

Exercise Session

Distributed Edge AI (and TinyML) Systems

Roberto Morabito
Assistant Professor @ EURECOM
<https://www.linkedin.com/in/robertomorabito>

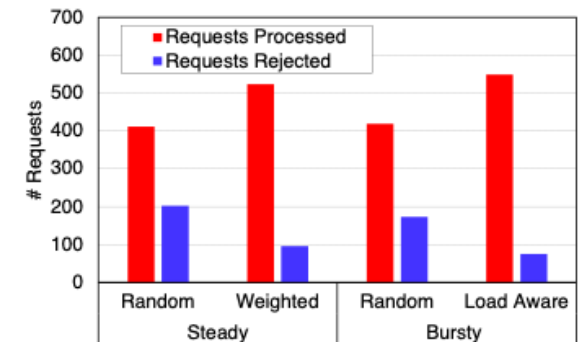
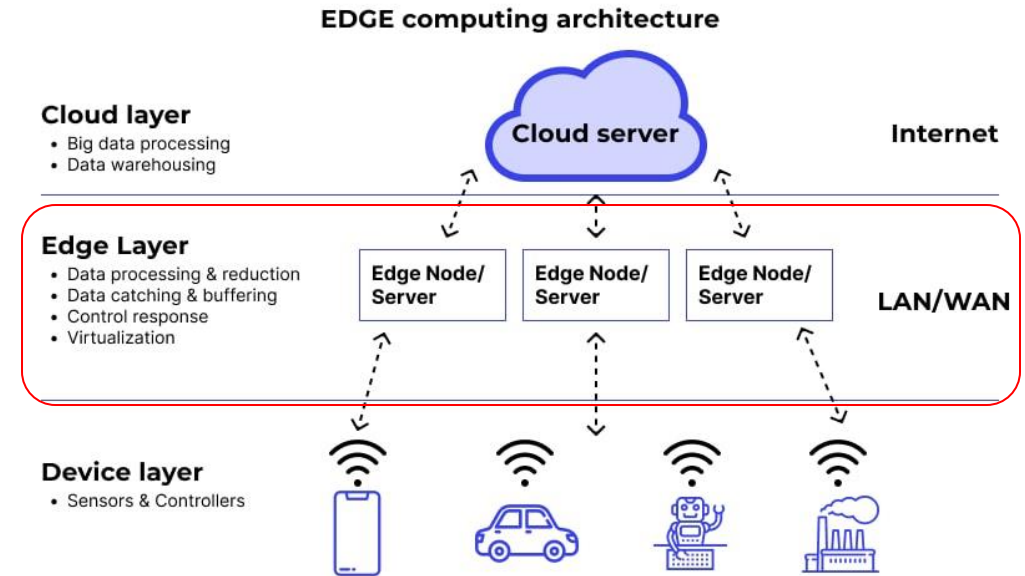
Recap From the Morning

Edge AI **challenges**:

- **Performance vs Latency**
- **Energy vs Accuracy**
- **Cost vs Autonomy**

The Edge–Cloud continuum: Device → Edge → Cloud.

Generative AI at the edge: why it matters



Goal of The Session



Explore TinyML in practice (training + quantization).



Experiment with computing continuum offloading trade-offs.






Work in groups to extend the simulator with new research ideas.



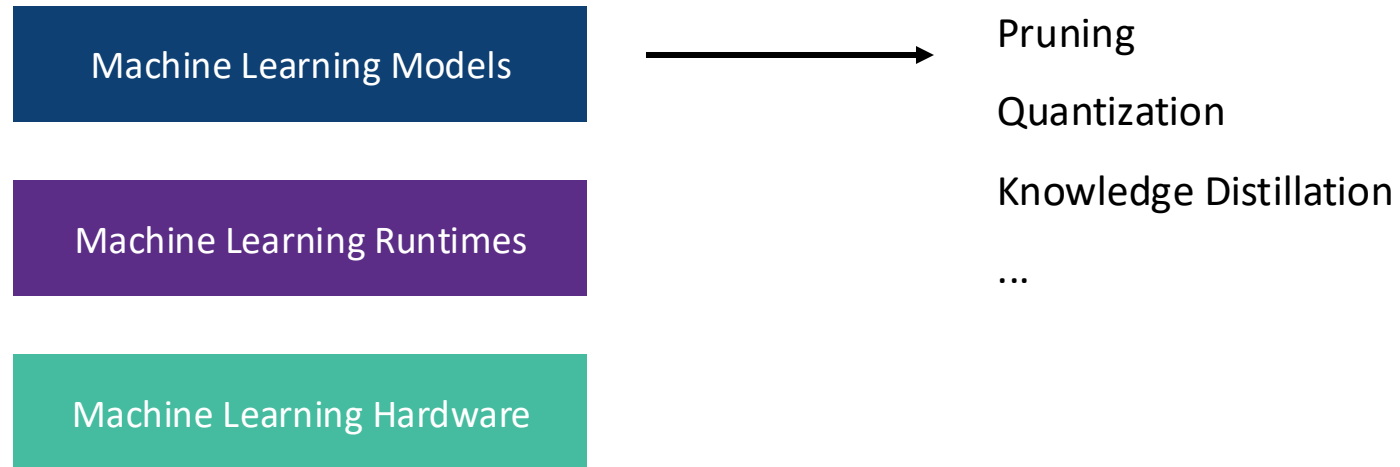
Instructions: <https://github.com/robertmora/distributed-edge-ai-lab>

TinyML Primer

	Microprocessor	>	Microcontroller
Platform	 		
Compute	1GHz–4GHz	~10X	1MHz–400MHz
Memory	512MB–64GB	~10000X	2KB–512KB
Storage	64GB–4TB	~100000X	32KB–2MB
Power	30W–100W	~1000X	150μW–23.5mW

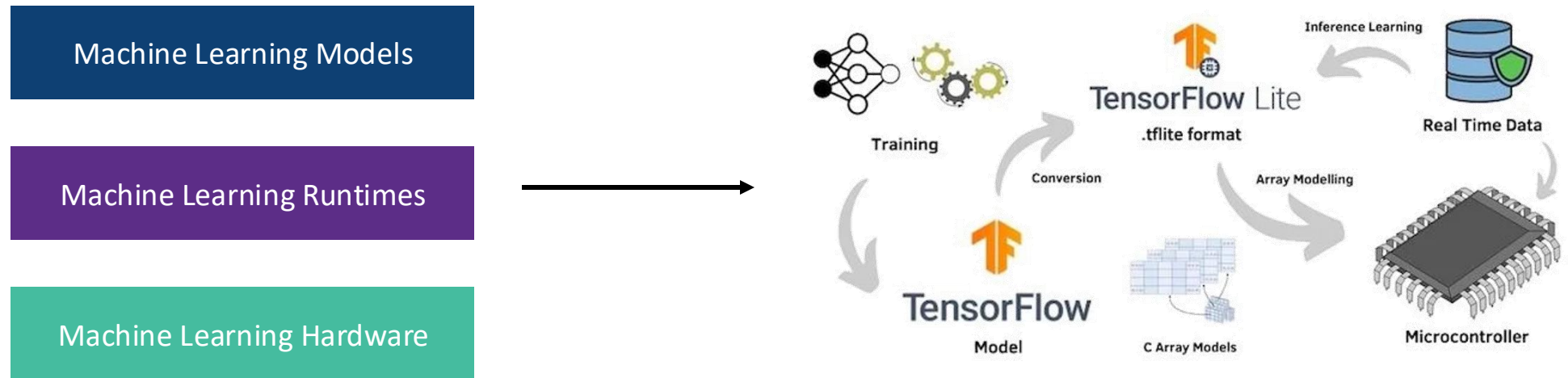
Source: Content on these slides is sourced from <https://github.com/edgeimpulse/courseware-embedded-machine-learning> and <https://github.com/tinyMLx/courseware/tree/master/edX>

How to Deal with This?



Source: S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64 270–64 277, 2018

How to Deal with This?



Source: S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64 270–64 277, 2018

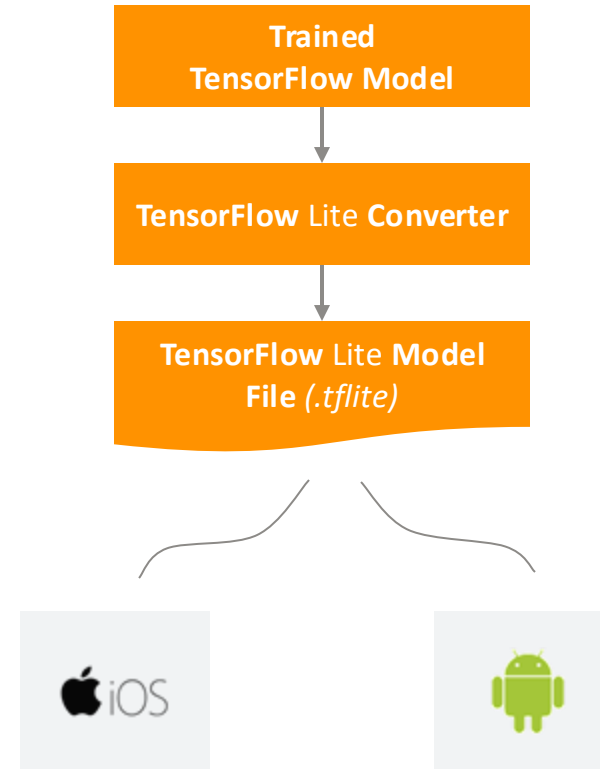
Machine Learning Runtimes

	 TensorFlow	 TensorFlow Lite
Topology	Variable	Fixed
Weights	Variable	Fixed
Binary Size	Unimportant	High Priority
Distributed Compute	Needed	Not Needed
Developer Background	ML Researcher	Application Developer

Machine Learning Runtimes



Architecture

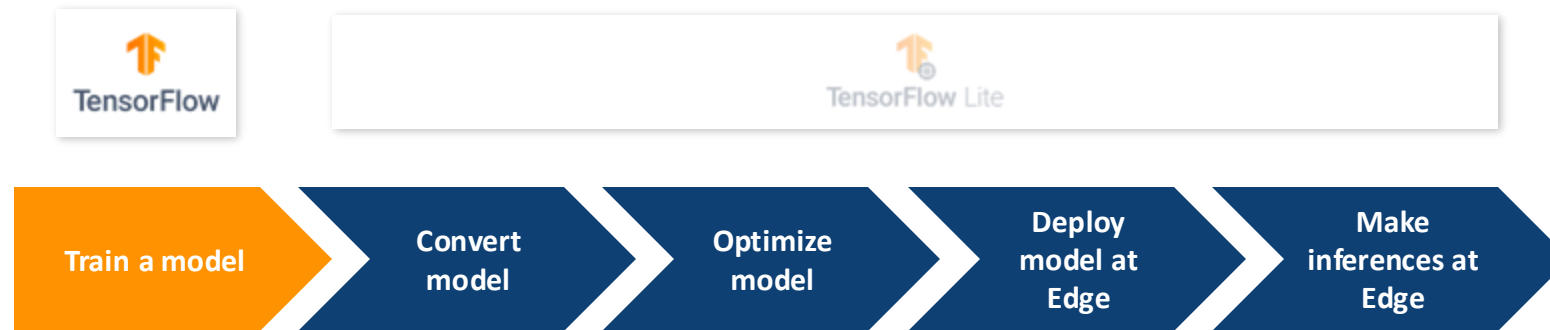


Machine Learning Runtimes

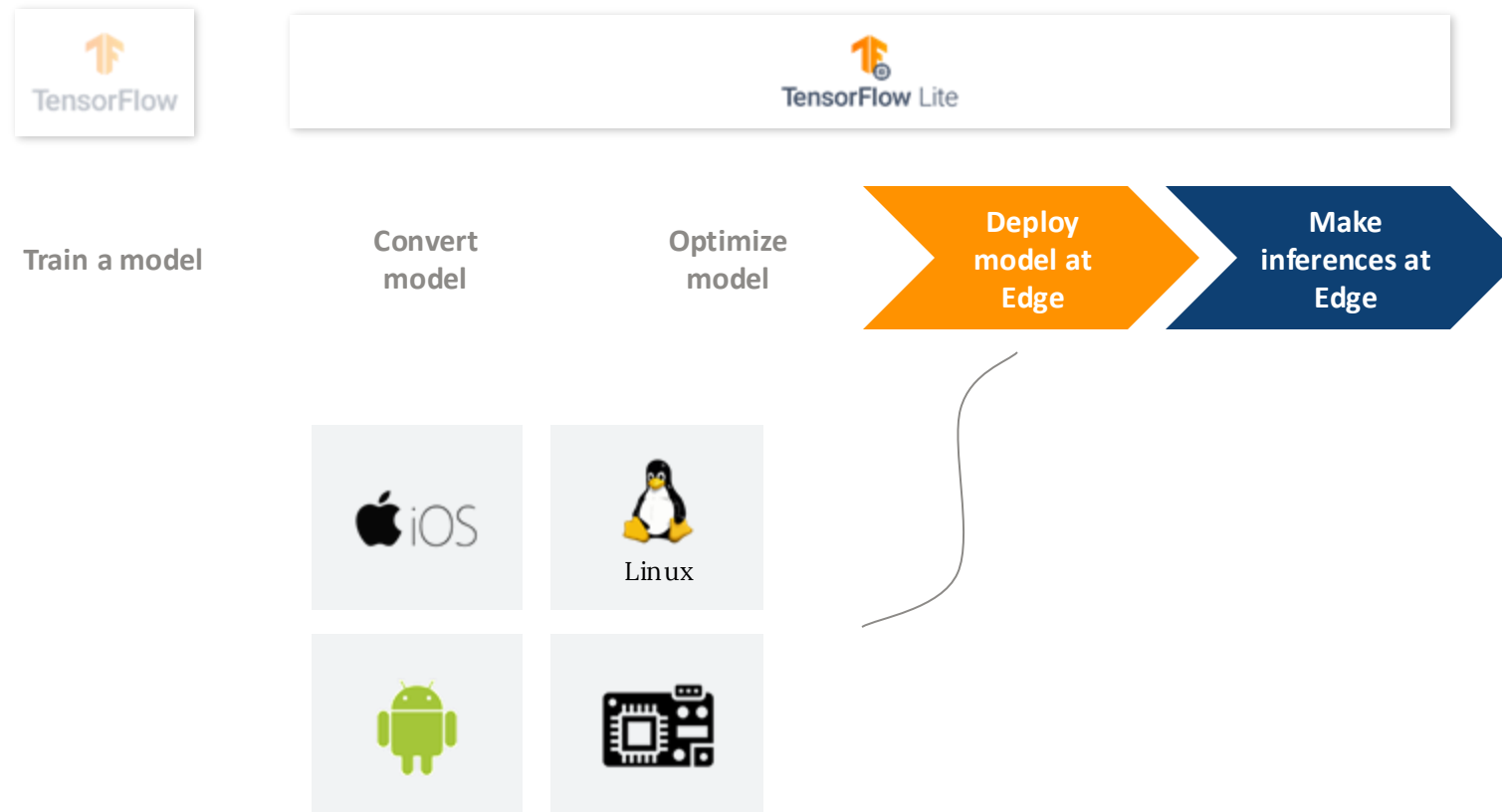


?

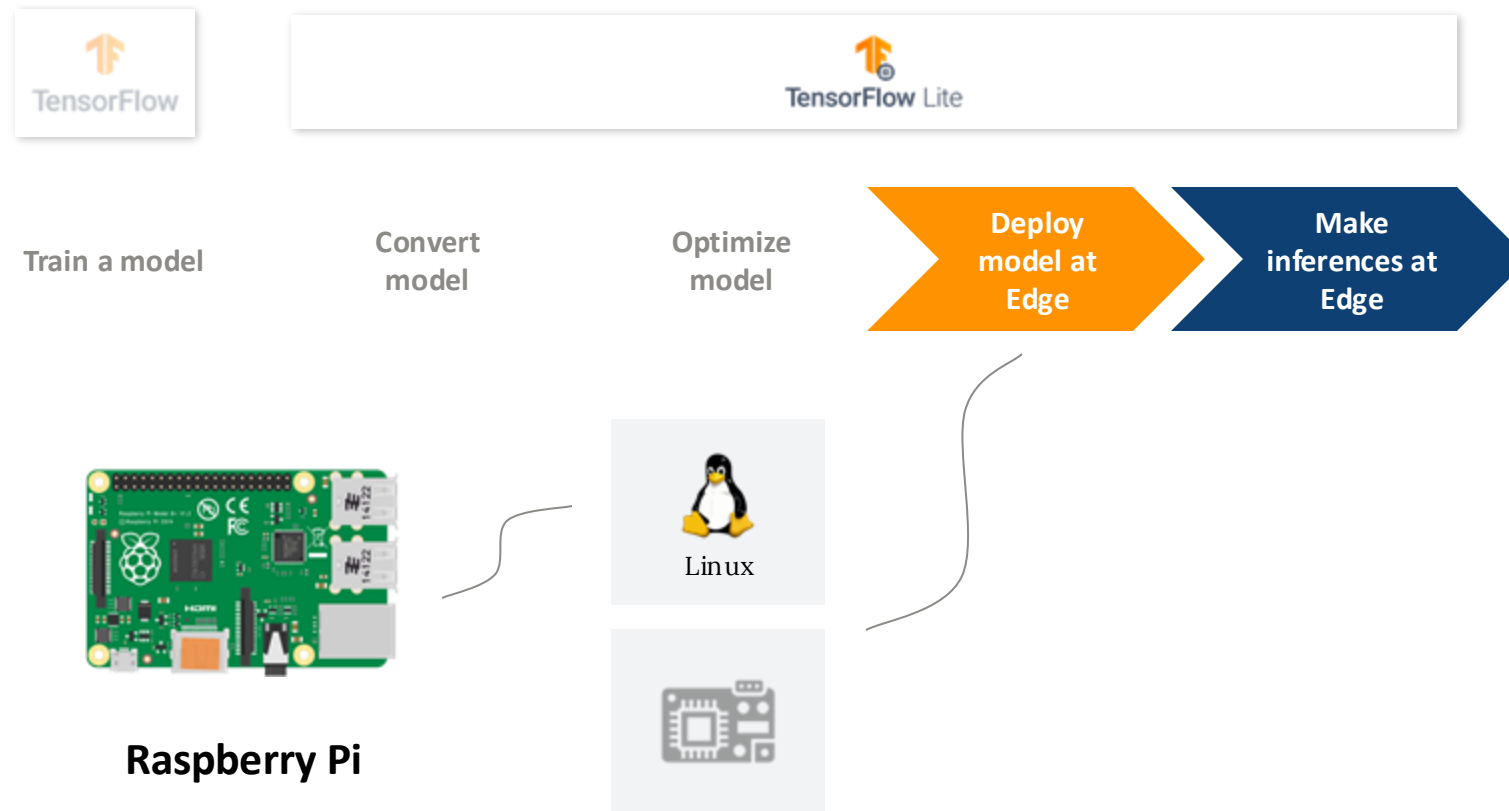
Machine Learning Runtimes



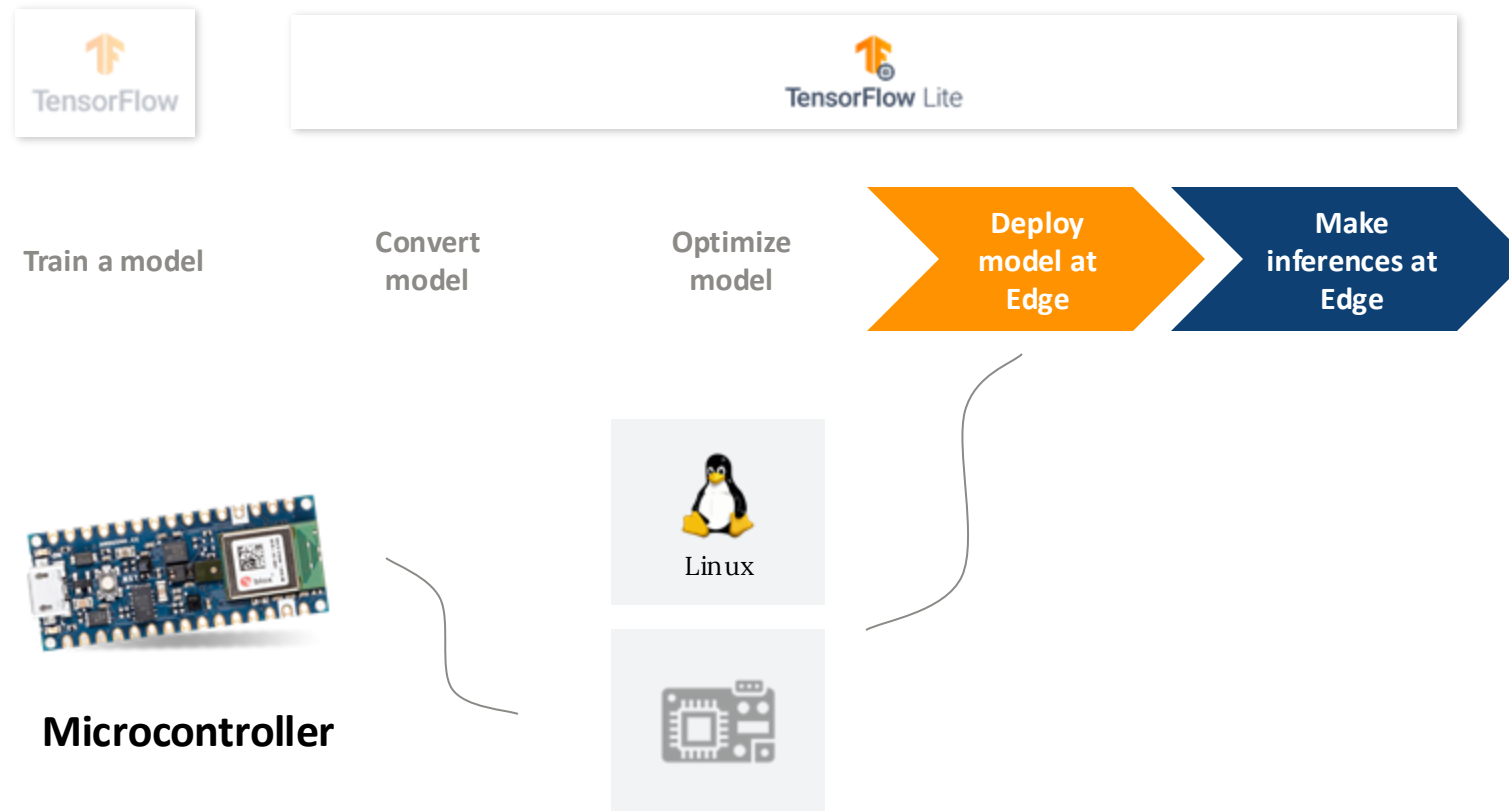
Machine Learning Runtimes



Machine Learning Runtimes



Machine Learning Runtimes



Exercise Part 1 – A 'tiny' TinyML activity

Notebook 1: TinyML Training

- Train a small CNN on MNIST.
- Quantize to INT8 → see size & accuracy differences.
- Export TFLite model & accuracy.
- **Activity:** Run cells, observe FP32 vs INT8 trade-offs.



<https://shorturl.at/kQfav>

Exercise Part 1 – A 'tiny' TinyML activity

Wha the TinyML notebook does

Loads dataset

- Uses **MNIST** (28×28 grayscale digits).
- Normalizes pixel values (0–1).

Defines a small CNN

- Conv → Pool → Conv → Pool → Flatten → Dense → Dense.
- Very lightweight architecture (dozens of KB parameters).
- Designed to be feasible on an MCU-class device.

Trains the model

- Runs for a few epochs on MNIST.
- Achieves reasonable accuracy (e.g., ~0.92 FP32 baseline).

Converts to TensorFlow Lite (TFLite) INT8

- Uses **post-training quantization** with representative samples.
- Forces model to use INT8 for both inputs and outputs.
- Saves to models/mnist_cnn_int8.tflite.

Evaluates INT8 accuracy

- Runs inference with the quantized model.
- Prints INT8 accuracy on test set (e.g., ~0.88).

What it doesn't do

- Doesn't run on real MCU hardware (no flashing).
- Doesn't include pruning or distillation (just quantization).
- Only trains on MNIST (simple dataset, just for demonstration).

Exercise Part 2 – Distributed AI Inference Simulator



Motivation: Real distributed AI systems have heterogeneous devices.



Simulated devices: End-Devices (ex. MCU) , Edge, Cloud.



Requests: Computer Vision tasks + Language Model tasks.

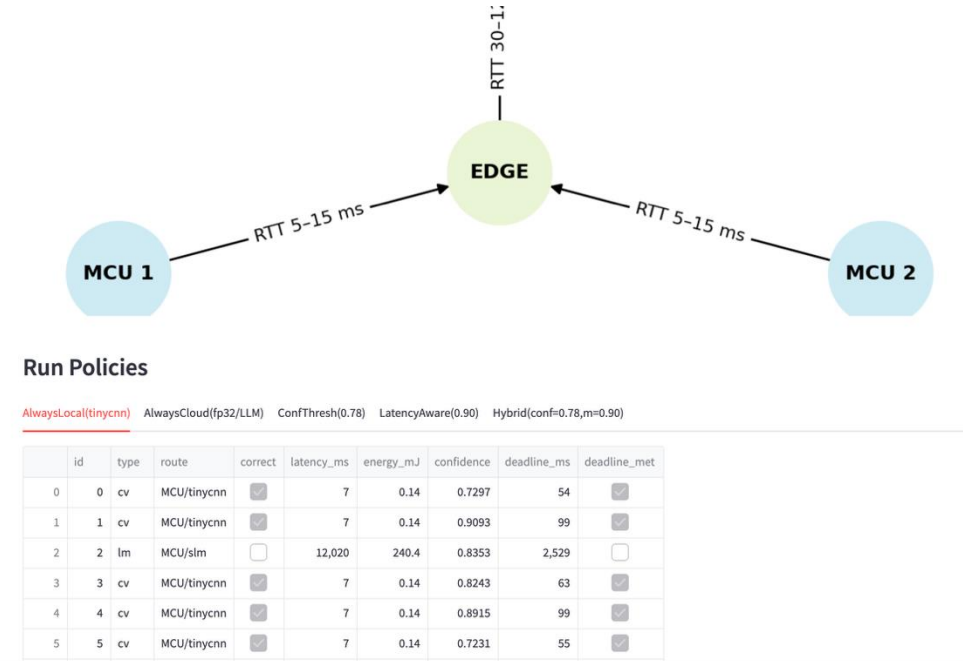


Metrics: latency, accuracy, energy, deadline.

Simulator Overview (i)

Simulator Walkthrough

- Configure devices & network.
- Configure and generate workloads.
- Evaluate baseline AI inference allocation policies:
 - Always Local
 - Always Cloud
 - Confidence Threshold
 - Latency-Aware
 - Hybrid
- *Activity:* Run policies, interpret plots (Accuracy vs Deadline, Energy vs Accuracy).



Exercise Part 2 (ii)

Workload generation

- Mix of *Computer Vision* (easy/hard) and *Language Models* (short/long) requests
- Each has a **deadline_ms** constraint

Policy routing

- Decides where to run each AI inference request: **MCU → EDGE → CLOUD**
- Models: TinyCNN / INT8 / FP32 / SLM / LLM

Simulated inference

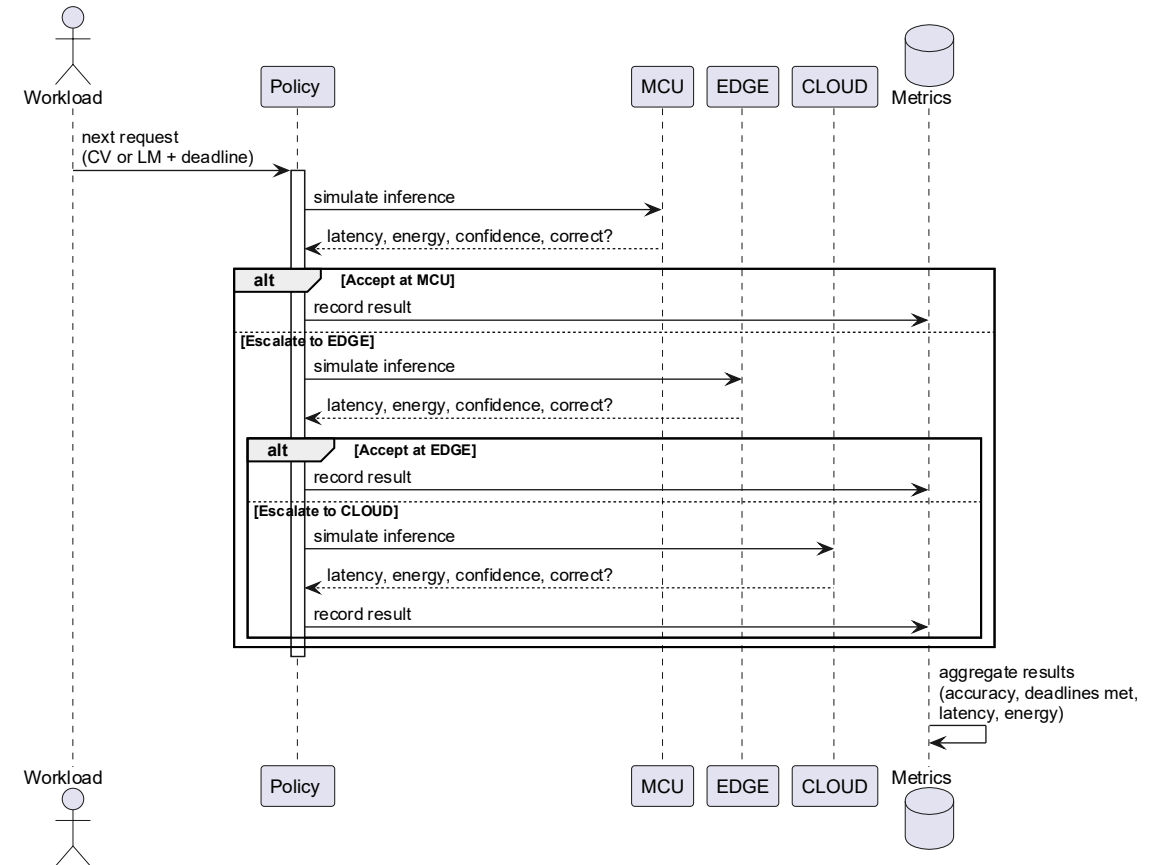
- **Latency** = compute time + network RTT
- **Energy** \propto latency \times device energy factor
- **Confidence** sampled from accuracy/quality
- **Correctness** drawn probabilistically

Decision semantics

- **Confidence Threshold**: accept if confidence \geq thr
- **Deadline Margin**: only use device if latency \leq margin \times deadline

Evaluation

- Record: latency_ms, deadline_met, energy_mJ, correct
- Aggregate to compare policies (plots: Accuracy vs Deadline, Energy vs Accuracy)



Example

Say a CV request comes in:

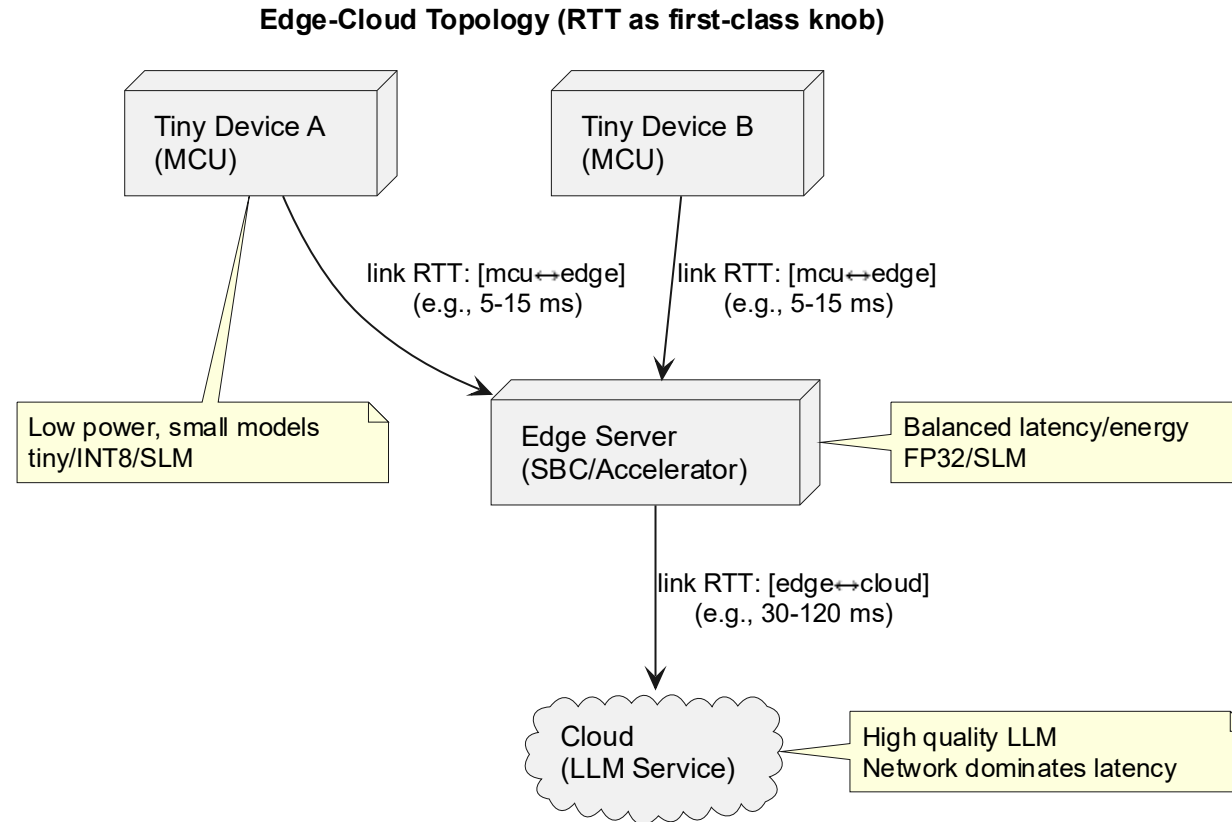
- Difficulty = **hard**, deadline = **80 ms**
- Policy = **Confidence Threshold (0.78)**
- First try MCU/int8: accuracy ~ 0.8 , confidence ~ 0.7 . Since $0.7 < 0.78$, **reject**
- Try EDGE/fp32: accuracy ~ 0.86 , confidence $\sim 0.82 \rightarrow$ accept
- Simulated latency = 35 ms (with RTT). Deadline met (≤ 80)
- Random draw \rightarrow correct = True
- Energy computed ~ 1.2 mJ

Result row =

{route:"EDGE/fp32", correct:True, latency_ms:35, deadline_met:True, energy_mJ:1.2}.

Aggregate across 100+ such requests \rightarrow plots of Accuracy vs Deadline, Energy vs Accuracy, etc.

Exercise Part 2



*Multiple MCUs are shown for realism (different request sources).
Requests are processed independently in series (no queues).*

Exercise Part 2

Configuration

Workload size

10

1

20

CV easy-ratio

0.60

0.00

1.00

CV deadline

20

10

LM request r

0.25

0.00

0.80

MCU RTT low

0

-

+

MCU RTT high

0

-

+

EDGE RTT low

5

-

+

EDGE

RTT 30-1;

RTT 5-15 ms

RTT 5-15 ms

Let's have a look!

AlwaysLocal(tinycnn)

AlwaysCloud(fp32/LLM)

ConfThresh(0.78)

LatencyAware(0.90)

Hybrid(conf=0.78,m=0.90)

	id	type	route	correct	latency_ms	energy_mJ	confidence	deadline_ms	deadline_met
	0	cv	MCU/tinycnn	<input checked="" type="checkbox"/>	7	0.14	0.7297	54	<input checked="" type="checkbox"/>
	1	cv	MCU/tinycnn	<input checked="" type="checkbox"/>	7	0.14	0.9093	99	<input checked="" type="checkbox"/>
	2	lm	MCU/slm	<input type="checkbox"/>	12,020	240.4	0.8353	2,529	<input type="checkbox"/>
	3	cv	MCU/tinycnn	<input checked="" type="checkbox"/>	7	0.14	0.8243	63	<input checked="" type="checkbox"/>
	4	cv	MCU/tinycnn	<input checked="" type="checkbox"/>	7	0.14	0.8915	99	<input checked="" type="checkbox"/>
	5	cv	MCU/tinycnn	<input checked="" type="checkbox"/>	7	0.14	0.7231	55	<input checked="" type="checkbox"/>

Summer School on Edge Artificial Intelligence – KTH, Stockholm

03/September/2025

Distributed Edge AI Simulator



Motivation: Real edge systems have heterogeneous devices.



Simulated devices: End Devices (ex. MCU), Edge, Cloud

Let's walk through the code!



Requests: Computer Vision tasks + Language Model tasks.



Metrics: latency, accuracy, energy, deadline.

Generate_workload() – how tasks are created.

```
# Energy model constant
ENERGY_K = 0.02

#Emulated device profiles (latency ms, energy factor)
DEFAULT_DEVICES = {
    "MCU": {"latency": {"fp32": 80, "int8": 25, "tinycnn": 7, "slm": 12000}, "energy_factor": 1.0, "net_rtt": (0, 0)},
    "EDGE": {"latency": {"fp32": 22, "int8": 10, "tinycnn": 4, "slm": 4000}, "energy_factor": 1.4, "net_rtt": (5, 15)},
    "CLOUD": {"latency": {"fp32": 12, "int8": 12, "tinycnn": 12, "slm": 1500}, "energy_factor": 1.9, "net_rtt": (30, 120)},
}

# Baseline accuracy matrix for "vision-like" tasks
# Accuracy model (per difficulty). Rough, but shows trade-offs.
# You can adjust for your audience: larger gaps → clearer trade-offs.
ACC_MATRIX = {
    "easy": {"fp32": 0.92, "int8": 0.88, "tinycnn": 0.80, "slm": 0.0},
    "hard": {"fp32": 0.86, "int8": 0.80, "tinycnn": 0.70, "slm": 0.0},
}

# For language model (SLM/LLM) requests we model "quality" as accuracy-like metric
LM_QUALITY = {
    "short": {"slm": 0.70, "llm_cloud": 0.90},
    "long": {"slm": 0.60, "llm_cloud": 0.92},
}
```

Generate_workload() – how tasks are created.

```
#Generate workload (10 requests by default)
def generate_workload(n=10, easy_ratio=0.6, deadline_range=(20,120), lm_ratio=0.2):
    rows = []
    for i in range(n):
        if random.random() < lm_ratio:
            length = "long" if random.random() < 0.4 else "short"
            deadline = random.randint(deadline_range[0]+2000, deadline_range[1]+13000)
            rows.append({"id": i, "type": "lm", "length": length, "difficulty": None, "deadline_ms": deadline})
        else:
            difficulty = "easy" if random.random() < easy_ratio else "hard"
            deadline = random.randint(*deadline_range)
            rows.append({"id": i, "type": "cv", "length": None, "difficulty": difficulty, "deadline_ms": deadline})
    print(pd.DataFrame(rows))
    return pd.DataFrame(rows)
```


`infer()` / `Im_infer()` – how devices behave

```
def infer(device_name: str, model_name: str, difficulty: str, devices: dict) -> tuple:
    dev = devices[device_name]
    base_lat = dev["latency"].get(model_name, 999)
    rtt = rnd_net_rtt(dev["net_rtt"])
    latency = base_lat + rtt
    # Energy model (arbitrary but consistent):
    # energy (mJ) = energy_factor * latency_ms * k
    energy = dev["energy_factor"] * latency * ENERGY_K
    # Simulate confidence loosely from accuracy
    acc = ACC_MATRIX.get(difficulty, {}).get(model_name, 0.75)
    conf = float(np.clip(np.random.normal(loc=0.65 + 0.3*acc, scale=0.1), 0.01, 0.999))
    correct = (random.random() < acc)
    return correct, float(latency), float(energy), conf
```

infer() / **lm_infer()** – how devices behave

```
def lm_infer(route: str, length: str, devices: dict) -> tuple:
    """
    route: 'MCU/slm', 'EDGE/slm', or 'CLOUD/llm_cloud'
    length: 'short' or 'long'
    Returns (correct, latency_ms, energy_mJ, confidence, quality)
    """
    if route.startswith("CLOUD"):
        base = devices["CLOUD"]["latency"].get("slm", 1500)
        rtt = rnd_net_rtt(devices["CLOUD"]["net_rtt"])
        latency = base + rtt + (10 if length=="long" else 5)
        energy = devices["CLOUD"]["energy_factor"] * latency * ENERGY_K
        quality = LM_QUALITY[length]["llm_cloud"]
    else:
        tier = "EDGE" if route.startswith("EDGE") else "MCU"
        base = devices[tier]["latency"].get("slm", 12000)
        rtt = rnd_net_rtt(devices[tier]["net_rtt"])
        latency = base + rtt + (20 if length=="long" else 8)
        energy = devices[tier]["energy_factor"] * latency * ENERGY_K
        quality = LM_QUALITY[length]["slm"]
    correct = (random.random() < quality)
    conf = float(np.clip(np.random.normal(loc=0.6 + 0.3*quality, scale=0.1), 0.01, 0.999))
    return correct, float(latency), float(energy), conf, quality
```

Policies (always_local, conf_threshold, etc.) – design ideas

```
def pol_conf_threshold(req, devices, thr=0.78):
    if req["type"]=="lm":
        c1, l1, e1, conf1, _q1 = lm_infer("MCU/slm", req["length"], devices)
        if conf1 >= thr:
            return "MCU/slm", c1, l1, e1, conf1
        c2, l2, e2, conf2, _q2 = lm_infer("EDGE/slm", req["length"], devices)
        if conf2 >= thr:
            return "EDGE/slm", c2, l2, e2, conf2
        c3, l3, e3, conf3, _q3 = lm_infer("CLOUD/llm_cloud", req["length"], devices)
        return "CLOUD/llm_cloud", c3, l3, e3, conf3
    else:
        c1, l1, e1, conf1 = infer("MCU", "int8", req["difficulty"], devices)
        if conf1 >= thr:
            return "MCU/int8", c1, l1, e1, conf1
        c2, l2, e2, conf2 = infer("EDGE", "fp32", req["difficulty"], devices)
        if conf2 >= thr:
            return "EDGE/fp32", c2, l2, e2, conf2
        c3, l3, e3, conf3 = infer("CLOUD", "fp32", req["difficulty"], devices)
        return "CLOUD/fp32", c3, l3, e3, conf3
```

evaluate_policy() – how we test them

```
def evaluate_policy(workload_df: pd.DataFrame, policy_fn, devices: dict) -> pd.DataFrame:
    out = []
    for _, req in workload_df.iterrows():
        route, correct, latency, energy, conf = policy_fn(req, devices)
        out.append({
            "id": int(req["id"]), "type": req["type"], "route": route, "correct": bool(correct),
            "latency_ms": float(latency), "energy_mJ": float(energy), "confidence": float(conf),
            "deadline_ms": int(req["deadline_ms"]), "deadline_met": float(latency) <= float(req["deadline_ms"])
        })
    print(pd.DataFrame(out))
    return pd.DataFrame(out)
```

Exercise Part 2 – Group Work Setup

- Form groups of 3–5 participants.
 - But feel free to work alone :)
 - Each group extends the simulator with one or more new ideas / features.
 - Work in the code, document what you tried + results.
- You don't need to “finish” → just experiment.
 - Focus on **what changes in the trade-offs**.
 - Capture a quick result (plot or table).
 - Prepare a 2 / 3 minute summary

Challenge Cards

Option 1: Enrich LM modeling (e.g., task length scaling).

Option 2: More realistic energy models (battery, DVFS).

Option 3: Add new accelerators (GPU, TPU) → new latencies.

Option 4: Vary traffic (bursty, arrival rates).

Option 5: New topologies (multi-edge, hierarchical).

Option 6: Smarter policies (queue-aware, RL-inspired).

Option 7: Modify Quality

Option 8: Always edge

Option 9: Your choice!!

Group Work

Suggested flow for the 50-min group session

Explore the Code (5–10 min)

1. Skim through `simulate.py` and notebook cells
2. Make sure everyone in the group understands the baseline

Brainstorm Extensions (5–10 min)

1. Discuss which challenge card to pick (LM modeling, accelerators, energy, traffic...)
2. Agree on one idea to prototype

Implement & Test (25–30 min)

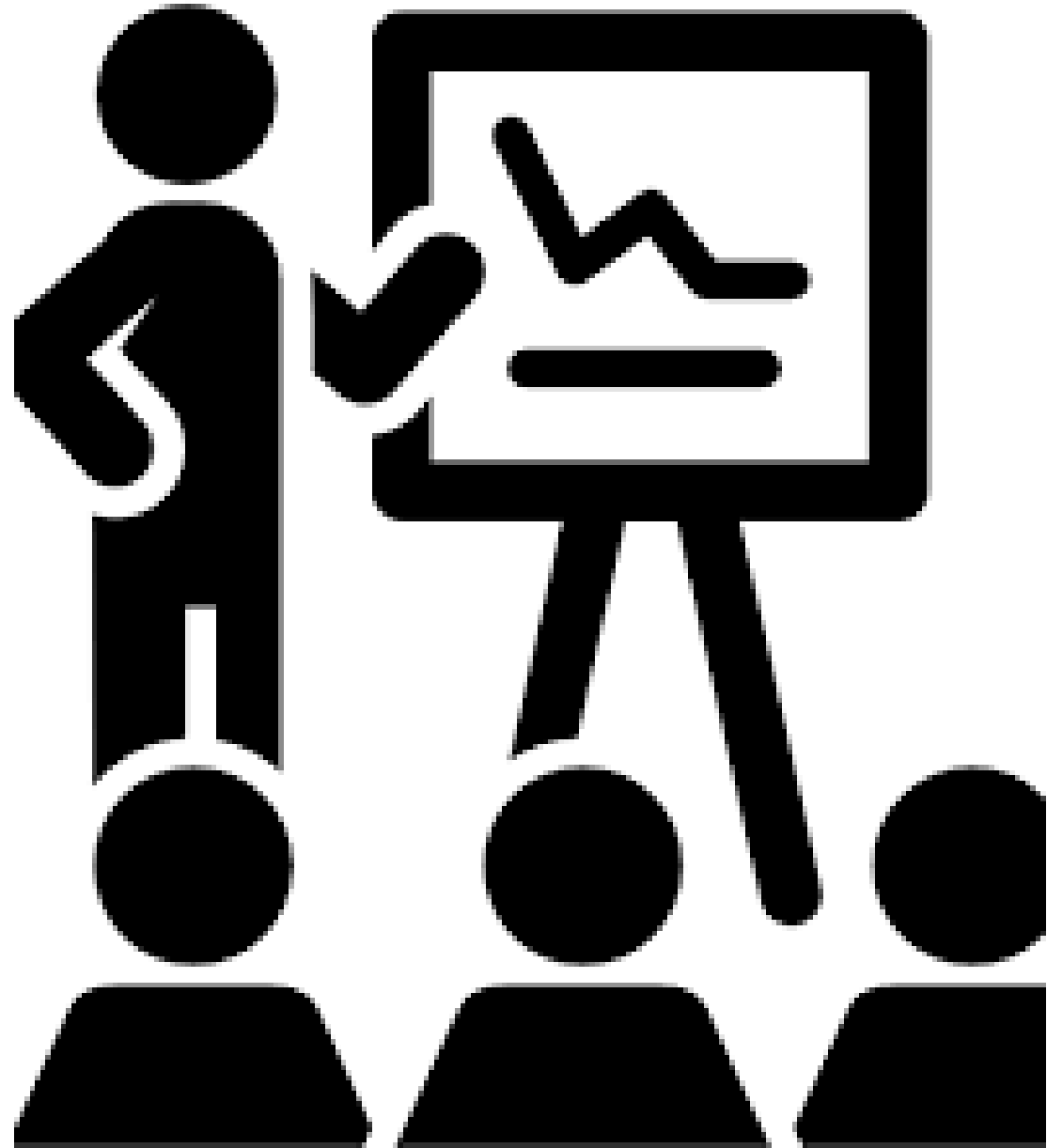
1. Modify the simulator or notebook
2. Run a few policies and collect quick results (table or plot)



Group Presentations

Each group:

- What extension did you try?
- What changed in results?
- One insight or question.



Exercise Session

Distributed Edge AI (and TinyML) Systems

Roberto Morabito
Assistant Professor @ EURECOM
<https://www.linkedin.com/in/robertomorabito>