Quantization

ECBS5200 — Week 5

You diagnosed them. Now compress them.

Where we left off

Week 4 handed you five diagnostic tools. The biggest finding:

Model	Macro F1	ECE (pre-scaling)
Encoder (149M)	0.209	0.062
Decoder (494M)	0.240	0.101

Decoder wins on F1. **Decoder is also more overconfident.**

Size doesn't predict calibration. You measure it.

This week's thesis

> **Quantization is a toolbox, not a technique.**
>
> Each method targets a different constraint — training memory, deployment scale, inference latency, edge hardware. The trade-off depends on your tool, your model, and your hardware.
>
> **Most claims are true somewhere and false elsewhere. Measure on your setup before you ship.**

Today's shape

Lecture → Lab (80 min) → Homework + memo.

Lab: load both Week 3 checkpoints at fp16 / int8 / int4. Measure six configurations. F1, per-tier Δacc, latency, peak VRAM, ECE.

Homework: extend the per-tier and calibration analysis, write a 5-section deployment memo.

Six rows of measurements. One defensible deployment decision.

Vocabulary you'll hear today

Tool names: bitsandbytes, AWQ, GPTQ, FP8, SmoothQuant, GGUF
Numerical formats: fp16, bf16, int8, int4, NF4
Frameworks: vLLM, TensorRT-LLM, SGLang, llama.cpp, TransformerEngine

Each name has a place. By the end of today you will know which place.

Act 1: What quantization actually does

Before we measure anything, look at the operation.

Precision: what a weight costs to store

A weight is a number. **How many bits you spend on it determines its precision AND its storage cost.**

Format	Bits / weight	Range	Your usage
fp32	32	$\pm 3.4 \times 10^{38}$, very precise	original pretraining
fp16	16	±65,504, less precise	your Week 3 training default
bf16	16	same range as fp32	Ampere+ hardware
int8	**8**	**−128 to +127**	**today, Act 1**
int4	**4**	**−8 to +7**	**today, Act 2**

int8 via LLM.int8() — the algorithm we use

Dettmers et al. 2022 (NeurIPS). Each matrix multiply decomposes into:

1. **INT8 fast path** — the bulk of weight columns
2. **FP16 outlier path** — small number of columns with magnitudes that break quantization
3. **Recompose** — sum the two paths

The algorithm powering `BitsAndBytesConfig(load_in_8bit=True)` in today's lab.

Promise	encoder int8 vs fp16	decoder int8 vs fp16	Verdict
Macro F1 preserved	0.2071 vs 0.2095 (Δ −0.0024)	0.2401 vs 0.2399 (Δ +0.0002)	**Kept**
Peak VRAM drops	0.59 GB vs 0.41 GB (+44%)	1.87 GB vs 1.15 GB (+63%)	**Broken**
Latency drops	5.80 vs 2.31 ms/ex (2.5× slower)	12.65 vs 5.05 ms/ex (2.5× slower)	**Broken**

The footnote that matters today

From the runtime discussion in the LLM.int8 paper:

> *"The quantization overhead can slow inference for models with less than 6.7B parameters, as compared to a FP16 baseline."*

Paper's threshold	Your encoder	Your decoder
6,700M	149M	494M
	45× below	14× below

You are firmly in the regime the paper warned about.

int4: four bits per weight

Format	Bits	Values representable
fp16	16	continuous range, ~65k distinct values
int8	8	256 integer levels
int4	4	**16 integer levels**
NF4	4	16 levels **placed non-uniformly** to match normal weight distributions

NF4 + double quant (QLoRA paper) is what `bnb_4bit_quant_type="nf4"` gives you.

bitsandbytes: training memory, not deployment latency

Your Week 3 QLoRA trained through bitsandbytes. It's an excellent **training-memory** tool.

> *"bitsandbytes is the on-ramp tool. It's designed to let you fine-tune models you couldn't otherwise afford. It was never engineered as the path you ship to production."*

Today's lab uses it for inference anyway — so you can **see the trade-offs with your own numbers**.

The three promises of quantization

Quantization is routinely pitched on three promises:

1. **Accuracy preserved** — model quality barely drops
2. **Memory drops** — smaller weights, less VRAM
3. **Latency drops** — less data to move, faster inference

The first promise is usually true on small models. **The other two depend.**

Today you will check all three against your own T4 measurements.

Before the measurement — predict

For each promise you are about to check, on T4 with bitsandbytes, on 149M and 494M models:

- Macro F1 at int8 vs fp16: goes UP, DOWN, or stays the same?
- Peak VRAM at int8 vs fp16: DOWN, UP, or same?
- Latency at int8 vs fp16: DOWN, UP, or same?

Write your predictions down. You'll check them in the lab. The memo rewards specific numbers over folk wisdom.

Act 2: The 2026 production stack

bitsandbytes is the training tool. What's the inference tool?

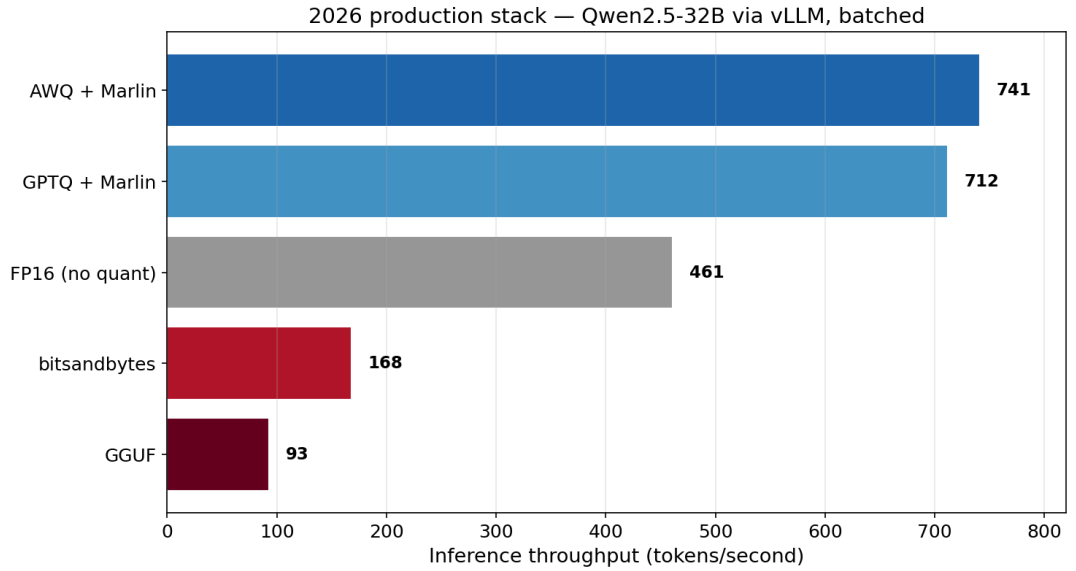
The ecosystem in 2026

Inference throughput, Qwen2.5-32B via vLLM:

Tool	tok/s	×vs bitsandbytes
AWQ + Marlin	741	4.4×
GPTQ + Marlin	712	4.2×
FP16 (no quant)	461	2.7×
bitsandbytes	**168**	**1.0×**
GGUF (llama.cpp)	93	0.6×

These numbers matter for your memo.

Caveat: ratios are at 32B scale on H100. They don't transfer 1:1 to your 149M / 494M models — the *direction* does, the *magnitude* does not.



AWQ — the 2026 default for int4

Lin et al. 2024 (MLSys Best Paper). [Paper: 2306.00978]

Key insight: **~1% of weights are "salient" and deserve protection.** Rank them by activation statistics (not weight magnitude), protect those, aggressively quantize the rest.

Major model families ship pre-quantized AWQ checkpoints on HuggingFace. Supported by vLLM, TensorRT-LLM, SGLang.

This is the paper you'll see cited at work for int4 inference.

GPTQ — AWQ's precursor, still encountered

Frantar et al. 2023 (ICLR). [Paper: 2210.17323]

One-shot second-order post-training quantization. Used to be the state of the art.

In 2026:

- AutoGPTQ wrapper was archived April 2025
- vLLM RFC #39583 proposes deprecating it
- Still encountered in checkpoints, in papers, in older production systems

You should understand it to read the literature. You would not pick it for a new project today.

FP8 — Hopper-era lossless

Kurtic et al. 2024 — *"Give Me BF16 or Give Me Death?"* [2411.02355]

Llama-3.1 family across FP8 / INT8 / INT4 via vLLM:

- **FP8 W8A8 is effectively lossless across all model scales.**
- Requires modern FP8-capable hardware. T4 does not support this path.
- Not available on T4.

FP8 is the format you'll see Nvidia actively pushing for modern inference.

SmoothQuant — weights AND activations

Xiao et al. 2023 (ICML). [2211.10438]

bitsandbytes and AWQ are weight-only quantization. **SmoothQuant also quantizes activations** — W8A8.

Rescales activations and weights via an offline mathematical equivalence.

On Turing/T4 (where FP8 is unavailable) this is the mainstream W8A8 path in TRT-LLM and ONNX Runtime.

GGUF — CPU and edge inference

Format used by **llama.cpp** and its ecosystem.

Not a quantization algorithm per se — a **container format** that supports multiple quantization schemes (Q4_K_M, Q5_K_S, etc.) optimized for CPU inference.

Production use case: run quantized LLMs on laptops, phones, Raspberry Pis, anywhere without a GPU.

If your constraint is "no GPU," GGUF is the tool.

The toolbox, organized

Constraint	Tool	Where it lives
Training memory (QLoRA)	**bitsandbytes NF4**	HuggingFace, PEFT
Production int4 inference	**AWQ**	vLLM, TRT-LLM, SGLang
Modern-hardware inference	**FP8**	TransformerEngine on H100+
Legacy / research	**GPTQ**	Older checkpoints
W8A8 on Turing	**SmoothQuant**	TRT-LLM, ONNX Runtime
No GPU	**GGUF**	llama.cpp

Today's lab uses row one. At work you'd touch all six.

Act 3: What we'll measure

The frame. Then the results.

Six configurations

Two models × three precisions:

- encoder fp16, encoder int8, encoder int4
- decoder fp16, decoder int8, decoder int4

For each configuration, five measurements:

1. **Macro F1** — quality
2. **Peak VRAM** — memory footprint at inference
3. **Latency** — milliseconds per example, batched
4. **Per-tier Δacc** — where damage lands (head / mid / tail)
5. **ECE** — calibration, pre and post temperature-scaling

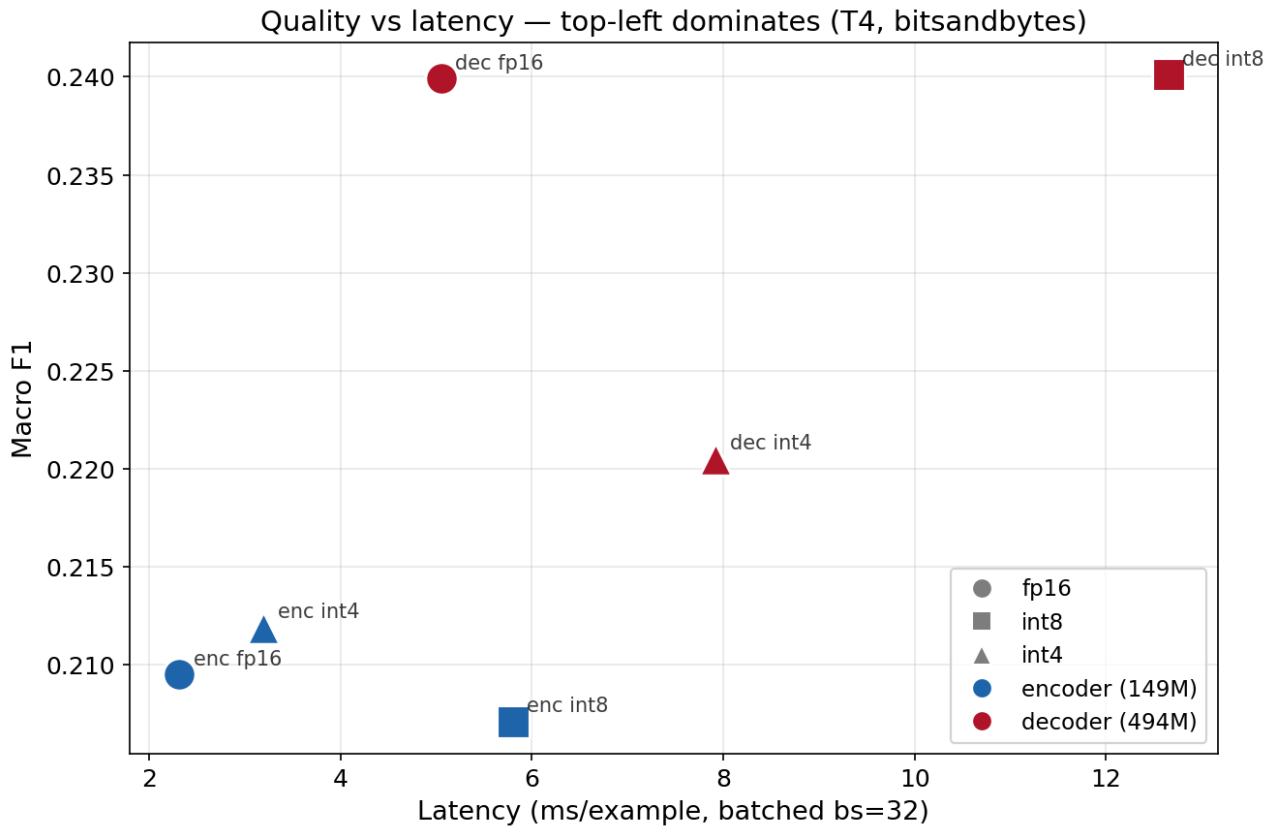
The Pareto frame

Six points on two axes.

Pareto-dominated: if another point is better on BOTH axes, you'd never ship this one.

Pareto-frontier: points where no other point dominates them on every axis you care about.

The deployment choice lives on the frontier.



The per-tier frame

Aggregate macro F1 averages over 113 classes. **The average can hide structure.**

Tiers by training frequency:

- **Head** — top 20 classes (most training data)
- **Mid** — next 40 classes
- **Tail** — bottom 53 classes (least training data, fewer val examples)

Per-tier Δaccuracy shows you whether compression damage is **uniform** across tiers or **concentrated** somewhere.

The calibration frame

A model can be 57% accurate AND be catastrophically overconfident about it.

ECE = expected gap between stated confidence and empirical accuracy.

Temperature scaling = one-parameter fix: divide all logits by a scalar T before softmax.

Quantization can shift calibration. You measure ECE before and after T-scaling at each precision.

The deployment frame

Real deployment is not "which model is best." It's:

> **"Given constraints X Y Z, which configuration satisfies all of them?"**

Today's hypothetical:

- Single T4 GPU
- 100 requests/second sustained → ≈10 ms/ex batched
- Macro F1 floor 0.20
- Post-scaling ECE ceiling 0.05

Six configurations, four constraints. **You'll check each against each.**

Act 4: What the numbers show

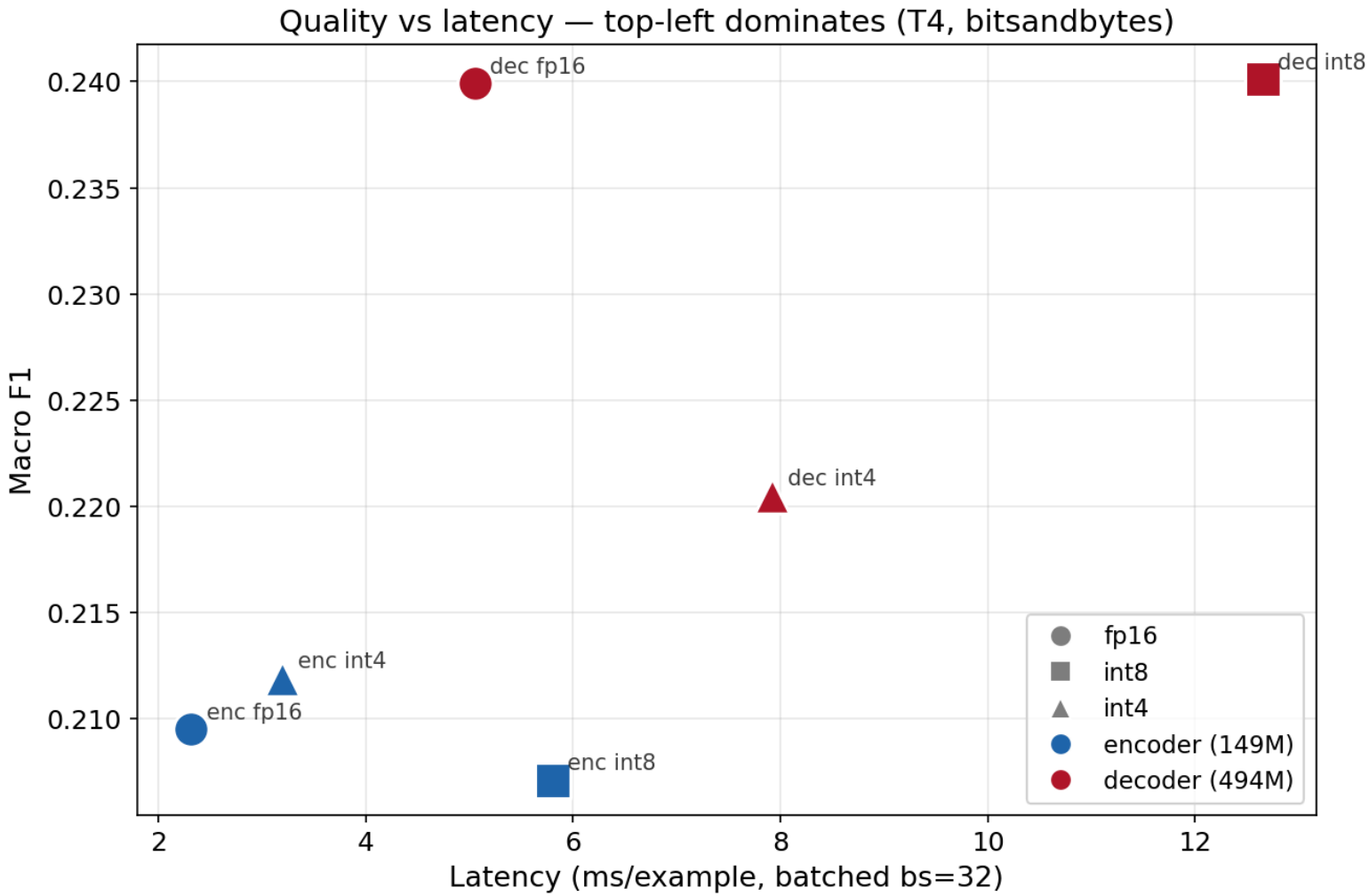
Now the results. Your lab will reproduce these.

The six-row summary

Model	Precision	Macro F1	Latency ms/ex	VRAM GB	ECE post
encoder	fp16	0.210	2.31	0.41	0.040
encoder	int8	0.207	5.80	0.59	0.039
encoder	int4	0.212	3.19	0.62	0.027
decoder	fp16	0.240	5.05	1.15	0.069
decoder	int8	0.240	12.65	1.87	0.072
decoder	int4	0.221	7.92	1.89	0.070

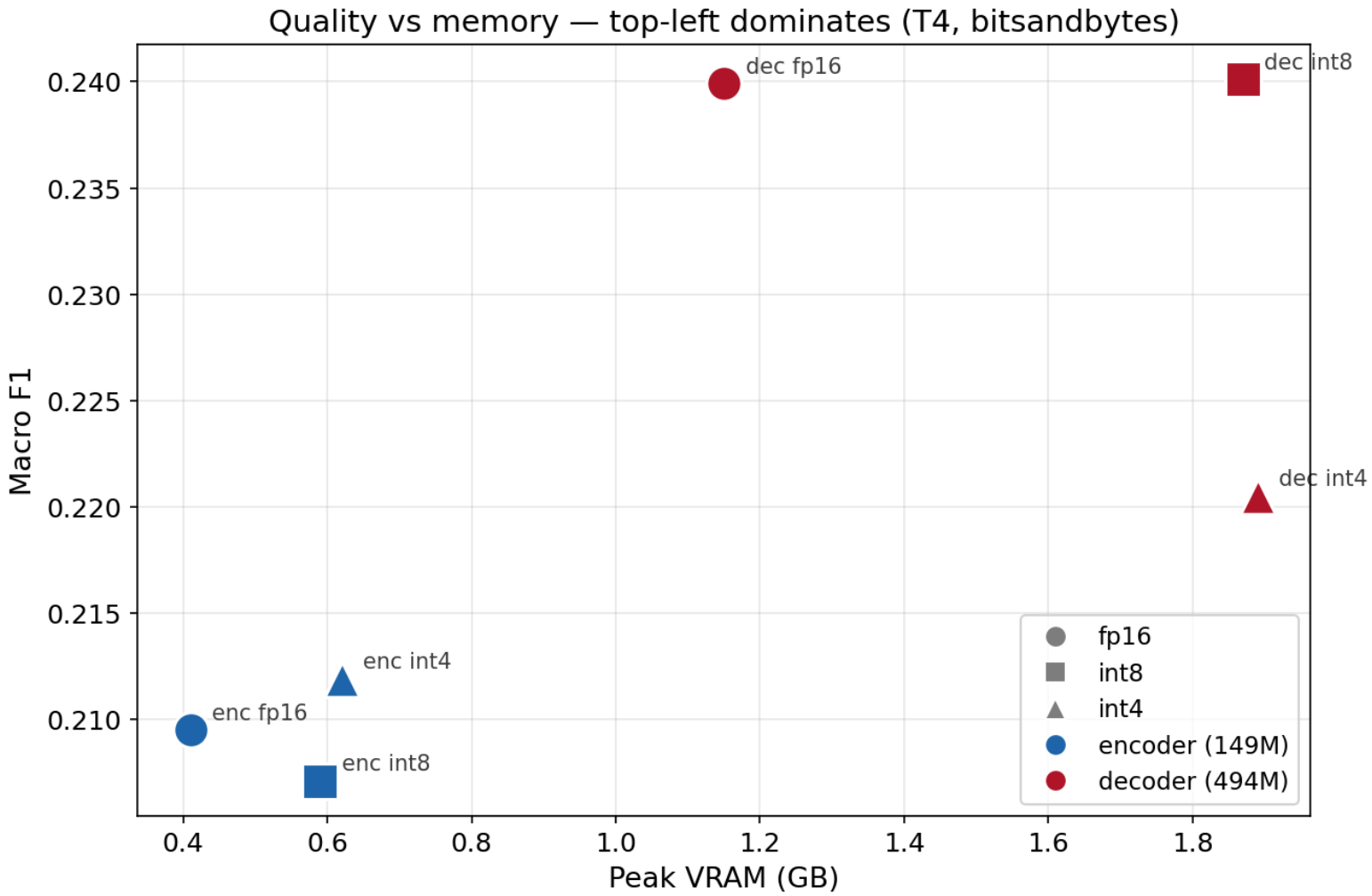
Six configs. Five columns of evidence per config. Everything else today reads this table.

Pareto 1: quality vs latency



Look at which precision is fastest on each model.

Pareto 2: quality vs memory



Look at which precision uses the least memory.

The three promises — checked

On your T4 with bitsandbytes, int8 vs fp16:

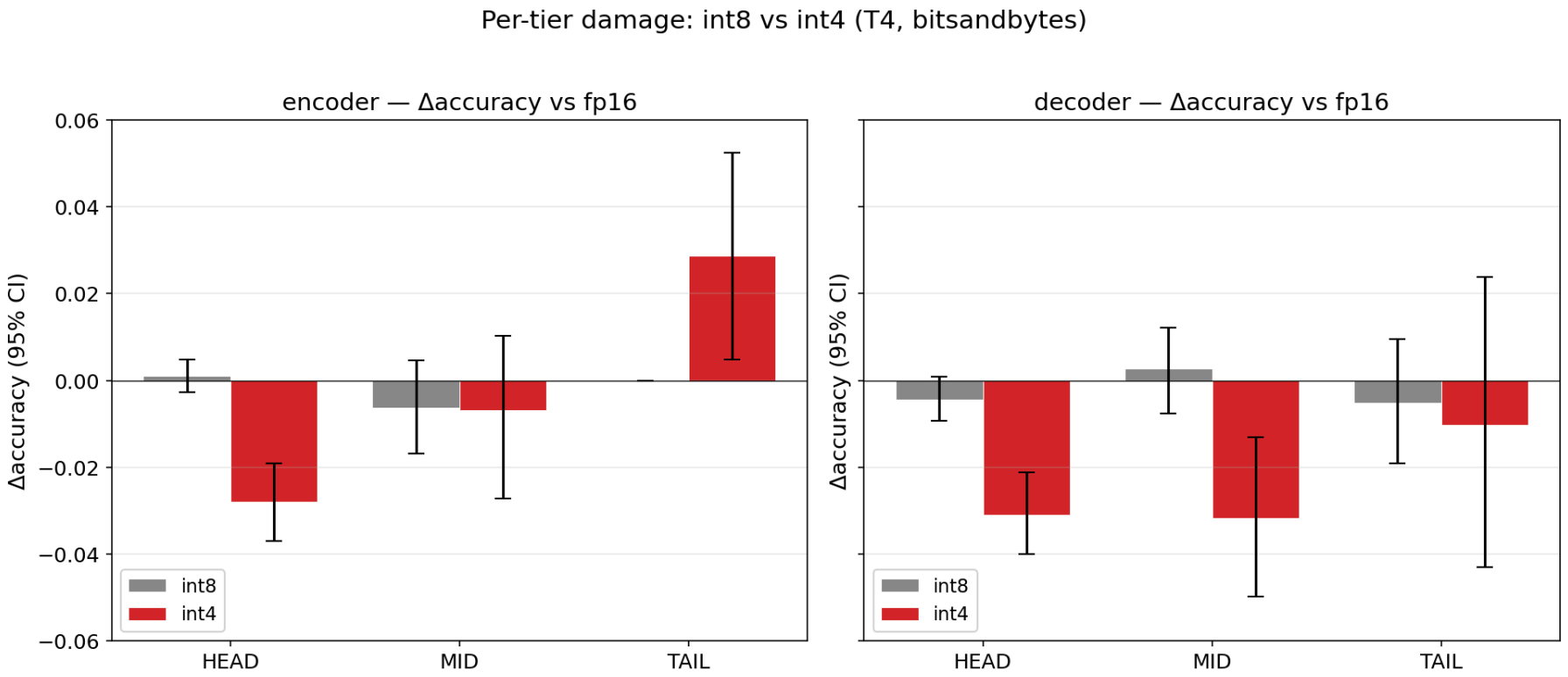
1. **Macro F1 preserved** → kept
2. **Peak VRAM drops** → **reversed** (+44%-63%)
3. **Latency drops** → **reversed** (2.5× slower)

Two of three promises broken at this scale on this tool.

The lesson of the week, empirically.

Promise	encoder int8 vs fp16	decoder int8 vs fp16	Verdict
Macro F1 preserved	0.2071 vs 0.2095 (Δ −0.0024)	0.2401 vs 0.2399 (Δ +0.0002)	**Kept**
Peak VRAM drops	0.59 GB vs 0.41 GB (+44%)	1.87 GB vs 1.15 GB (+63%)	**Broken**
Latency drops	5.80 vs 2.31 ms/ex (2.5× slower)	12.65 vs 5.05 ms/ex (2.5× slower)	**Broken**

Per-tier Δacc — where damage lands



int8 bars: nothing statistically distinguishable.

int4 bars: head and mid take real damage on the decoder.

A real CI (excluding zero) needs BOTH narrow CI AND enough flipped predictions to survive a reshuffle — we'll come back to that.

The bootstrap caveat

The encoder tail at int4 has CI [+0.005, +0.053] — it excludes zero.

But: 210 val examples, 53 classes. A 3pp Δacc on that tier ≈ 6 flipped predictions.

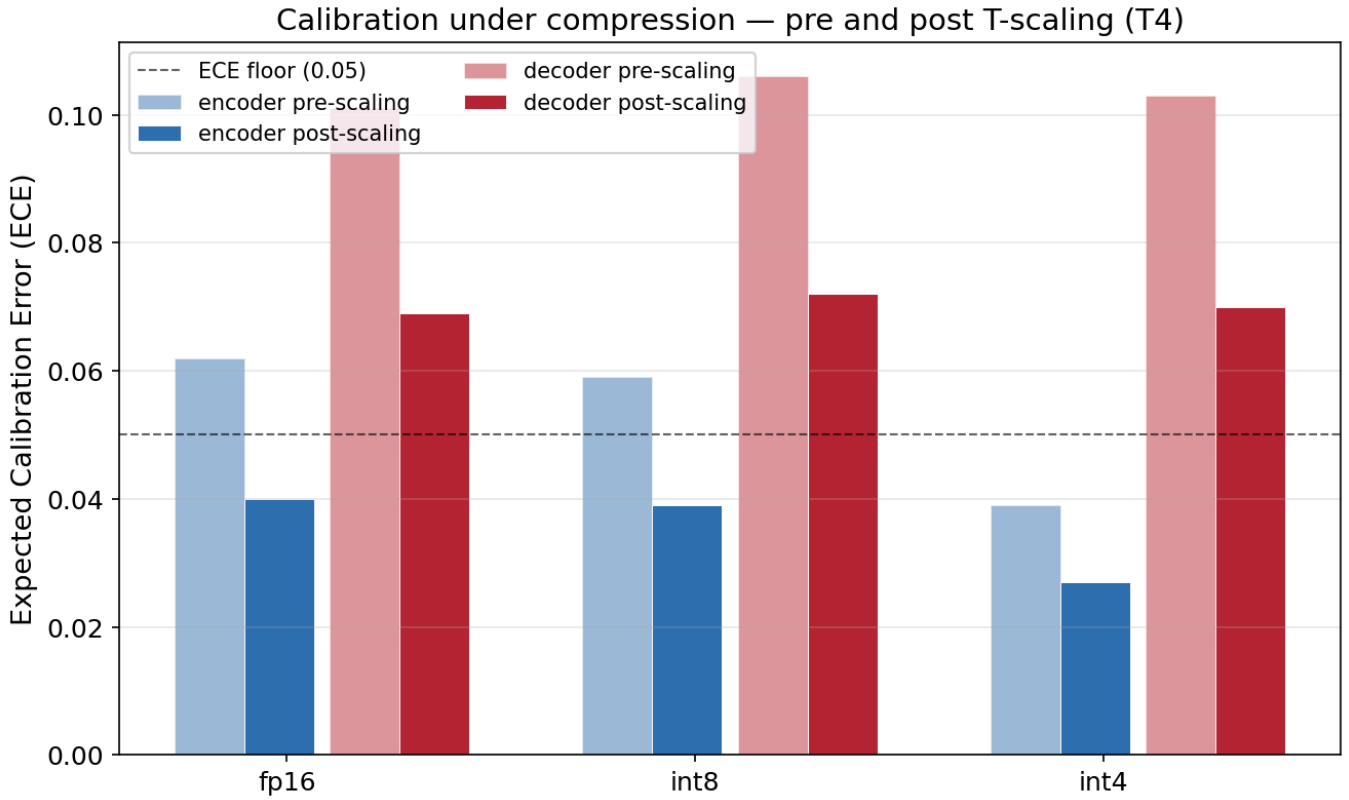
A bootstrap CI measures how these 6 flips would vary under **resampling this val set**. It does NOT tell you the effect would survive under a **different stratified split**.

Real effect = narrow CI **AND** enough flipped predictions. One criterion without the other is directional, not proof.

Act 5: Calibration under compression

ECE is orthogonal to accuracy. Quantization can move it.

ECE under compression



Six configs, pre- and post-T-scaling.

Decoder is more miscalibrated than encoder at every precision.

Temperature scaling helps both, but the decoder bottoms out around 0.07.

Does T-scaling generalize?

Homework exercise: **fit T on half of val, evaluate on the other half.**

If T from half A drops ECE on half B by the same amount it dropped on the full val set, **the fit generalized.**

If not, T is partially memorizing the fit set.

For the encoder at int4, the generalization is weaker than at fp16. That's a memo-worthy observation.

The deployment implication

If your deployment gates on **confidence thresholds** (e.g., route below 0.8 confidence to human review):

- A miscalibrated model routes the wrong examples to review
- Temperature scaling **must be re-fit on quantized logits**, not reused from fp16
- If T-scaling only partially recovers, the threshold itself shifts

This is why calibration lives in the rubric — it changes your deployment recipe.

Act 6: Deployment reasoning

Six configs. Four constraints. One recommendation.

The constraint envelope

Constraint	Threshold	Rationale
Hardware	Single T4	Fixed budget
Throughput	100 req/s → ≤10 ms/ex	Product SLA
Quality	macro F1 ≥ 0.20	Floor for useful triage
Calibration	post-scaling ECE ≤ 0.05	Confidence-gating viability

Each constraint binds or does not. **You check each config against each constraint.**

Which configs pass?

Config	Latency	F1	ECE	All 4?
encoder fp16	✅	✅	✅	✅
encoder int8	✅	✅	✅	✅
encoder int4	✅	✅	✅	✅
decoder fp16	✅	✅	✗	✗
decoder int8	✗	✅	✗	✗
decoder int4	✅	✅	✗	✗

Three viable configs. All encoder. The decoder fails on calibration at every precision.

Within the viable set

Three encoder configurations meet the envelope. Which do you pick?

	encoder fp16	encoder int8	encoder int4
Macro F1	0.210	0.207	0.212
Latency ms/ex	**2.31**	5.80	3.19
VRAM GB	**0.41**	0.59	0.62

F1 differences are within measurement noise. **fp16 dominates on latency AND memory.**

→ **Encoder fp16.**

Hardware-dependent claims

Your answer is **stack-specific.** If we changed the setup:

Change	What would move
Move to H100 + vLLM + AWQ	int4 path becomes fastest AND smallest (Lin 2024 benchmarks)
Raise ECE ceiling to 0.08	Decoder configs become viable; dec fp16 wins F1
Move to L4 or A10	dec int8 passes latency; decoder viable if ECE relaxes too
Move to on-device (no GPU)	GGUF and quantization become mandatory

Memo section 4 rewards naming this explicitly.

Week 6 preview: another compression tool

You compressed **within a family** (same model, fewer bits per weight).

Next week: compress the family. Take the bigger model and train a smaller one to match its outputs.

Distillation is another row in the quantization toolbox — **same measurement discipline, different mechanism.**

Lab and homework — logistics

Lab (80 min, T4): `week5_lab.ipynb`. Six quantized-inference runs. ~25 min compute + ~55 min reading and interpretation.

Homework (~5 hours): `week5_homework.ipynb`. Attach your lab notebook's outputs via `Add Input → Your Work`. No separate dataset upload needed.

Memo (100 points): embedded in the homework notebook. Rubric: `assessments/week5_memo_rubric.md`.

Due: Wednesday morning before Week 6 class.

Today in one sentence

Quantization is not a thing you do.

It's a cabinet of tools you choose from, measure on your setup, and defend against constraints.

If you have time…

I tried to live-load a **14B model at int4** on a **single T4**. Three slides. ~3 minutes.

The toolbox thesis — and a ridiculous Kaggle story about disk space.

I tried to give you a wow moment

Plan: live-load a **14B model at int4** on a **single T4** in front of you. The fp16 model is bigger than the GPU. Quantization makes it fit. Magic.

I couldn't even download it.

Qwen2.5-14B fp16 weights	**29.4 GB**
Kaggle `/kaggle/working` quota	**20.9 GB**
Outcome	`OSError: No space left on device`

Pre-quantized variants exist for exactly this

`unsloth/Qwen2.5-14B-Instruct-bnb-4bit` — same model, weights stored as int4 on disk. Download: **~10 GB**. Fits.

	Single T4
Peak VRAM after load	**9.93 GB / 15.64 GB**
Headroom	**5.70 GB**
Load time	12 s

> *"Category: Fee Dispute. The complaint falls under the 'Fee Dispute' category because it involves a disagreement over an unexpected late fee charged despite timely payment."*

— actual generated output, batch 1, on a T4

But you wouldn't ship this

Generation throughput: 6.3 tokens/second.

A user typing into a chat box would feel the lag. That's not a product.

	tok/s
14B via bitsandbytes int4 (this demo)	**6.3**
32B via AWQ + Marlin on H100 (Act 2)	741

Different model, different hardware, different regime — qualitative gap, not a measured ratio.

bitsandbytes **fits** the model. The production stack **runs** it.

Toolbox, not technique.