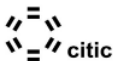


MICROCREDENCIAL ALGORITMOS VERDES PARA LA IA DISEÑO E IMPLEMENTACIÓN

Tema: Natural language processing

Sesión 2: Quantization

Ponente: Olga Zamaraeva



UNIVERSIDADE DA CORUÑA



Financiado por la
Unión Europea
NextGenerationEU



GOBIERNO
DE ESPAÑA

MINISTERIO
PARA LA TRANSFORMACIÓN DIGITAL
Y DE LA FUNCIÓN PÚBLICA



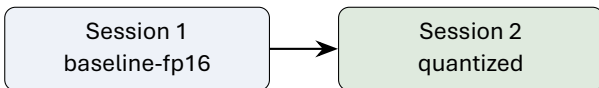
2026

TODAY'S STRUCTURE

- Lecture block 1: what precision means, and what quantization does to numbers
- Notebook break: quantization playground, no GPU needed
- Lecture block 2: why quantization helps, when it fails, and what GPTQ/GGUF/QAT mean
- Notebook block: rerun the Session 1 benchmark with a quantized model
- Final comparison: quality vs. energy vs. carbon vs. time

WHERE WE LEFT OFF

- Session 1: one QA pipeline, measured once
- Output: `config=baseline-fp16` in `leaderboard.csv`
- Same eight documents, same eight questions, same scoring rule today
- Today's change: lower numerical precision



HOW TO READ TODAY'S LEADERBOARD

- **Quality:** how many questions the model answers correctly
- **Energy:** kWh measured by CodeCarbon
- **Carbon:** estimated CO₂eq from that energy use
- **Time:** wall-clock time for the run
- **Memory:** peak GPU memory, when available

The comparison

A useful quantized run should be cheaper or faster, while staying close enough in quality for the task.

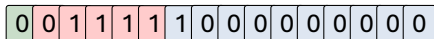
HOW TO REPRESENT NUMERICAL VALUES

- Reminder: binary digits are powers of two, the same idea as decimal digits being powers of ten —
 $1101_2 = 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 13$
- Floats are represented by bits, per the IEEE-754 standard
- Three parts: sign, exponent, fraction or mantissa
- “32-bit” and “16-bit” mean how many bits we spend per number



sign exponent fraction / mantissa

A WORKED EXAMPLE: DECODING ONE FP16 NUMBER



- Sign: 0 $\rightarrow$ positive
- Exponent bits 01111: read as a plain binary number, that's $8 + 4 + 2 + 1 = 15$
- FP16 subtracts a fixed **bias** of 15 from that, so the actual power of two is $15 - 15 = 0$
- Mantissa bits 1000000000: read as fractions after an implied leading 1 — first bit is worth $\frac{1}{2}$, so this is $1 + \frac{1}{2} = 1.5$
- Combine: $(+1) \times 1.5 \times 2^0 = 1.5$

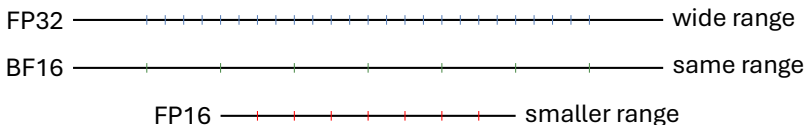
MORE BITS = MORE PRECISION, MORE RANGE

- FP32: 1 sign, 8 exponent, 23 mantissa bits
- FP16: 1 sign, 5 exponent, 10 mantissa bits
- Fewer mantissa bits $\rightarrow$ coarser values
- Fewer exponent bits $\rightarrow$ smaller representable range
- Too few bits $\rightarrow$ overflow to ∞ or underflow to 0.0

Format	Sign	Exponent	Mantissa
FP32	1	8	23
FP16	1	5	10
BF16	1	8	7

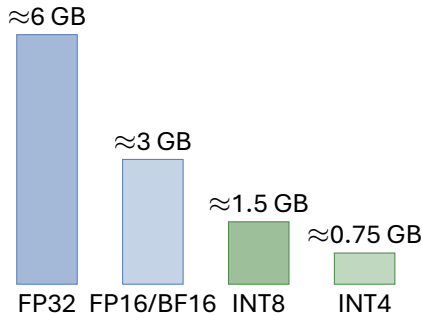
DYNAMIC RANGE VS. PRECISION: BF16

- **Dynamic range:** the interval of representable numbers
- **Precision:** the distance between neighboring representable values
- BF16 keeps FP32's exponent bits, so it keeps FP32's range
- BF16 has fewer mantissa bits, so nearby values are represented more coarsely
- Actual ranges: FP32 and BF16 both span roughly $\pm 1.2 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$; FP16 spans roughly $\pm 6.1 \times 10^{-5}$ to ± 65504



MEMORY USAGE: OUR OWN MODEL, IN BITS

- $\text{Memory} = (\text{bits} \div 8) \times \text{number of parameters}$
- Qwen2.5-1.5B-Instruct: 1.5 billion parameters



ACTIVITY: QUICK MEMORY ESTIMATES

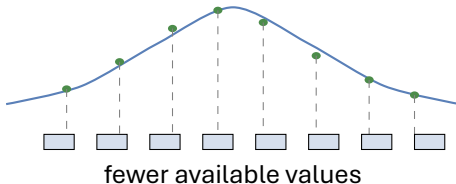
Memory calculation, 8–10 min

Estimate the weight memory for a 7B, a 32B, and a 70B-parameter model, each in FP16, INT8, and INT4.

- Use: $\text{memory} = \text{parameters} \times \text{bits} \div 8$
- Ignore activations and framework overhead for now
- Check whether each result would fit into 16 GB of GPU memory
- Which sizes need quantization just to fit at all, not just to run faster?

QUANTIZATION: THE BASIC IDEA

- Quantization maps high-precision values to a lower-precision format
- Many original values have to share fewer available codes
- The expected tradeoff is smaller/faster computation versus loss of precision



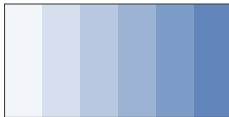
AN ANALOGY: FEWER COLORS

- Reducing colors in an image is also a lossy mapping
- The image becomes grainier, but may remain recognizable
- Quantization is similar: the model becomes coarser, but may remain useful

Original: smooth gradient

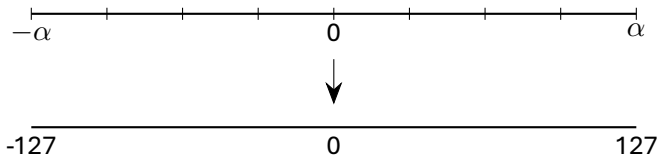


Quantized: 6 visible bands



SYMMETRIC ABSMAX QUANTIZATION

- Symmetric: map a range centered on zero, $[-\alpha, \alpha]$, onto the integer range
- Absmax: choose α as the largest absolute value in the data
- Zero in FP32 maps to zero in INT8



ABSMAX: THE FORMULA

$$s = \frac{127}{\alpha}$$

$$x_q = \text{round}(s \cdot x)$$

$$x_{deq} = \frac{x_q}{s}$$

- α : highest absolute value in the data
- s : scale factor
- Quantize by multiplying and rounding; dequantize by dividing back

ABSMAX: A WORKED EXAMPLE

- Suppose the largest absolute value is $\alpha = 10.8$
- Target: INT8, so $s = 127/10.8 \approx 11.76$
- For $x = 3.08$: $3.08 \times 11.76 \approx 36.2$, so $x_q = 36$
- Dequantized value: $36/11.76 \approx 3.06$

Consequence

Different original values can become the same quantized value.

DEQUANTIZATION AND QUANTIZATION ERROR

- Dequantization recovers an approximate value, not the original value
- Quantization error = original value — dequantized value
- Fewer bits usually means larger gaps between representable values

Original	Quantized	Dequantized	Error
3.08	36	3.06	0.02
3.02	36	3.06	-0.04
-1.85	-22	-1.87	0.02

ACTIVITY: QUANTIZATION PLAYGROUND A

Notebook 02b, Part A, 15–20 min

Implement absmax quantization and dequantization from scratch.

- Run the toy example from the slides
- Check which values collide onto the same integer
- Sweep over 8, 4, 3, and 2 bits
- Record what happens to mean absolute error

WHAT FEWER BITS ACTUALLY CHANGE

- INT8 gives roughly 256 possible codes
- INT4 gives 16 possible codes
- INT3 gives 8 possible codes
- INT2 gives 4 possible codes

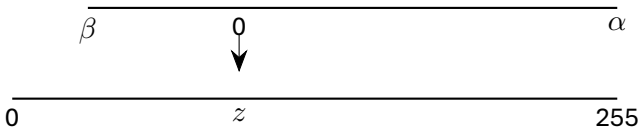
INT4 

INT3 

INT2 

ASYMMETRIC ZERO-POINT QUANTIZATION

- Symmetric quantization centers the range on zero
- Asymmetric quantization maps the actual minimum and maximum onto the integer range
- It needs a zero-point z , because FP32 zero no longer necessarily maps to integer zero
- Useful when the data is lopsided, for example activations after ReLU



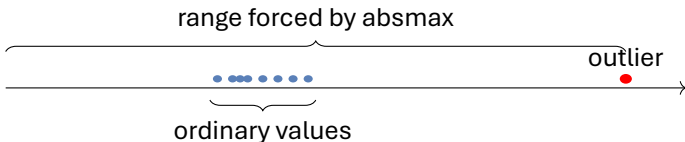
SYMMETRIC VS. ASYMMETRIC

Method	Range	Useful when
Symmetric	$[-\alpha, \alpha]$	values roughly balanced around zero
Asymmetric	$[\beta, \alpha]$	values are offset or one-sided

- Symmetric is simpler: one scale, zero stays zero
- Asymmetric fits lopsided data better: scale plus zero-point
- Neither is best in the abstract; it depends on the tensor

THE OUTLIER PROBLEM

- Absmax uses the largest absolute value to set the range
- One extreme value can stretch the range for everyone
- Ordinary values then get squeezed into too few integer buckets
- Precision is lost where most values actually are



CLIPPING AND CALIBRATION

- **Clipping:** choose a smaller range and clamp values outside it
- Ordinary values get better precision
- Clipped outliers get worse
- **Calibration:** choosing the range well

Calibration option	What it tries to do
Percentile	ignore rare extremes
Mean-squared error	minimize numerical reconstruction error
KL-divergence	preserve distribution shape

ACTIVITY: OUTLIERS AND CLIPPING

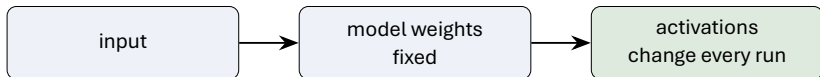
Notebook 02b, Parts B–C, 20–30 min

Use a toy vector with one large outlier and compare absmax with clipping.

- Count how many distinct integer codes the ordinary values use
- Compare reconstruction error with and without clipping
- Try at least two clipping thresholds
- Decide which threshold is best for ordinary values, and which is best overall

WEIGHTS ARE STATIC; ACTIVATIONS ARE DYNAMIC

- Weights are known after training and do not depend on the input
- Activations are computed fresh for each input
- Therefore activations can have different ranges from one example to another



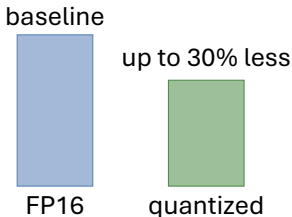
DYNAMIC AND STATIC POST-TRAINING QUANTIZATION

Strategy	Scale/zero-point	Tradeoff
Dynamic PTQ	recomputed during inference	more adaptive, slower
Static PTQ	fixed from calibration data	faster, depends on calibration data

- PTQ = post-training quantization
- Dynamic is safer when activation ranges vary a lot
- Static is faster when the calibration data is representative

REVEAL 1: THE SAVINGS ARE REAL

- Measurement studies generally find real savings from quantization
- Example: up to about 30% less energy per query in one recent study
- Some settings also show much lower latency



REVEAL 2: QUALITY CAN BREAK

- Aggressive quantization can damage task performance
- In one coding benchmark, a 13B model lost up to 92% of pass rate under aggressive quantization
- Smaller models can be more vulnerable than larger ones
- Our model today is 1.5B, so this is relevant

Source: Shi & Ding, 2025

REVEAL 3: HARDWARE MATTERS

- Quantization helps most when the hardware supports low-precision arithmetic efficiently
- 8-bit and 4-bit can reduce energy substantially on suitable hardware
- Too-aggressive bit widths, or unsupported formats, can make energy worse

Practical rule

Fewer bits only help if the software and hardware can actually use them efficiently.

Source: Husom et al., 2025

LOWER PRECISION IS OFTEN LITERALLY FASTER

- Memory is one reason to quantize: fewer bits, less VRAM
- Speed is another reason: many GPUs have low-precision tensor cores
- Same chip, same clock cycle, more low-precision operations can fit

Precision on NVIDIA T4	Peak throughput
FP32	8.1 TFLOPS
FP16	65 TFLOPS
INT8	130 TOPS
INT4	260 TOPS

WHAT COUNTS AS THE SAME MODEL?

- The architecture can stay the same: same layers, same attention blocks, same tokenizer
- The numerical representation of the weights changes
- The model may also use different kernels or loading code at inference time
- For benchmarking, record the exact checkpoint and loading path

For our leaderboard

`baseline-fp16`, `gptq-int4`, and `bnb-int8` are different configurations, not different tasks.

GPTQ: ROUND THE IMPORTANT NUMBERS CAREFULLY

- Squeezing numbers into fewer bits always loses some precision — the real question is *where* you lose it
- GPTQ finds the numbers that matter most to the answer, and rounds those the most carefully; it is freer with the rest

Item	Price	Rounded to the nearest \$10
Laptop	\$997	\$1000 — kept close
Headphones	\$203	\$200 — kept close
Gum	\$0.79	\$0 — rounded away
Mints	\$1.20	\$0 — rounded away

- The total still stays close to correct, because the *big* numbers were rounded carefully — being sloppy with the small ones barely matters

GPTQ, IN THE MODEL: THE SAME IDEA, APPLIED TO WEIGHTS

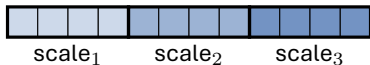
A model layer has thousands of numbers (weights) instead of four grocery items, but GPTQ treats them exactly the same way: find the ones that matter, protect those.

Weight	Value	How GPTQ treats it
w_{17}	0.83	kept precise — output depends on this a lot
w_{42}	0.61	kept precise — output depends on this a lot
w_5	-0.02	rounded loosely — barely changes the output
w_{88}	0.04	rounded loosely — barely changes the output

- “Matters to the output” is not the same as “big number” — GPTQ measures this directly, rather than guessing from the value alone
- We use an already-calibrated GPTQ checkpoint — no calibration run today

GGUF: LET NEARBY NUMBERS SHARE A SCALE, BUT ONLY NEARBY ONES

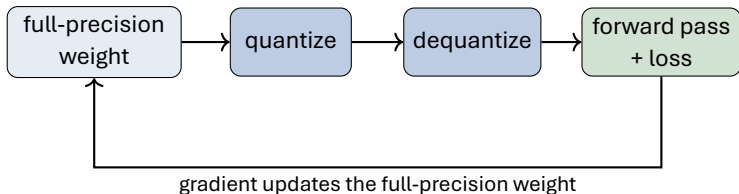
- Imagine fitting 50,000 people in a stadium into *one* t-shirt size — most people get a bad fit
- Now split the crowd into small sections; each section picks its own size — much better fit, still far cheaper than tailoring for every single person
- GGUF does this with weights: instead of one scale for a huge chunk of the model, small blocks of weights each get their own scale



- Popular for running models locally — laptops, CPU/GPU mixing; not what we run today

QAT: PRACTICE UNDER REAL CONDITIONS

- PTQ: train normally, then quantize at the end — like training in a calm pool and hoping race day's waves do not throw you off
- QAT: train *while* simulating the quantization noise the model will face later — like practicing in the actual rough water ahead of time



- Usually more accurate at low precision, but needs training time

BITNET: A 3-POSITION SWITCH INSTEAD OF A DIMMER

- Reminder: a neuron combines several input numbers — raw data, or signals from the previous layer — into one output, as a *weighted sum*: each input times its own weight (how much it counts, sign included), all added together

inputs: $[2, 5, 3]$

Weight type	Weights	Result
Normal (dimmer)	$[0.4, -0.1, 0.9]$	$0.8 - 0.5 + 2.7 = 3.0$
BitNet (switch)	$[+1, -1, 0]$	$2 - 5 + 0 = -3$

- Multiplying by $+1$ is just using the input as-is; by -1 , flipping its sign; by 0 , ignoring it — the weighted sum still works, no real multiplication needed
- Not our method today, but a real research direction

BITNET: WHY DOES THIS ACTUALLY WORK?

3 weights, each $\in \{-1, 0, +1\}$: only 7 possible totals



12 weights, same 3 values each: 25 possible totals



- Coarse individual weights, combined in large numbers, still cover a fine-grained range — real layers have thousands of them
- Trained for it from the start, like QAT — never adjusted afterward
- The payoff: add/subtract cost far less energy than multiply, in hardware

WHICH TECHNIQUE, WHEN

Technique	Use case
Dynamic PTQ	most adaptive post-training option, extra work during inference
Static PTQ	faster inference, needs representative calibration data
GPTQ	strong option below 8 bits without retraining
GGUF	local CPU/GPU mixed inference
QAT	best low-precision accuracy when retraining is possible
BitNet	architectural route to very low-precision compute

ACTIVITY: IS QUANTIZATION A FREE LUNCH?

Discussion, 10–15 min

Consider when quantization is not worth it.

- Wrong hardware or unsupported kernels
- Too few bits for the task
- Small model already near its quality limit
- Calibration data unlike deployment data
- Quality metric too weak to detect the failure

WHAT WE ARE ACTUALLY RUNNING TODAY

- Primary path: Qwen/Qwen2.5-1.5B-Instruct-GPTQ-Int4
- Fallback path: load_in_8bit with bitsandbytes
- Same QA corpus and questions as Session 1
- Same MT passage as Session 1
- CodeCarbon measures the run again

NOTEBOOK WORKFLOW

1. Upload or recreate the Session 1 leaderboard
2. Load the quantized model, or the fallback model
3. Run the same eight QA questions
4. Score answers with the same keyword rule
5. Append a new row to `leaderboard.csv`
6. Translate the Galician passage again
7. Append a row to `mt_leaderboard.csv`

WHAT TO COMPARE AT THE END

- Did QA accuracy change?
- Did energy or CO₂eq change?
- Did runtime change?
- Did peak memory change?
- Did the MT output change in a linguistically meaningful way?
- Was the quantized path actually GPTQ Int4, or the fallback Int8 path?

WRAP-UP

- Quantization reduces the number of bits used to represent model values
- The main benefit is lower memory, often lower latency and energy
- The main risk is quality loss, especially at very low precision
- Calibration, hardware support, and task choice determine the outcome
- Today's benchmark adds one more comparable row to the course leaderboard