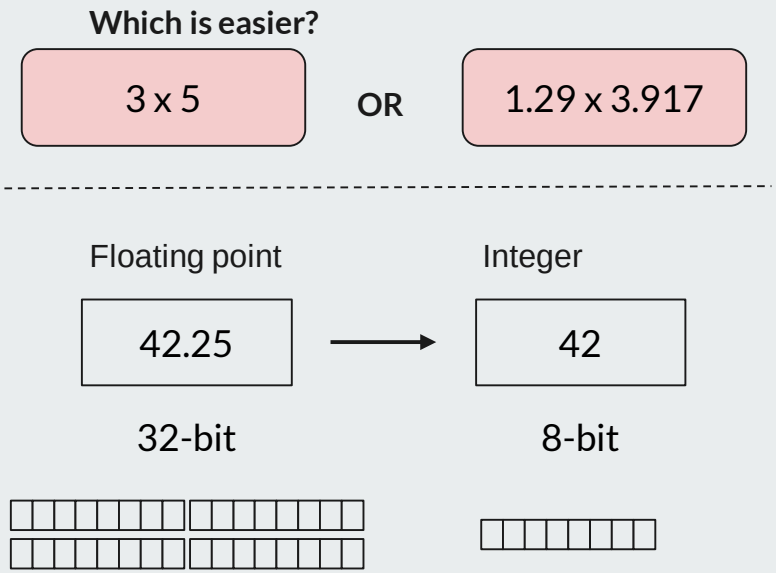
- **Removes individual weights based on simple thresholds**
- **Easy to implement with low computational cost**
- **Zeroes out weights without changing model structure**
- **Minimal impact on inference speed due to retained structure**
- **Can denoise weights and support lossless compression**

Weight Pruning
(Unstructured Pruning)



- Removes single, low-impact weights from weight matrices.
- Reduces overall parameter count.
- Keeps model structure unchanged.

Quantization



- Computers are slow at computing floating-point operations compared to integer operations.
- Reduces the precision of weights and activations.
- E.g. from 32-bit floating-point numbers to 8-bit or even lower.

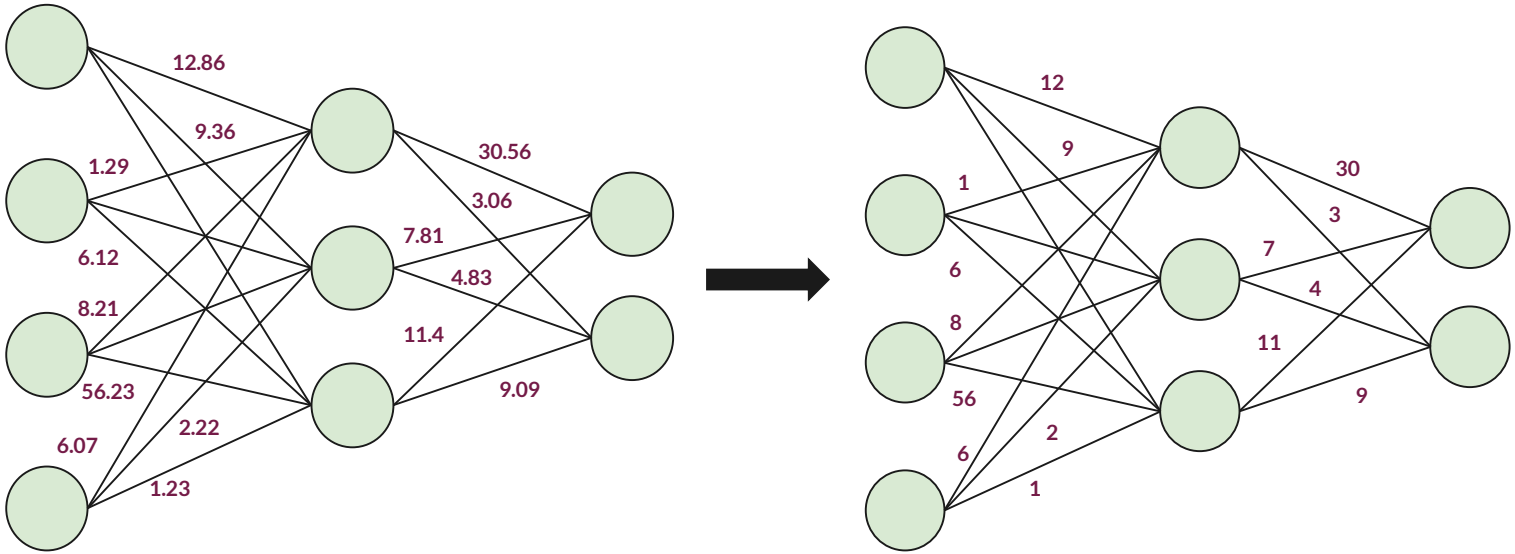
  FP32 → INT8 / INT4

- Saves a lot of inference computation cost.

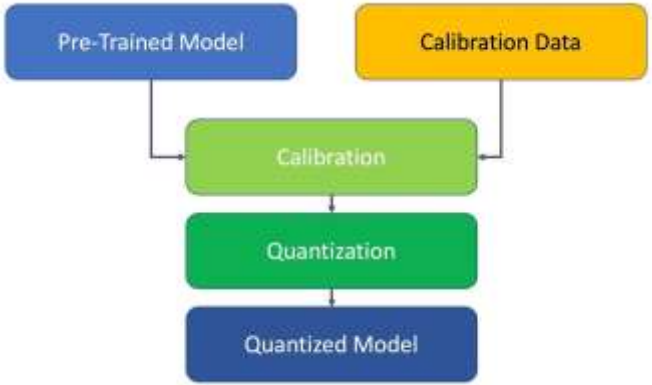
Operations

Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Mult	0.2
32b Mult	3.1
16b FP Mult	1.1
32b FP Mult	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

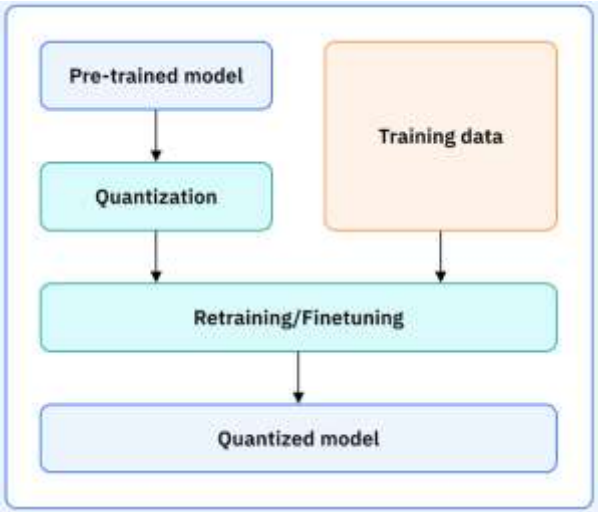
Quantization



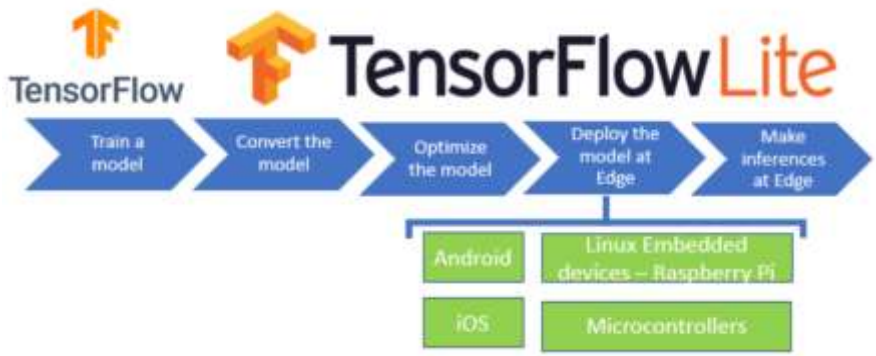
PTQ



QAT



Edge AI Frameworks



Edge AI Frameworks

Frameworks that are specialized software libraries that allow machine learning (ML) models to be run directly on local devices (like smartphones, embedded devices, wearables etc.), without needing to send data to cloud.

Key Features:

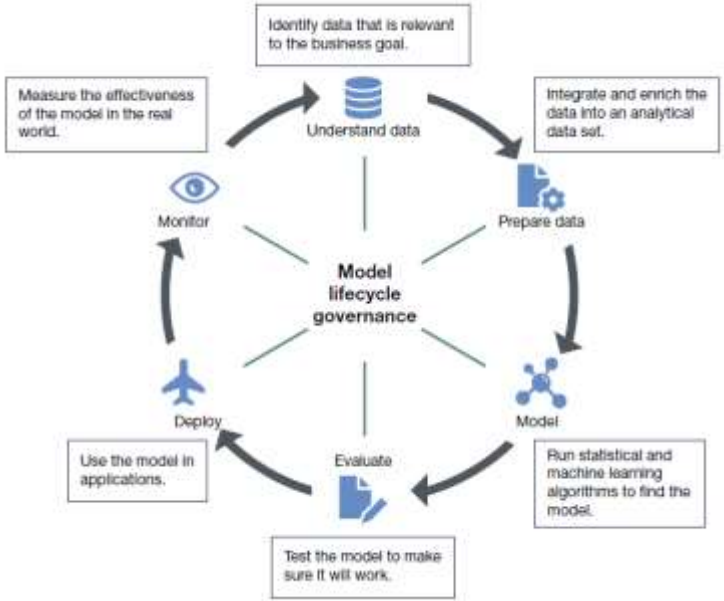
- Designed to be lightweight, efficient, hardware-aware.
- Enable real-time inference with low latency.
- Enhance privacy and security by processing data locally.
- Reduce network dependency and cloud costs.

Edge AI Frameworks

Framework	Platform	Highlights
TensorFlow Lite	Mobile phones, Raspberry Pi, MCU	Optimized for mobile and embedded devices
OpenVINO	Intel CPUs, VPUs, FPGAs	Optimized for Intel Hardware, high performance for computer vision.
Core ML	iPhones, iPads	Apple's native framework for on-device inference
PyTorch Mobile	Android, iOS, mobile devices	Lightweight version of PyTorch for mobile
MediaPipe	Phones, AR/VR headsets, IoT devices	Real-time ML solutions for vision and audio
Edge Impulse	Embedded devices, MCUs, Wearables	End-to-end tool for deploying ML on tiny devices

ModelOps

- Focus on end-to-end ML lifecycle.
- Data collection → experimentation → deployment
- Set of tools to deploy, manage and monitor different ML models.
- Deployment across environments (cloud, edge, on-premise).
- One-stop solution – Build, Run and Manage Models



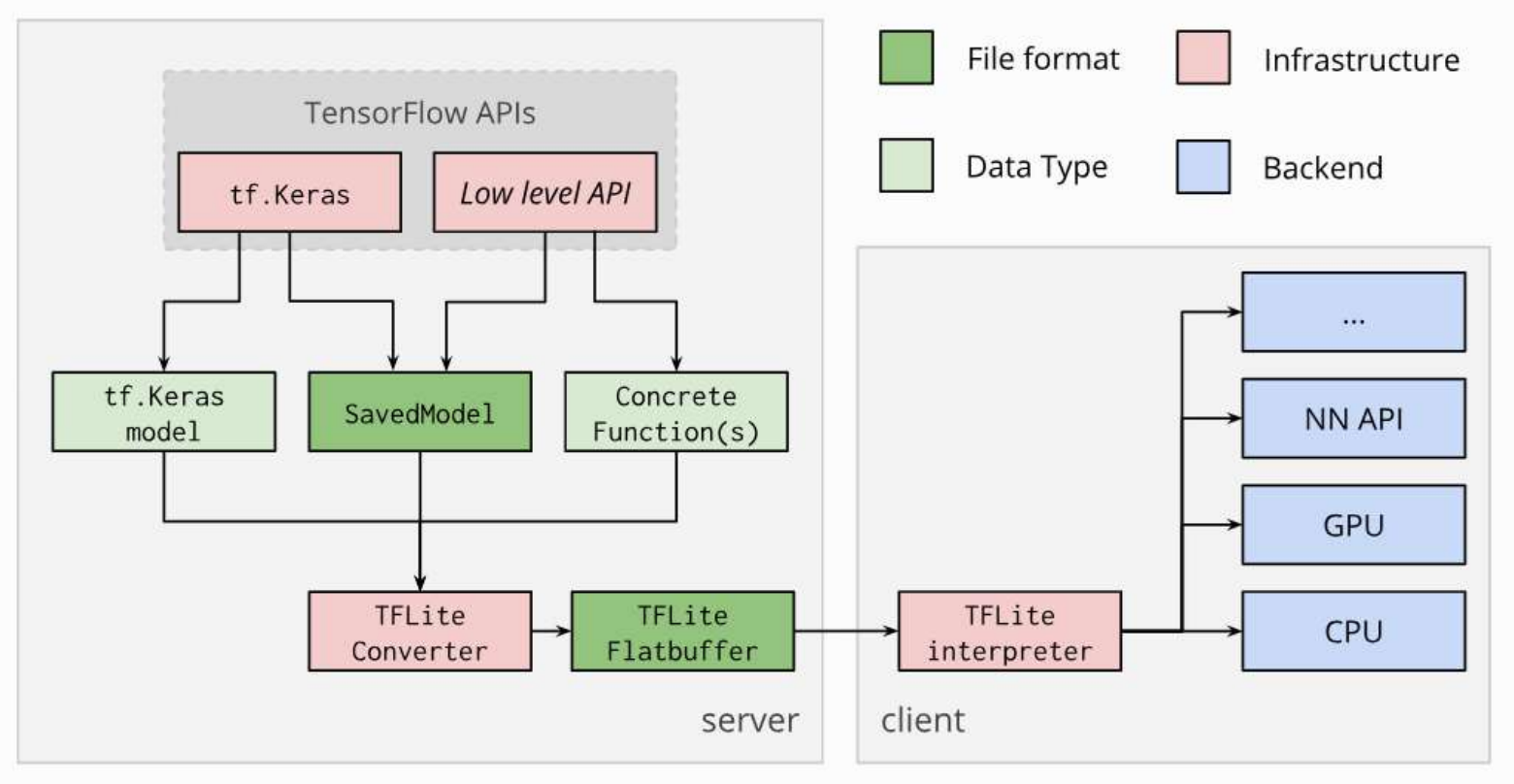
Tensorflow Lite

- TensorFlow Lite (TFLite) is an open-source framework by Google
- Used for deploying lightweight machine learning models on edge devices like mobiles, IoT, and microcontrollers.

Key Features-

- **Optimized for Edge**: Small footprint, low latency, and low power consumption.
- **Model Conversion**: Converts TensorFlow models to TFLite format for efficient execution.
- **Hardware Acceleration**: Supports GPU, DSP, and NNAPI for faster inference.
- **Cross-Platform**: Runs on Android, iOS, Linux, and microcontrollers.

Tensorflow Lite Workflow



Tensorflow Lite Workflow

1. **Train**: Build and train a model using TensorFlow.
2. **Convert**: Use TFLite Converter to optimize and compress the model.
3. **Deploy**: Run the model on edge devices with TFLite Interpreter.
4. **Optimize**: Apply quantization and pruning for efficiency.

OpenVINO

OpenVINO (Open Visual Inference and Neural network Optimization) is Intel's toolkit for deploying high-performance deep learning models on edge devices.

Key Features

- **Optimized Inference**: Accelerates deep learning on Intel hardware (CPU, GPU, VPU).
- **Model Optimizer**: Converts models from frameworks like TensorFlow, PyTorch to Intermediate Representation (IR).
- **Inference Engine**: Executes optimized models across diverse hardware.
- **Cross-Platform**: Supports Windows, Linux, and embedded systems.

CoreML

- Framework used by Apple.
- Integrate the ML models into the apps.
- Model can be retrained or fine-tuned on-device with edge data.
- Optimizes on-device performance by minimizing the memory footprint and power consumption.
- No need for network connection and privacy of data is maintained.

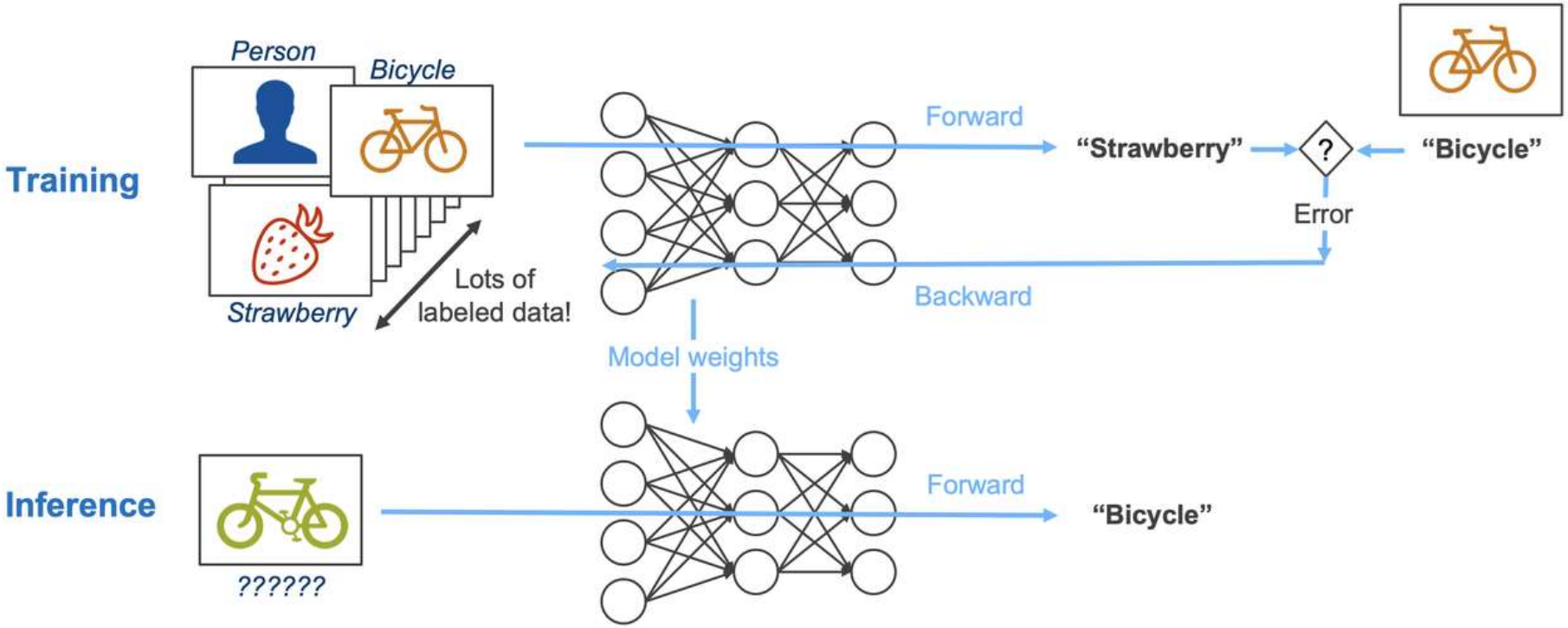
Inference of AI Models

What is inference?

“Process of using a trained ML/DL model to make predictions or decisions on new, unseen data.”

Key Aspects:

- Input: Sensor data (e.g., temperature, images) or user inputs.
- Processing: Model applies learned patterns to classify, predict, or detect.
- Output: Actionable results (e.g., anomaly detection, object recognition).



Inference of AI Models

Importance in IoT:

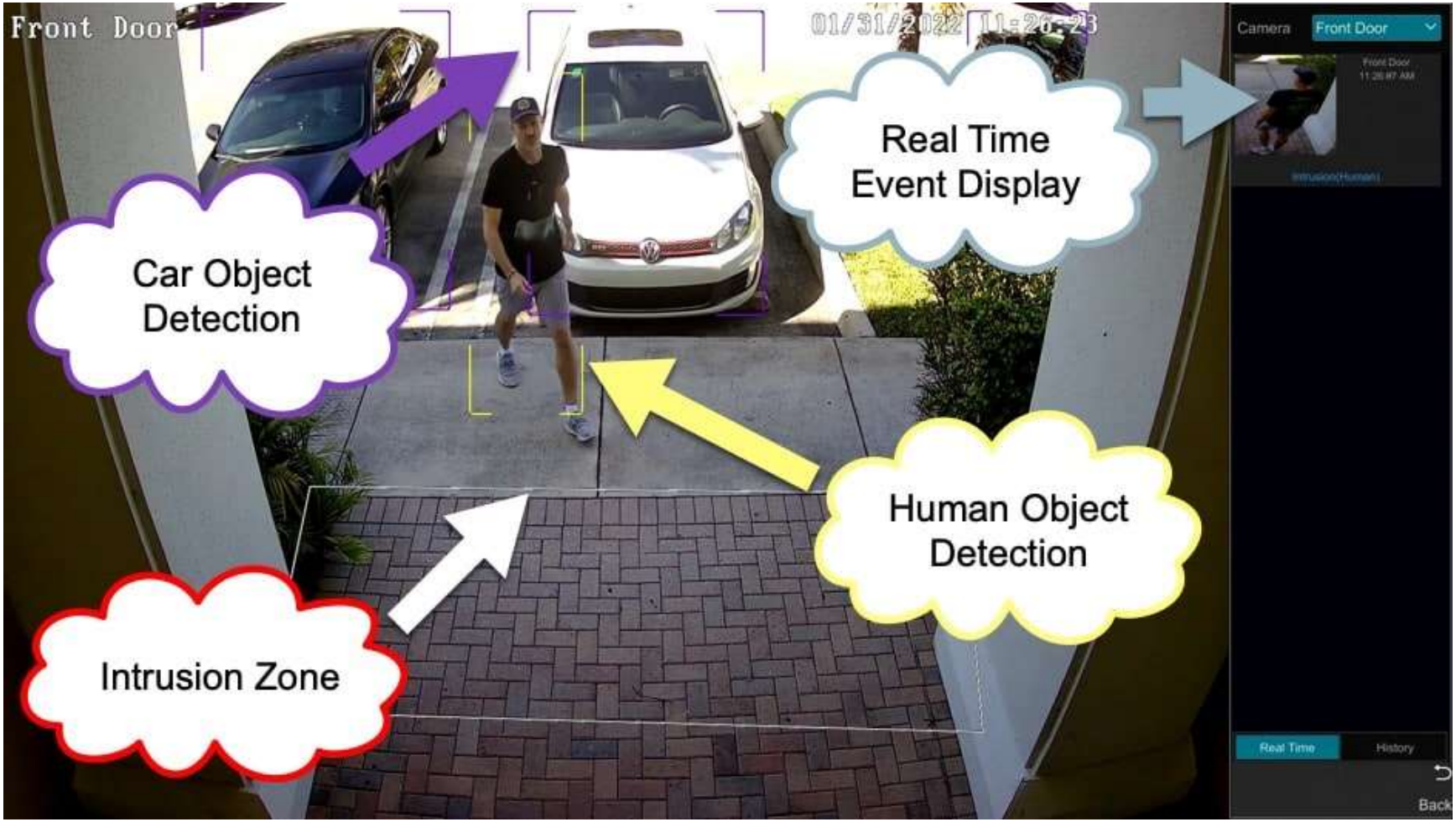
- Enables real-time decision-making (e.g., classifying temperature patterns as normal, hot, or cold).
- Balances latency, accuracy, and resource constraints.

Examples: ??

Edge Inference

Running ML/DL models directly on edge devices (e.g., smart cameras, IoT sensors) to process data locally.

Examples: ??



Edge Inference

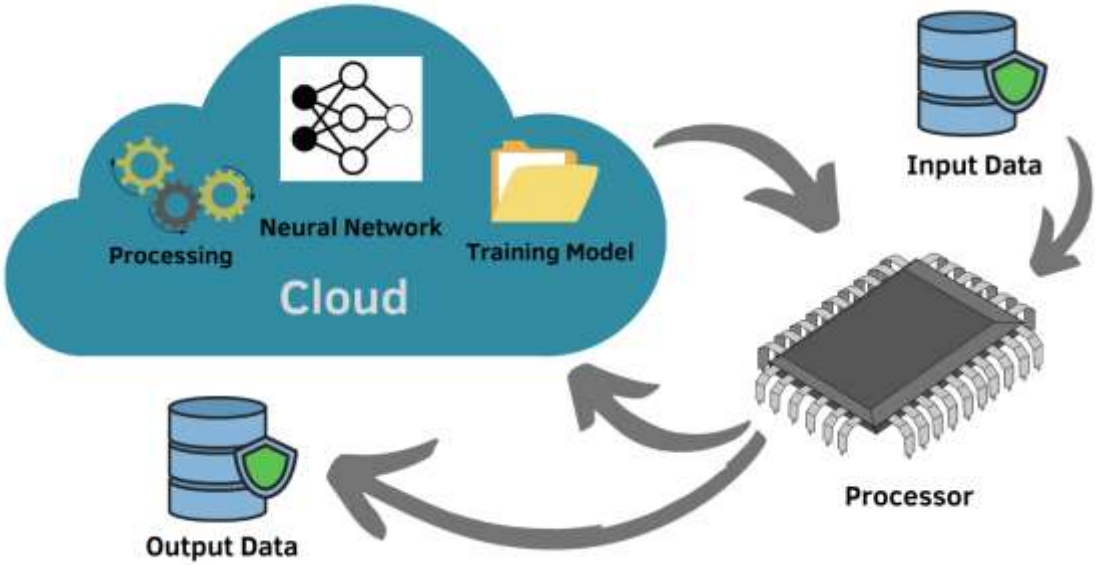
- Low latency: Immediate processing near data source (e.g., real-time anomaly detection).
- Privacy: Data stays on-device, reducing exposure (e.g., medical devices).
- Power efficiency: Optimized for low-energy hardware (e.g., 0.1–0.5W for embedded heart monitors).
- Offline capability: Operates without constant cloud connectivity.

Cloud Inference

Data (edge devices)

↓

Cloud Server (processing and Inference)



Cloud Inference

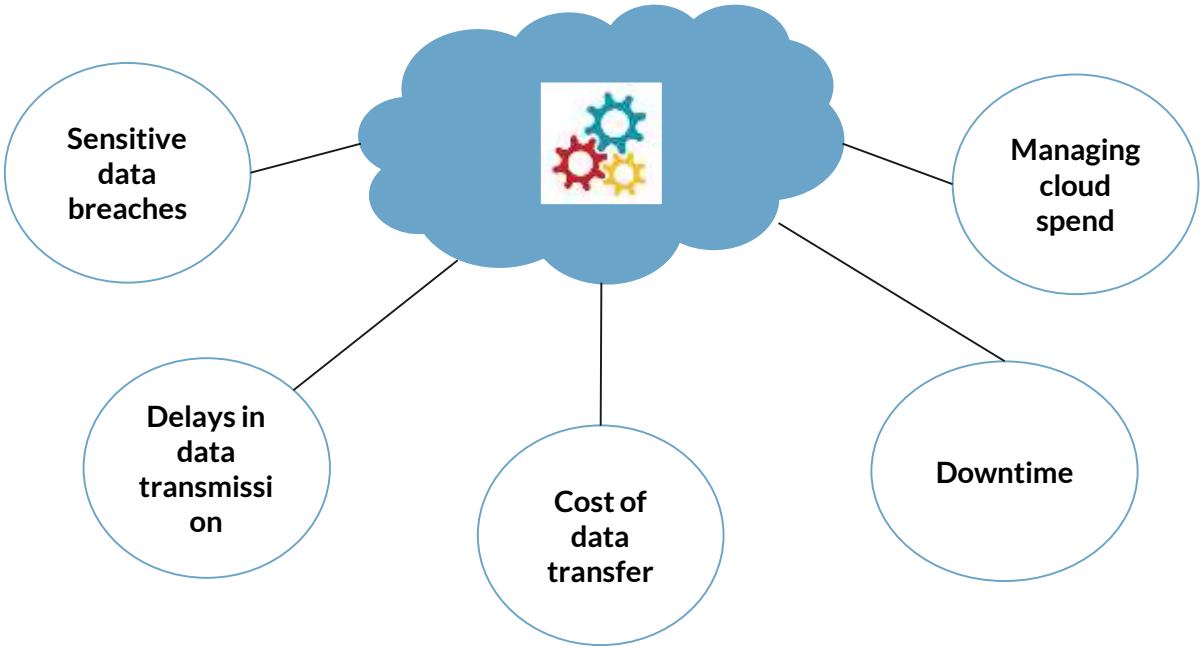
High compute power

Centralized updates

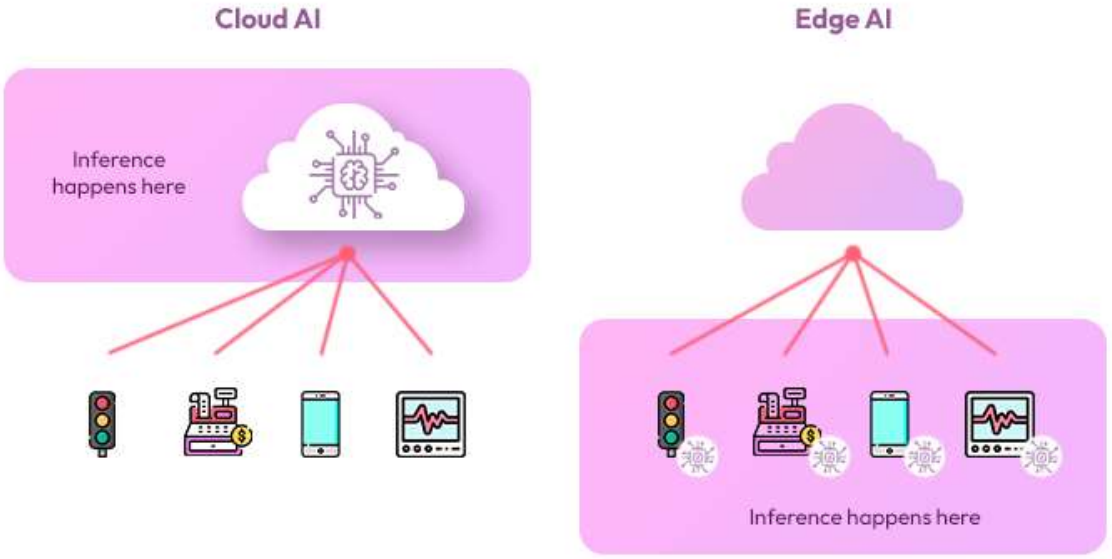
Scalability

Connectivity-dependent

Cloud inference challenges



Edge vs Cloud Inference



Simulate the deployment and inference of AI models

Perform inference of a trained model (e.g., classify temperature patterns) using **TensorFlow Lite** interpreter.

Model: Dummy model (a simple neural network)

Task: Temperature classification

Classes: 3(normal, hot, cold)

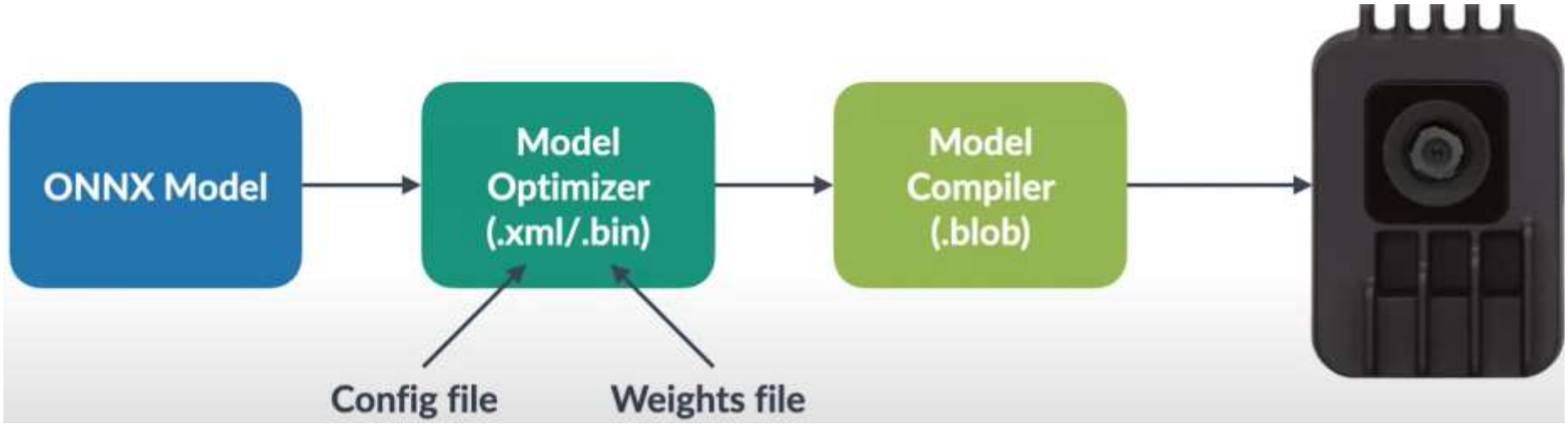
Dataset: Random generation of temperature values.

```python
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models

# Step 1: Create a simple TensorFlow model
def create_dummy_model():
    model = models.Sequential([
        layers.Input(shape=(3, 1)),  # 3 temperatures, 1 feature
        layers.Flatten(),
        layers.Dense(16, activation='relu'),
        layers.Dense(2, activation='softmax')  # 2 classes: normal, hot
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')
    return model

# Step 2: Generate dummy training data (for model initialization)
X_dummy = np.random.rand(100, 3, 1).astype(np.float32)  # 100 samples of 3 temperatures
y_dummy = np.random.randint(0, 2, 100)  # Random labels (0: normal, 1: hot)
model = create_dummy_model()
model.fit(X_dummy, y_dummy, epochs=1, verbose=0)  # Minimal training to initialize weights

# Step 3: Convert to TensorFlow Lite
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the TFLite model to a file
with open('dummy_temperature_classifier.tflite', 'wb') as f:
    f.write(tflite_model)
```

```python
# Step 4: Load the TFLite model for inference
interpreter = tf.lite.Interpreter(model_path='dummy_temperature_classifier.tflite')
interpreter.allocate_tensors()

# Step 5: Get input and output tensor details
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Step 6: Prepare sample input data (3 temperatures)
sample_input = np.array([[22.0, 23.5, 21.0]], dtype=np.float32).reshape(1, 3, 1)

# Step 7: Set input tensor and run inference
interpreter.set_tensor(input_details[0]['index'], sample_input)
interpreter.invoke()

# Step 8: Get and interpret output
output_data = interpreter.get_tensor(output_details[0]['index'])
class_names = ['normal', 'hot']
predicted_class = class_names[np.argmax(output_data[0])]
confidence = np.max(output_data[0])

print(f"Input temperatures: {sample_input.flatten()}")
print(f"Predicted class: {predicted_class}")
print(f"Confidence: {confidence:.4f}")
print(f"Raw output probabilities: {output_data[0]}")
```

```
Saved artifact at '/tmp/tmpqhot6vjt'. The following endpoints are available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): TensorSpec(shape=(None, 3, 1), dtype=tf.float32, name='keras_tensor')
Output Type:
  TensorSpec(shape=(None, 2), dtype=tf.float32, name=None)
Captures:
  138914682923984: TensorSpec(shape=(), dtype=tf.resource, name=None)
  138914682927248: TensorSpec(shape=(), dtype=tf.resource, name=None)
  138914682928976: TensorSpec(shape=(), dtype=tf.resource, name=None)
  138914682928400: TensorSpec(shape=(), dtype=tf.resource, name=None)
Input temperatures: [22.  23.5 21. ]
Predicted class: normal
Confidence: 0.9948
Raw output probabilities: [0.99478877 0.00521122]
```

ONNX (Model format)



Thankyou