EAI 805 — Embedded AI Hardware

Week 3 Lecture Notes — Comparison of Embedded AI Hardware Platforms

Duration: two-hour lecture (120 minutes) · Platform emphasis: AMD/Xilinx PYNQ-Z2 (Zynq-7020 SoC)

Session at a Glance

Paced for a single two-hour block with one short break. Section 4 (roofline) and Section 6 (PYNQ measurement) are the two segments worth protecting if time runs short.

Time	Segment	Format
00–10 min	Recap of Week 2; why platform comparison is genuinely hard	Lecture
10–35 min	The five quantitative axes: TOPS/W, latency, area, $/TOPS, model size	Lecture + worked numbers
35–55 min	Platform survey: CPU, GPU, TPU, MCU+NPU, FPGA	Lecture + comparison table
55–65 min	Break	—
65–90 min	Roofline analysis and the ridge point	Lecture + PYNQ-Z2 worked example
90–105 min	Choosing a platform for a given workload	Decision framework + case discussion
105–120 min	Measuring it yourself on the PYNQ-Z2 + Lab 2 preview	Live notebook demo

Learning Objectives

By the end of this lecture, a student should be able to:

- Define the five comparison axes — TOPS/W, latency, area, $/TOPS, and model-size limits — and explain what each does and does not tell you.
- Explain why peak TOPS is a poor single-figure metric, and why achieved throughput on a real model matters more.
- Contrast the architectural character of CPU, GPU, TPU/ASIC, MCU+NPU, and FPGA platforms for edge inference.
- Compute arithmetic intensity for a layer, place it on a roofline, and determine whether a workload is compute-bound or memory-bound.
- Construct an approximate roofline for the PYNQ-Z2 and use it to predict where an accelerator design will be limited.
- Apply a structured decision framework to select a platform for a stated workload and set of constraints.
- Measure latency and throughput on the PYNQ-Z2 from a Jupyter notebook, and interpret the result honestly.

1. Why Platform Comparison Is Harder Than It Looks

Week 2 examined the internal architecture of one device — the Zynq-7020 — and the AXI channels that move data inside it. Week 3 zooms out: given a workload, how do you choose between the very different classes of silicon on the market, and how do you defend that choice with numbers?

The difficulty is that vendors compete on a single headline number, usually peak **TOPS** (tera-operations per second). Peak TOPS is a theoretical ceiling: it assumes every arithmetic unit is busy every cycle, with data always available. Real inference rarely comes close, because the accelerator spends much of its time waiting

for weights and activations to arrive from memory. A platform quoting 4 TOPS may deliver a small fraction of that on your network — and a different platform quoting fewer TOPS may beat it.

Two consequences shape the whole lecture. First, always distinguish **peak** from **achieved** performance, and prefer measured latency on your own model over any datasheet figure. Second, comparison is multi-dimensional: a platform that wins on throughput may lose badly on power, unit cost, or the maximum model size it can hold. The engineer's job is to state the constraints first, then find the platform that satisfies all of them — not to maximise one number.

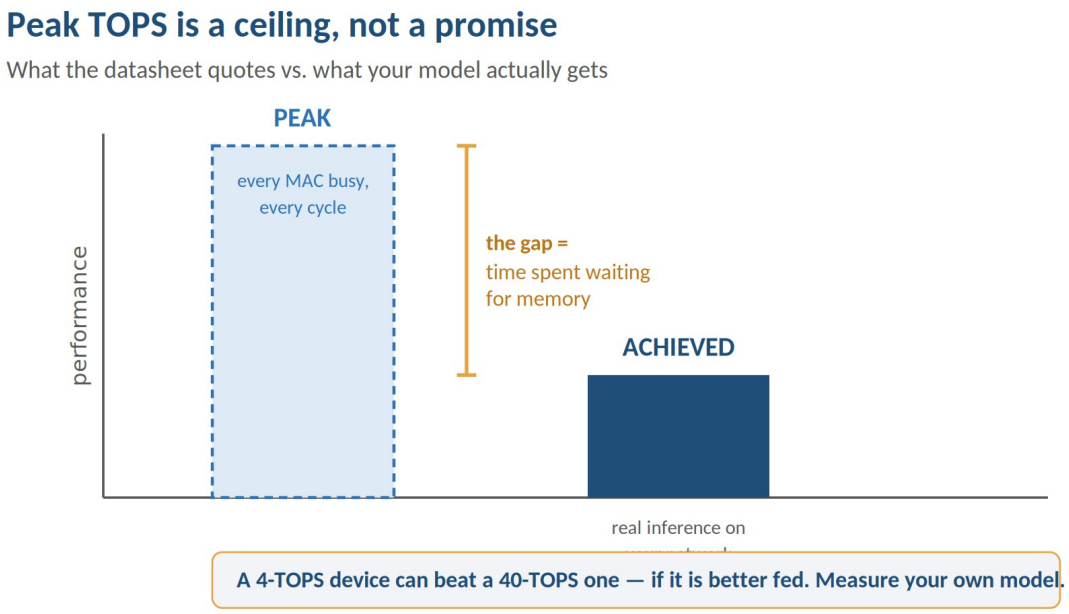

Figure 1. *Peak TOPS assumes every arithmetic unit is busy every cycle; achieved performance on a real model is far lower. The gap is time spent waiting for memory — which is why a lower-peak device can beat a higher-peak one.*

2. The Five Quantitative Comparison Axes

2.1 Throughput and efficiency: TOPS and TOPS/W

TOPS counts arithmetic operations per second, conventionally at a stated precision (almost always INT8 at the edge). A multiply-accumulate is usually counted as two operations. **TOPS/W** divides that by power, and is the single most important axis for battery- or thermally-constrained deployment: it tells you how much inference you buy per joule.

Read both sceptically. Ask: at what precision? Peak or measured? Does the power figure include the memory system and the host, or only the accelerator core? Is sparsity assumed? Two "equal TOPS" devices can differ by an order of magnitude in delivered performance once these are pinned down.

2.2 Latency (and its relationship to throughput)

Latency is the time from input to result for a single inference — the metric that matters for a control loop, a robot, or anything interactive. **Throughput** is inferences per second in steady state. They are not reciprocals: a pipelined or batched design raises throughput while leaving (or worsening) single-sample latency. Batching is the classic example — excellent for server GPUs, often useless at the edge where samples arrive one at a time.

Report latency as a distribution, not a mean. Tail latency (p95/p99) is what breaks real-time deadlines, and it is where shared, OS-scheduled platforms (CPU, GPU under Linux) tend to look far worse than deterministic fabric-based designs.

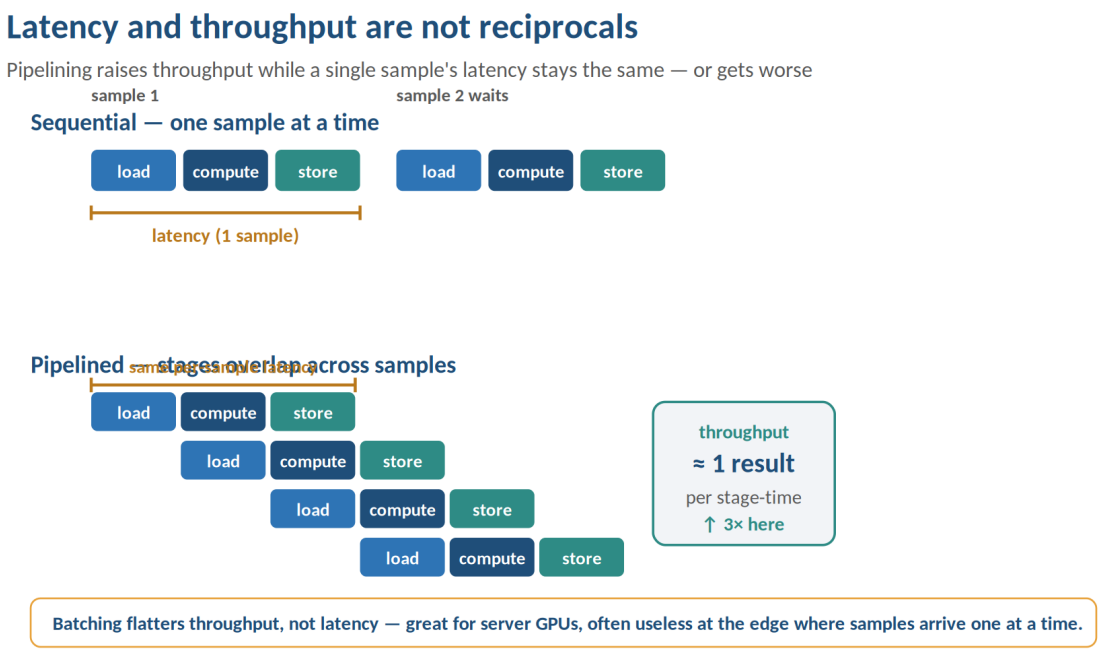

Figure 2. *Latency and throughput are not reciprocals. Pipelining and batching overlap work to raise throughput, but a single sample's latency is unchanged (or worse) — the reason batching helps server GPUs yet rarely helps at the edge.*

2.3 Area (and the FPGA equivalent)

For an ASIC or IP block, **area** (mm^2 at a given process node) drives die cost and power. The Arm Ethos-U55 microNPU, for instance, is quoted at roughly 0.1 mm^2 — which is why it can be dropped into a cost-sensitive microcontroller.

On an FPGA the equivalent currency is **resource utilisation**: LUTs, flip-flops, BRAM, and above all DSP slices. The Zynq-7020 on the PYNQ-Z2 has a fixed budget (on the order of 53k LUTs, 140 BRAM blocks, 220 DSP slices). Your accelerator's "area" is the fraction of that budget it consumes, and it directly caps how much parallelism you can instantiate. This is why resource reports are part of every lab deliverable in this course.

2.4 Cost: $/TOPS and total system cost

$/TOPS normalises purchase price by compute, and is the axis that decides whether a design ships at volume. But it is easily gamed and easily misread: the honest figure is total system cost — module plus memory, power supply, thermal solution, carrier board, certification, and the engineering effort of the toolchain. An FPGA can win on flexibility and lose on non-recurring engineering time; an ASIC accelerator wins on unit cost only at volume.

2.5 Model-size limits

A platform is useless if the model does not fit. Two ceilings matter: the on-chip memory that holds weights and activations at speed (SRAM, BRAM, cache), and the off-chip memory that holds the rest. A microcontroller-class NPU may have tens of kilobytes of local SRAM; the PYNQ-Z2 has a few megabits of BRAM and 512 MB of DDR3; a Jetson module has gigabytes of LPDDR. This axis often decides the shortlist before any performance number is considered — it is the first filter, not the last.

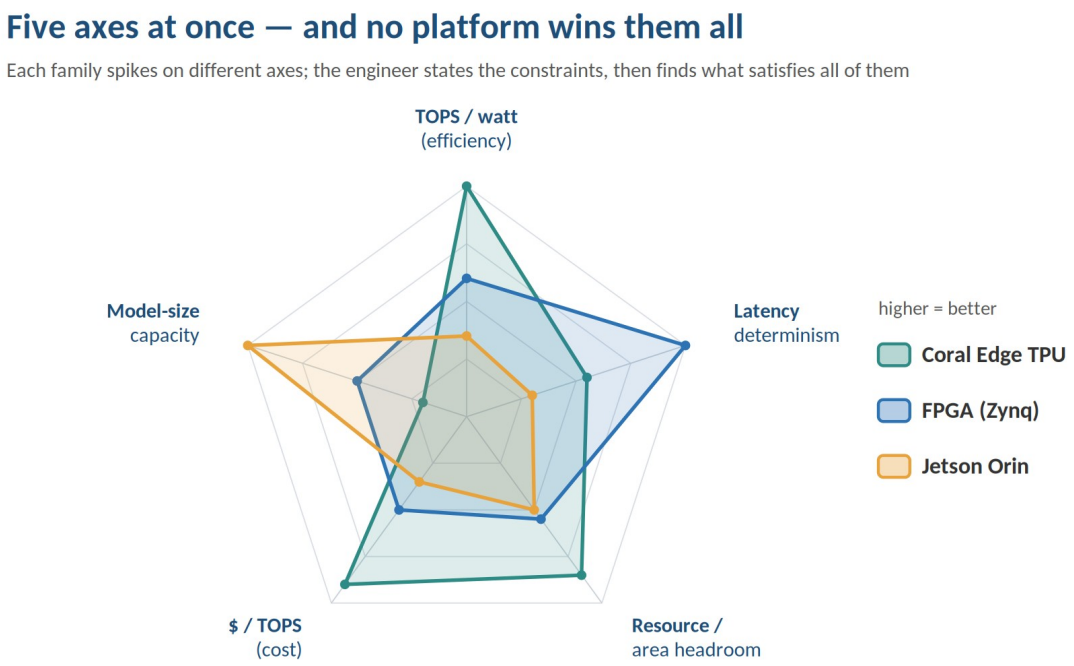

Figure 3. *The five axes plotted together for three archetypes (higher = better). No family wins on every axis: the Edge TPU spikes on efficiency, the FPGA on latency determinism, the Jetson on model capacity.*

2.6 Summary of the axes

Axis	What it measures	Main pitfall
TOPS / TOPS-per-watt	Peak arithmetic rate; efficiency per joule	Peak ≠ achieved; precision and power boundary often unstated
Latency	Time for one inference	Means hide the tail; batching flatters throughput, not latency
Area / resources	Die area (ASIC) or LUT/BRAM/DSP use (FPGA)	On FPGA it caps parallelism — must be reported with the design
$/TOPS	Purchase cost per unit of compute	Ignores memory, board, thermals, and engineering effort
Model-size limit	Largest model that fits and runs	On-chip vs off-chip capacity are different ceilings

3. The Platforms

Five architectural families dominate embedded AI. Each represents a different answer to the same question: how much flexibility do you trade for efficiency?

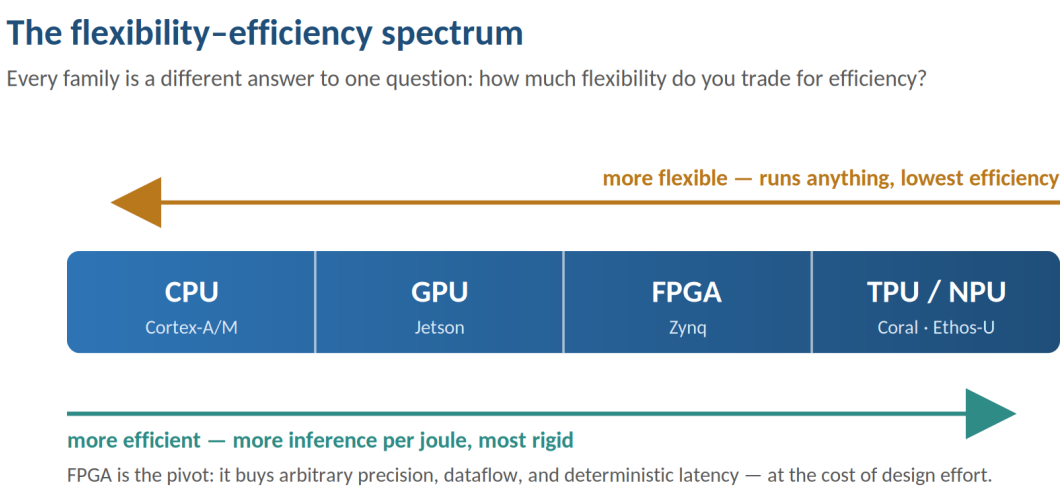

Figure 4. *The five families arranged along the flexibility–efficiency spectrum. Moving right buys efficiency and loses generality; the FPGA is the pivot, trading design effort for arbitrary precision, dataflow, and deterministic latency.*

3.1 CPU — Arm Cortex-A / Cortex-M

The general-purpose baseline. Cortex-A cores (such as the dual Cortex-A9 in the PS of our Zynq-7020) run Linux and any framework, with NEON SIMD giving modest vector throughput. Maximum flexibility, minimum efficiency: no dedicated MAC array, so inference is limited by SIMD width and memory bandwidth. Its real role at the edge is as the control processor and baseline — the number your accelerator must beat, which is exactly how Lab 2 uses it.

3.2 GPU — NVIDIA Jetson

Massively parallel SIMT cores plus tensor/deep-learning accelerators, with a mature software stack (CUDA, TensorRT). The Jetson Orin family spans roughly 20–40 TOPS for the entry Orin Nano at 5–15 W, through Orin NX, up to 275 TOPS on AGX Orin at 15–60 W; a software update raised the Orin Nano to about 67 TOPS in "Super" mode at 7–25 W. Strengths: throughput, large models, fast development, excellent batching. Weaknesses: power and cost at the low end, and non-deterministic latency under a general-purpose OS.

3.3 TPU / fixed-function ASIC — Google Coral Edge TPU

A hardened systolic array specialised for INT8 TensorFlow Lite inference. The Edge TPU delivers 4 TOPS at about 2 W — roughly 2 TOPS/W — and runs MobileNet v2 at close to 400 FPS. It is the efficiency champion within its niche, and almost useless outside it: the model must be quantized to INT8 and compiled to supported operators, or layers fall back to the host CPU and performance collapses. Excellent when your workload is a standard vision CNN; inflexible when it is not.

3.4 MCU + NPU — Arm Ethos-U55 with Cortex-M

The TinyML end of the spectrum. The Ethos-U55 is a configurable microNPU (32, 64, 128 or 256 MAC units) delivering roughly 64–512 GOP/s at 1 GHz in about 0.1 mm², paired with a Cortex-M55/M7/M4/M33 host, running bare-metal or under an RTOS with TensorFlow Lite for Microcontrollers. Internal SRAM is measured in tens of kilobytes, so models must be small and heavily quantized (INT8/INT16), with weight compression. Unbeatable for always-on keyword spotting or anomaly detection at milliwatts; incapable of anything large.

3.5 FPGA — Zynq-7000 and Zynq UltraScale+

Reconfigurable fabric: you build the datapath rather than mapping onto a fixed one. This buys three things the others cannot offer — **arbitrary precision** (binary, 2-bit, 4-bit, mixed), **dataflow architectures** that stream layer-to-layer without round-tripping through DRAM, and **deterministic latency**. The Zynq-7020 on the PYNQ-Z2 is deliberately modest (220 DSP slices, ~53k LUTs); Zynq UltraScale+ MPSoC devices scale the

same model up substantially and add hardened accelerators. The cost is development effort: you are designing hardware, and the toolchain is the barrier the rest of this course exists to lower.

3.6 Comparison at a glance

Indicative figures for orientation only — always re-check current vendor data and, more importantly, measure your own model.

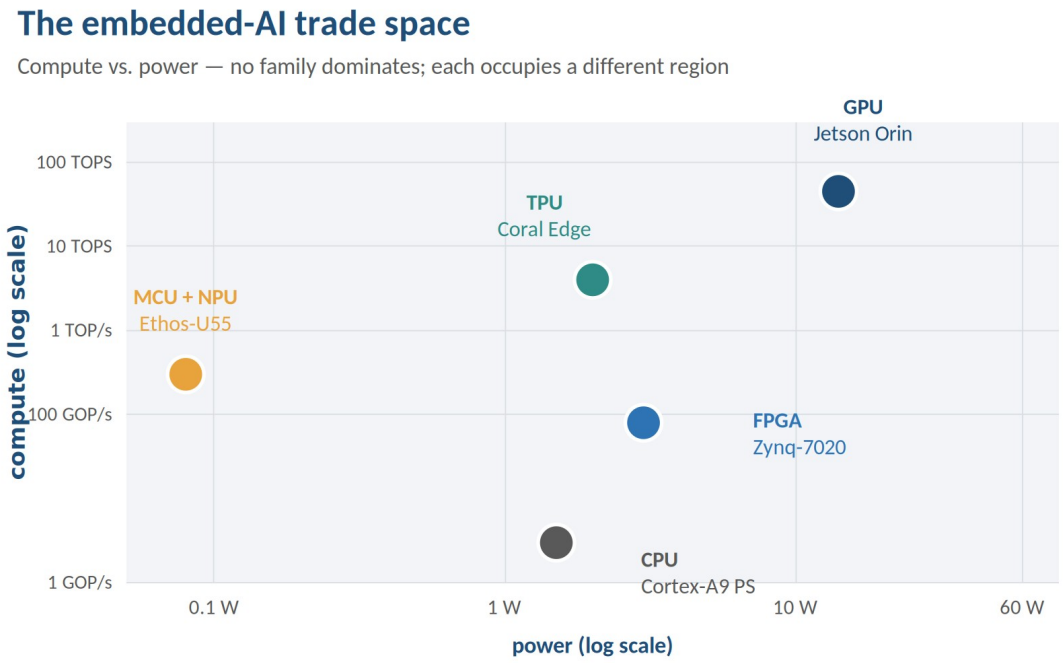

Figure 5. *The embedded-AI trade space on log–log axes of power against compute. Each family occupies a distinct region — a map of where the shortlist comes from before any measurement.*

Platform	Typical compute	Power	Precision	Character
CPU (Cortex-A9, PYNQ-Z2 PS)	~GOP/s class	~1–2 W	FP32/INT8	Baseline; total flexibility, lowest efficiency
GPU (Jetson Orin Nano)	20–40 TOPS (up to ~67)	5–25 W	INT8/FP16	Throughput and big models; mature stack
TPU (Coral Edge TPU)	4 TOPS	~2 W	INT8 only	~2 TOPS/W; superb in-niche, rigid outside it
MCU+NPU (Ethos-U55)	64–512 GOP/s	mW class	INT8/INT16	TinyML; ~0.1 mm², tens of KB SRAM
FPGA (Zynq-7020)	Design-dependent	~2–5 W	Arbitrary (1–8 bit)	Custom dataflow, deterministic latency

4. Roofline Analysis

The roofline model (Williams, Waterman and Patterson, 2009) is the standard tool for answering "what is actually limiting me?" It plots attainable performance against **arithmetic intensity** — the ratio of operations performed to bytes moved from memory.

4.1 The model

Attainable performance is the lower of two ceilings:

attainable OPS = min(peak compute, arithmetic intensity × peak memory bandwidth)

Plotted on log-log axes this gives a sloped memory-bound region on the left and a flat compute-bound plateau on the right. The **ridge point** — where the slope meets the plateau — is the arithmetic intensity a workload must exceed to be limited by arithmetic rather than by data movement. A high ridge point means a machine that is hard to feed.

The diagnosis it gives is actionable. Left of the ridge (memory-bound): adding multipliers changes nothing; increase data reuse through tiling, keep intermediates on-chip, use a dataflow architecture, or reduce precision so each value costs fewer bytes. Right of the ridge (compute-bound): add parallelism — more DSPs, wider vectors, lower precision to fit more MACs.

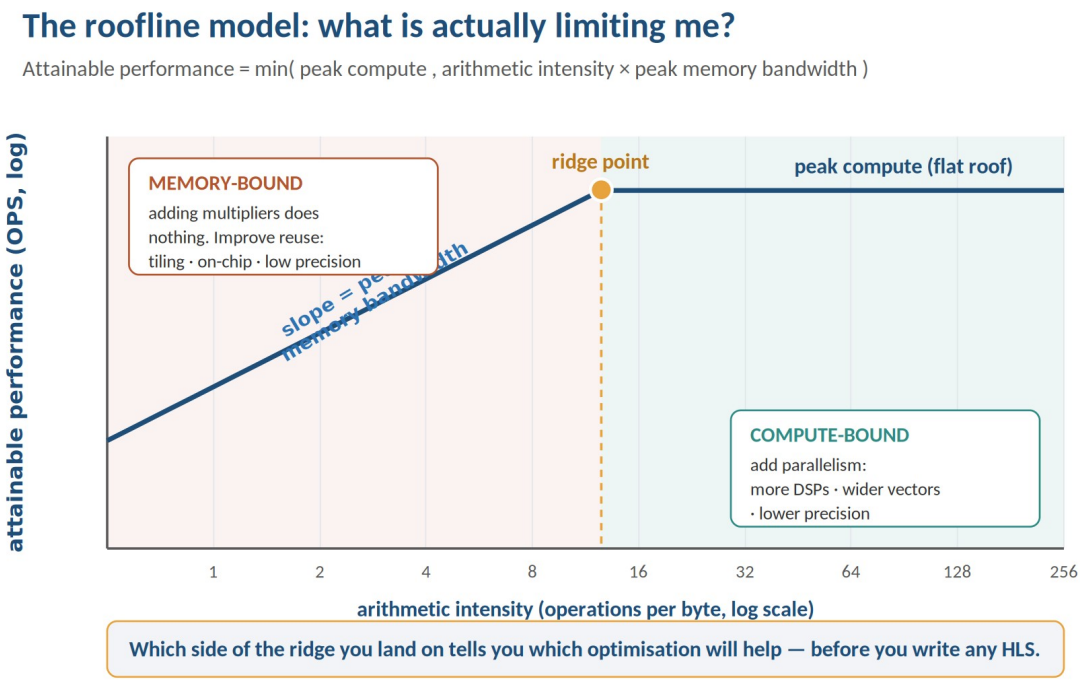

Figure 6. *The roofline model. Attainable performance is the lower of a sloped memory-bound roof and a flat compute-bound roof; the ridge point is the arithmetic intensity a workload must exceed to be limited by compute. Which side you land on dictates which optimisation helps.*

4.2 Arithmetic intensity of common layers

Intensity is a property of the layer and the dataflow, not of the hardware:

- **Fully-connected / matrix-vector:** each weight is used once per inference, so intensity is roughly 2 ops per weight byte — very low. Almost always memory-bound. This is why MLPs and LSTM layers stress bandwidth, not multipliers.
- **Convolution:** each weight is reused across every spatial position, so intensity is high and grows with feature-map size — usually compute-bound, and the reason CNNs map so well onto MAC arrays.
- **Depthwise-separable convolution:** far fewer operations per weight than standard convolution, so intensity drops sharply. MobileNet-class networks are much closer to memory-bound than their FLOP count suggests — a classic trap when predicting speed-up.

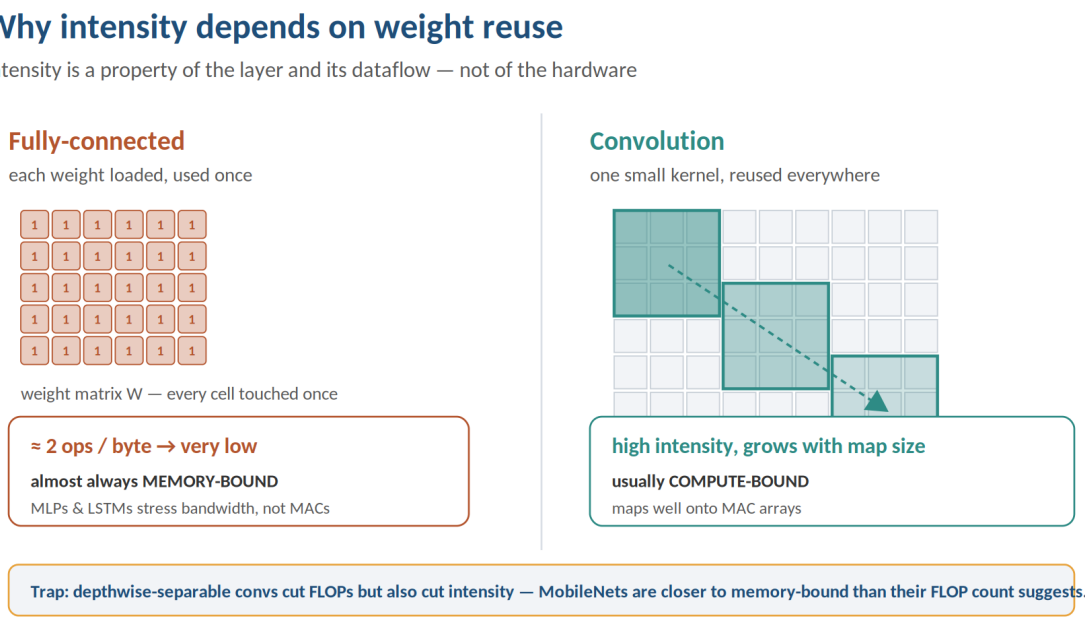

Figure 7. *Arithmetic intensity follows weight reuse. A fully-connected layer uses each weight once (low intensity, memory-bound); a convolution reuses one kernel across every spatial position (high intensity, compute-bound).*

4.3 Worked example: an approximate roofline for the PYNQ-Z2

Order-of-magnitude arithmetic, done deliberately roughly, to show the method. Treat every number as illustrative rather than a specification.

Compute ceiling. The Zynq-7020 has 220 DSP48E1 slices. Running the fabric at 100 MHz and counting one MAC (two operations) per DSP per cycle:

```
220 DSP × 100 MHz × 2 ops  ≈  44 GOP/s
```

Low-precision designs beat this substantially — packing several INT4 or binary operations into one DSP, or implementing MACs in LUTs, can multiply the effective rate. That headroom is exactly why the FINN/BNN approach exists, and it is the Week 8 case study.

Memory ceiling. The PYNQ-Z2 carries 512 MB of DDR3 on a 16-bit interface, giving a theoretical peak on the order of 2 GB/s (real sustained bandwidth is lower).

Ridge point. 44 GOP/s ÷ 2 GB/s ≈ 22 operations per byte.

Interpretation — and this is the point of the exercise. A layer with intensity below roughly 20 ops/byte on this board is memory-bound: the DSPs will idle no matter how cleverly you schedule them. A fully-connected layer (about 2 ops/byte) sits far to the left, so accelerating it with more multipliers is wasted effort; the fix is to avoid the DRAM round-trip. A convolution with good on-chip reuse sits to the right and can genuinely use the fabric. This single calculation predicts, before you write any HLS, which parts of a network are worth accelerating.

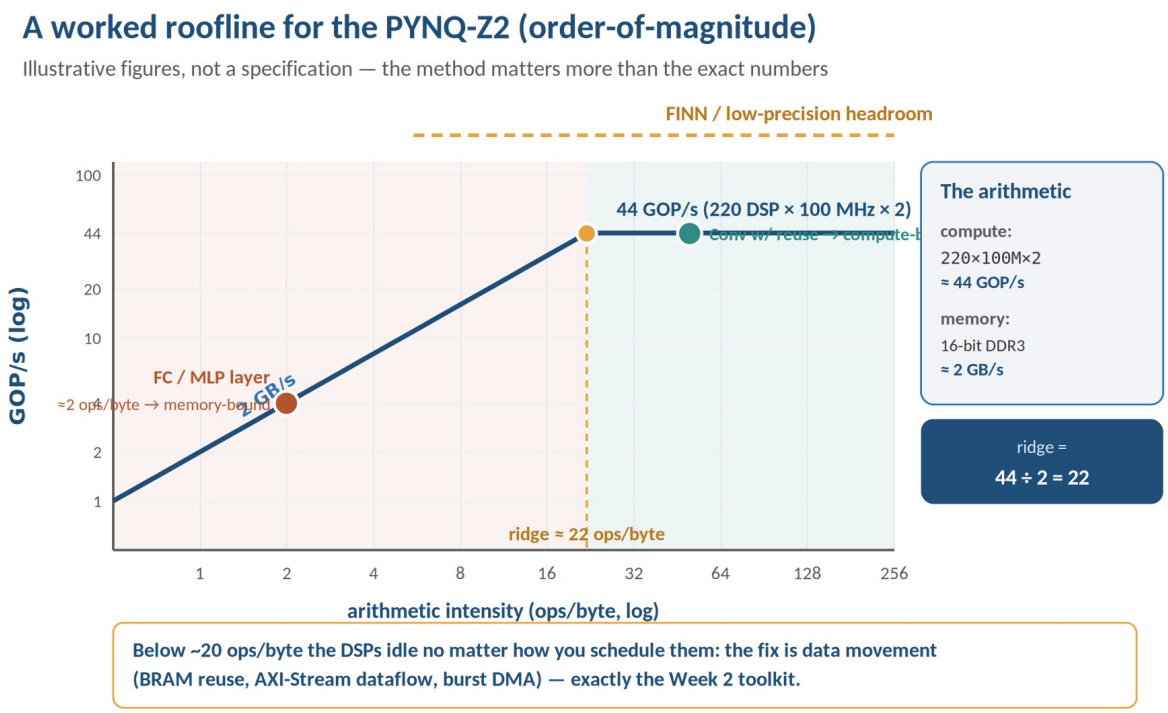

Figure 8. *An order-of-magnitude roofline for the PYNQ-Z2: a 44 GOP/s compute ceiling over roughly 2 GB/s of bandwidth gives a ridge near 22 ops/byte. A fully-connected layer sits far left (memory-bound); a convolution with reuse reaches the plateau. Figures are illustrative, not a specification.*

Notice how directly this connects to Week 2: staying left of the ridge is a data-movement problem, and the tools for fixing it are exactly the ones we studied — BRAM for on-chip reuse, AXI-Stream for layer-to-layer dataflow, and DMA over an HP port to keep DRAM traffic sequential and burst-friendly.

5. Choosing a Platform for a Given Workload

Work through the filters in order. Each one eliminates candidates; do not start with performance.

1. Does the model fit? Check on-chip and off-chip memory against model size after quantization. This is a hard filter.
2. What is the latency requirement, and is it a deadline or an average? A hard real-time deadline favours deterministic fabric or a dedicated NPU over a Linux-scheduled CPU/GPU.
3. What is the power and thermal budget? Milliwatts points to MCU+NPU; a few watts to FPGA or Edge TPU; tens of watts to Jetson.
4. How standard is the model? Standard INT8 vision CNN → Edge TPU or Jetson. Unusual operators, custom precision, or a non-standard dataflow → FPGA.
5. What volume, and what unit cost? High volume rewards ASIC/NPU economics; low volume and evolving requirements reward reconfigurable hardware.
6. What are the team's skills and the schedule? Toolchain maturity is a real engineering constraint, not a footnote — an FPGA design that ships six months late has lost to a GPU that shipped.
7. Only now: compare achieved performance on your own model, measured, not quoted.

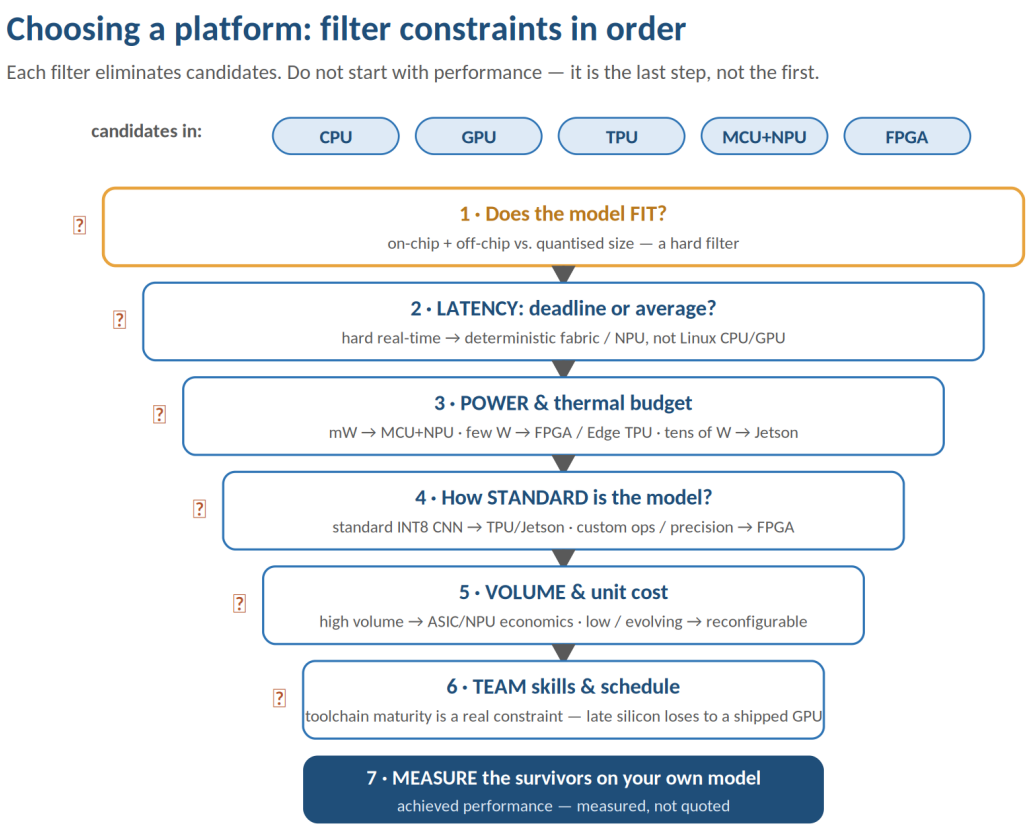

Figure 9. *The platform-selection funnel. Each constraint eliminates candidates in order — fit, latency, power, model standardness, volume, and team/schedule — and only the survivors are measured. Performance is the last filter, not the first.*

5.1 Three short cases

Workload	Reasonable choice	Why
Always-on keyword spotting, coin-cell powered	MCU + Ethos-U55	Milliwatt budget; tiny model; latency easily met
Multi-camera robot with several concurrent networks	Jetson Orin	Needs large models, high throughput, mature stack; watts available
Fixed industrial inspection, hard 10 ms deadline, custom low-bit CNN	FPGA (Zynq)	Deterministic latency, arbitrary precision, dataflow keeps data on-chip

The honest answer to "which platform is best?" is always "for which workload, under which constraints?" — and a defensible answer cites measured numbers on the shortlisted candidates.

6. Measuring It Yourself on the PYNQ-Z2 (Platform Focus)

The PYNQ-Z2 is an unusually good teaching platform for this topic because both sides of the comparison live on one board: the Cortex-A9 PS gives you a software baseline, and the PL gives you the accelerator. Measuring the ratio between them is a controlled experiment, with no cross-vendor confounds.

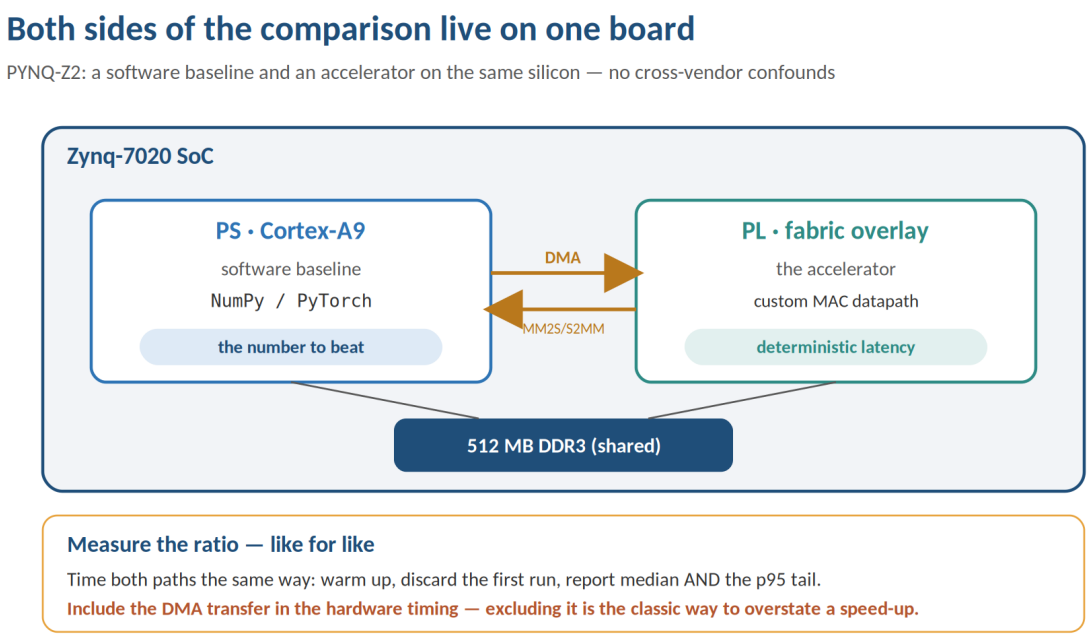

Figure 10. *The PYNQ-Z2 as a controlled experiment: the Cortex-A9 PS gives a software baseline and the PL fabric gives the accelerator, on one shared DDR3. Timing both paths the same way — including the DMA transfer — makes the speed-up honest.*

Timing a software baseline on the PS

Always measure a distribution, discard a warm-up run, and report the median alongside the tail.

```
import time, numpy as np

def benchmark(fn, arg, n=50, warmup=5):
    for _ in range(warmup):
        fn(arg)                      # warm caches, avoid first-call cost
    times = []
    for _ in range(n):
        t0 = time.perf_counter()
        fn(arg)
        times.append(time.perf_counter() - t0)
    times = np.array(times) * 1e3    # milliseconds
    return {"median_ms": np.median(times),
            "p95_ms":    np.percentile(times, 95),
            "fps":       1000.0 / np.median(times)}

print(benchmark(software_infer, test_image))
```

Timing the hardware path

Wrap the accelerator call the same way, so the comparison is like-for-like. Include the data movement — the DMA transfer is part of the latency your application sees, and excluding it is the most common way students accidentally overstate a speed-up.

```
from pynq import Overlay, allocate

ol = Overlay("classifier.bit")
dma = ol.axi_dma

def hw_infer(img):
    in_buf[:] = img.reshape(-1)      # host copy
    dma.sendchannel.transfer(in_buf)  # MM2S
    dma.recvchannel.transfer(out_buf) # S2MM
    dma.sendchannel.wait()
    dma.recvchannel.wait()            # end-to-end, transfer included
    return out_buf
```

```
print(benchmark(hw_infer, test_image))
```

Reading the resource report as your "area" number

After implementation, Vivado reports LUT, FF, BRAM and DSP utilisation. Quote it whenever you quote a speed-up: "3.4× faster than the PS baseline, using 62% of the DSPs and 40% of the BRAM" is an engineering result. "3.4× faster" alone is marketing.

Lab 2 preview. Next session you will run the same small CNN inference on (a) the PS only, using NumPy/PyTorch on the Cortex-A9, and (b) a pre-built PYNQ overlay, then record latency, throughput and — where measurable — energy. You will place both results on the roofline you constructed in Section 4.3 and explain the gap. Expect the accelerated path to disappoint slightly relative to the peak-TOPS ratio: that gap between theory and measurement is the real lesson of this week.

7. Common Pitfalls and Misconceptions

- **Comparing peak TOPS across vendors.** Different precisions, different power boundaries, sometimes sparsity assumed. Compare measured latency on the same model instead.
- **Quoting a mean latency.** Real-time systems fail on the tail. Report median and p95/p99.
- **Excluding data movement from the timing.** If the DMA transfer and host copies are not in the measurement, the speed-up is not real. Time the path your application actually takes.
- **Assuming fewer FLOPs means faster.** Depthwise-separable convolutions cut FLOPs but also cut arithmetic intensity, pushing the layer into the memory-bound region.
- **Reporting speed-up without resource cost.** On FPGA, performance is meaningless without the LUT/BRAM/DSP budget it consumed.
- **Forgetting the model-size filter.** A platform that cannot hold the model does not belong on the shortlist, whatever its TOPS/W.

8. Summary — Key Takeaways

- Peak TOPS is a ceiling, not a promise; achieved performance on your model is the only number that settles an argument.
- Compare on five axes at once: TOPS/W, latency (with its tail), area or FPGA resources, $/TOPS and total system cost, and model-size limits.
- The platform families trade flexibility against efficiency: CPU (most flexible) → GPU → FPGA (custom precision, deterministic) → TPU/NPU (most efficient, most rigid).
- The roofline model separates memory-bound from compute-bound work; the ridge point tells you which side you are on and therefore what optimisation will help.
- For the PYNQ-Z2, a rough roofline (≈ 44 GOP/s over ≈ 2 GB/s, ridge ≈ 22 ops/byte) predicts that low-reuse layers cannot be rescued by adding multipliers — the fix is data movement, exactly the Week 2 toolkit.
- Choose by filtering constraints in order (fit → latency → power → model standardness → cost/volume → team and schedule), then measure the survivors.

9. Self-Assessment Questions

1. Platform A quotes 4 TOPS at 2 W; platform B quotes 40 TOPS at 15 W. Give two reasons why A might deliver lower latency than B on your network, and two reasons why B might win.
2. A fully-connected layer has an arithmetic intensity of about 2 ops/byte. Using the PYNQ-Z2 roofline from Section 4.3, is it compute- or memory-bound? What is the attainable performance, and what would you change first?
3. Explain why a mean latency of 8 ms may still fail a 10 ms hard deadline. What should you have reported instead?

4. Choose a platform for a battery-powered wildlife camera doing occasional species classification, and justify it against each of the five axes.
5. Your accelerated design is 3× faster than the PS baseline but the peak-TOPS ratio suggested 20×. Give three plausible explanations.

References and Resources

All links verified against live sources; PYNQ documentation is the primary reference for the platform-specific material, and vendor pages are cited for the figures quoted in Section 3.

1. PYNQ — Getting Started (board images, overlays, notebooks; basis for the Lab 2 measurements). https://pynq.readthedocs.io/en/latest/getting_started.html

2. PYNQ — PS/PL Interfaces (GP/HP ports, MMIO, allocate, DMA): the data-movement mechanisms behind the roofline discussion in §4.3. https://pynq.readthedocs.io/en/latest/overlay_design_methodology/pspl_interface.html

3. Xilinx Research — FINN: end-to-end quantized neural-network inference on PYNQ-Z1/Z2, the route to low-precision compute on the Zynq fabric. https://github.com/Xilinx/finn

4. AMD — Zynq-7000 SoC Technical Reference Manual (UG585): PS, PL and memory subsystem underlying the PYNQ-Z2 roofline estimate. https://docs.amd.com/r/en-US/ug585-zynq-7000-SoC-TRM

5. Williams, Waterman & Patterson, "Roofline: an insightful visual performance model for multicore architectures", Communications of the ACM 52(4), 2009 — the source of §4. https://dl.acm.org/doi/abs/10.1145/1498765.1498785

6. Coral — Edge TPU performance benchmarks (4 TOPS, 2 TOPS/W, per-model inference times). https://www.coral.ai/docs/edgetpu/benchmarks/

7. NVIDIA — Jetson Orin module family (Orin Nano through AGX Orin: TOPS and power envelopes quoted in §3.2). https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/

8. Arm — Ethos-U55 microNPU product page (area and performance claims in §3.4). https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-u55

9. Arm — Ethos-U Processor Series product brief (PDF): MAC configurations, GOP/s, SRAM and host-CPU options. https://armkeil.blob.core.windows.net/developer/Files/pdf/arm-ethos-u-processor-series-brief-v2.pdf