Language Model Quantization II
Quantizing Matrices

Amin Saberi

MS&E 319

Autumn 2026



Three Things to Quantize in an LLM

Weights. The matrices of the model: fixed after training, known before deployment, the same for every input.

Activations. The vectors computed inside the model: new for each input, rounded as the model runs.

Cached keys and values. Stored for every past token so that attention need not recompute them; one more per token.

Today's lecture: quantizing weights and activations.



Part I:
Quantizing Weights



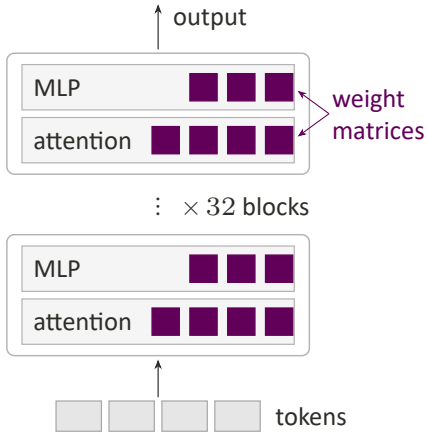
Why Quantize Weights?

Size. Llama 3.1 8B, all weights:

16 bits 16.1 GB
4 bits 4.0 GB

Hardware. 4-bit weights fit in 8 GB of memory; 16-bit weights do not.

Speed. A question of hardware: we return to it later.





What Counts as a Good Quantization?

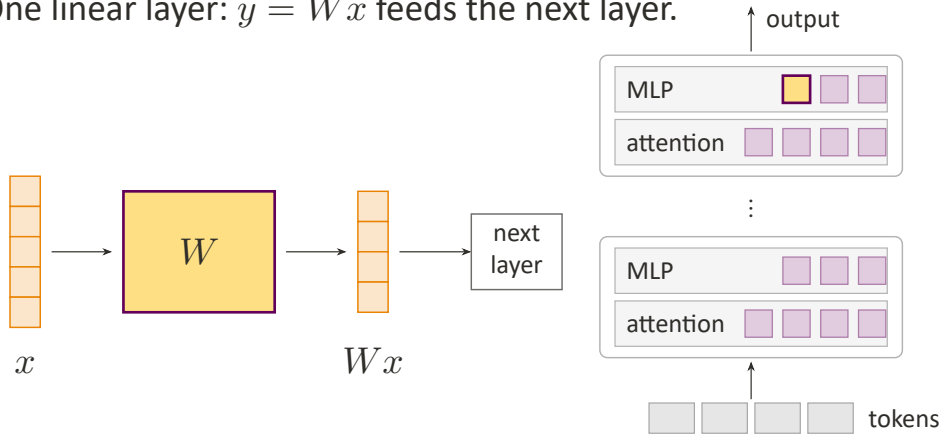
Level	Question
Task performance	Does the model still solve the task?
Prediction loss	Does it still predict the next token well?
Layer output error	Does each layer act similarly on its inputs?
Weight error	Are the stored entries close?

Efficiency, measured separately: actual memory, latency, throughput.



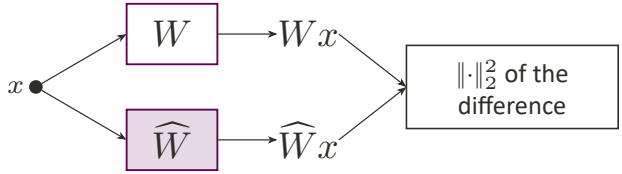
The Weight Matrix

One linear layer: $y = Wx$ feeds the next layer.



Three Errors for the Weight Matrix

$W \in \mathbb{R}^{m \times d}$, $x \in \mathbb{R}^d$, $y = Wx$, and $E_W := \widehat{W} - W$.



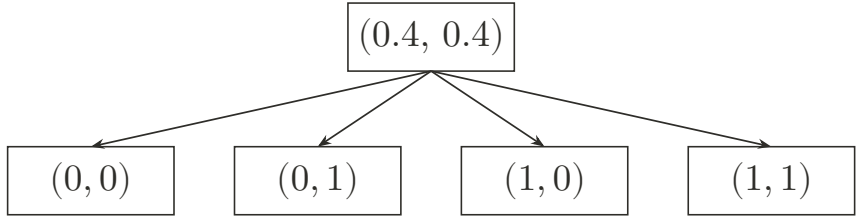
Entries	$\|E_W\|_F^2$
Worst unit input	$\sup_{\|x\|_2=1} \|E_W x\|_2^2 = \|E_W\|_{\text{op}}^2$
Actual input distribution	$\mathbb{E}_x \|E_W x\|_2^2$



A Rounding Puzzle

Input: $w = (0.4, 0.4)$, $x = (x_1, x_2)$, x_1, x_2 independent Uniform$[0, 1]$.

Round w to $\widehat{w}$ with $\widehat{w}_1, \widehat{w}_2 \in \{0, 1\}$.



Which $\widehat{w}$ minimizes $\mathbb{E}\left[(\widehat{w}^\top x - w^\top x)^2\right]$?



Independent Inputs, Coupled Errors

$\widehat{w}$	Expected squared output error
$(0, 0)$, nearest rounding	$14/75 \approx 0.1867$
$(1, 1)$	$21/50 = 0.4200$



Independent Inputs, Coupled Errors

$\widehat{w}$	Expected squared output error
$(0,0)$, nearest rounding	$14/75 \approx 0.1867$
$(0,1)$	$4/75 \approx 0.0533$
$(1,0)$	$4/75 \approx 0.0533$
$(1,1)$	$21/50 = 0.4200$



Independent Inputs, Coupled Errors

$\widehat{w}$	Expected squared output error
$(0,0)$, nearest rounding	$14/75 \approx 0.1867$
$(0,1)$	$4/75 \approx 0.0533$
$(1,0)$	$4/75 \approx 0.0533$
$(1,1)$	$21/50 = 0.4200$

Even though x_1 and x_2 are independent, a dependent rounding gives a smaller distortion.



Quantization Problem for One Row

The error separates over the rows $w^{(1)}, \dots, w^{(m)}$ of W:

$$\mathbb{E}_x \|(\widehat{W} - W)x\|_2^2 = \sum_{i=1}^{m} \mathbb{E}_x \left[\left((\widehat{w}^{(i)} - w^{(i)})^\top x \right)^2 \right].$$

So take one row, $y = w^\top x$, and minimize over $\widehat{w}$ on the grid:

$$\mathbb{E}_x \left[((\widehat{w} - w)^\top x)^2 \right] = (\widehat{w} - w)^\top H (\widehat{w} - w),$$

with $H := \mathbb{E}[xx^\top]$, the uncentered second moment of the inputs.

In practice H is estimated from calibration data, the columns of $X_{\text{cal}} \in \mathbb{R}^{d \times n_{\text{cal}}}$: $H_{\text{emp}} := X_{\text{cal}} X_{\text{cal}}^\top / n_{\text{cal}}$.



The Input Geometry

Let $e := \widehat{w} - w$.

The problem is easy with no cross terms: $H = I$ implies

$$e^\top H e = \sum_i e_i^2 \,.$$

Each coordinate on its own: nearest rounding is optimal.

In our example: $H = \begin{pmatrix} 1/3 & 1/4 \\ 1/4 & 1/3 \end{pmatrix}$.

What should we do here?



Round and Repair

Step 1: round $w_1 = 0.4$ to $\widehat{w}_1 = 0$.



Round and Repair

Step 1: round $w_1 = 0.4$ to $\widehat{w}_1 = 0$.

Step 2: find the real $\widetilde{w}_2$ minimizing

$$(1/3)(0.4)^2 + (1/3)(\widetilde{w}_2 - 0.4)^2 + (1/2)(-0.4)(\widetilde{w}_2 - 0.4)\,.$$



Round and Repair

Step 1: round $w_1 = 0.4$ to $\widehat{w}_1 = 0$.

Step 2: find the real $\widetilde{w}_2$ minimizing

$$(1/3)(0.4)^2 + (1/3)(\widetilde{w}_2 - 0.4)^2 + (1/2)(-0.4)(\widetilde{w}_2 - 0.4)\,.$$

It turns out $\widetilde{w}_2 = 0.7$.



Round and Repair

Step 1: round $w_1 = 0.4$ to $\widehat{w}_1 = 0$.

Step 2: find the real $\widetilde{w}_2$ minimizing

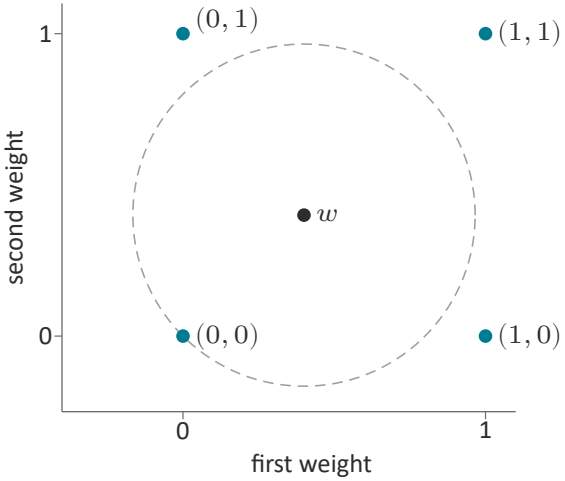
$$(1/3)(0.4)^2 + (1/3)(\widetilde{w}_2 - 0.4)^2 + (1/2)(-0.4)(\widetilde{w}_2 - 0.4)\,.$$

It turns out $\widetilde{w}_2 = 0.7$.

Step 3: round $\widetilde{w}_2 = 0.7$ to $\widehat{w}_2 = 1$.



Not All Directions Are Equal



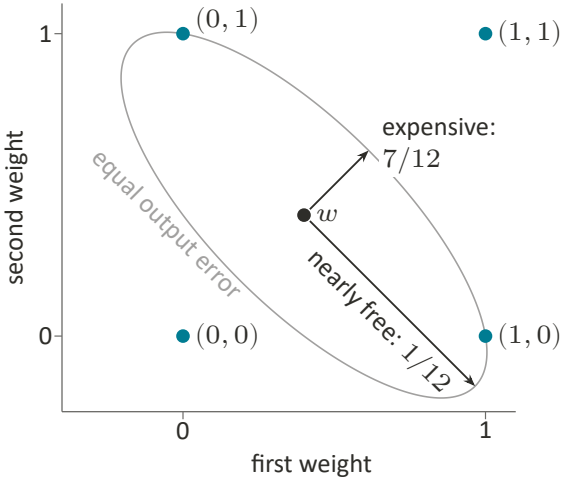
$w = (0.4, 0.4)$.

The closest grid point is $(0, 0)$.

But not all directions are equal.



Entries Versus Action



$$H = \begin{pmatrix} 1/3 & 1/4 \\ 1/4 & 1/3 \end{pmatrix},$$

$$w = (0.4,\, 0.4).$$

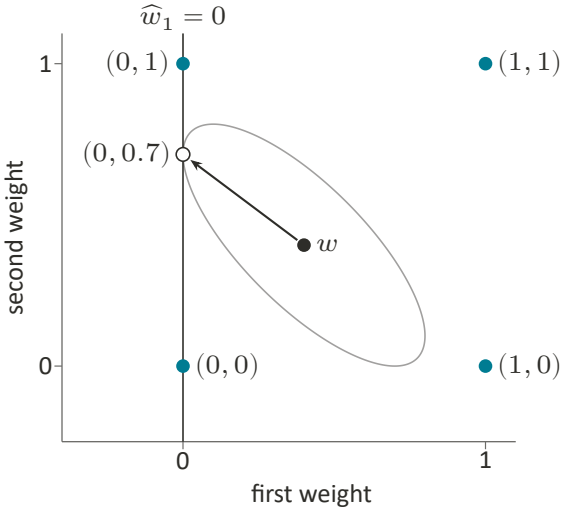
Error along directions with a large eigenvalue is expensive.

Closest grid point in the H metric is NP-hard in general.

the closest vector problem



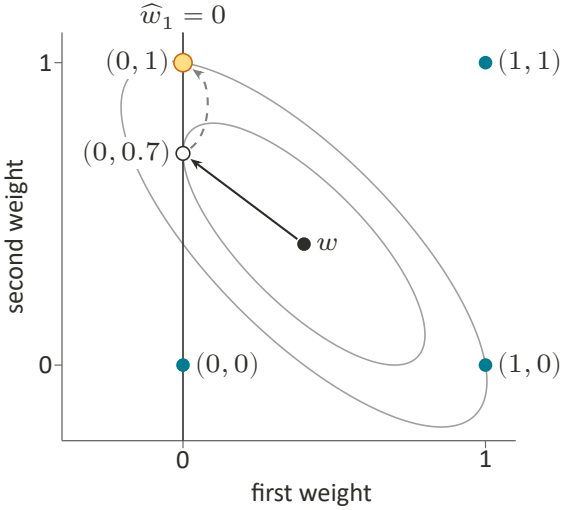
The Repair in Coefficient Space



Round and repair moves w to $(0,\ 0.7)$



The Repair in Coefficient Space



Round and repair moves w to $(0,\ 0.7)$ and then to $(0,\ 1)$.



GPTQ

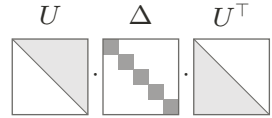
Frantar et al. (2023)

Factor $H = U\Delta U^\top$, U unit upper triangular, Δ diagonal:

$$e^\top H e = \sum_{i=1}^{d} \Delta_{ii} \left(e_i + \sum\nolimits_{j<i} U_{ji}\, e_j \right)^2 .$$

what input i adds beyond the later ones

only earlier errors: round in order



Term i involves e_i and earlier errors only: make them small one at a time.



GPTQ: The Algorithm

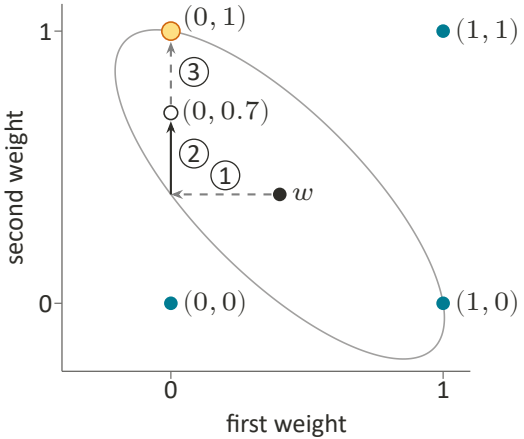
For $i = 1, \dots, d$:

1. corrected target $\widetilde{w}_i := w_i - \sum_{j<i} U_{ji}\, e_j$;
2. round $\widetilde{w}_i$ to $\widehat{w}_i$, the nearest grid point;
3. record the error $e_i := \widehat{w}_i - w_i$.

Error feedback: the errors already made are pushed onto the weight about to be rounded.



GPTQ on the Puzzle



1 Round w_1: $\widetilde{w}_1 = 0.4 \to 0$, so $e_1 = -0.4$.

2 Feedback: $\widetilde{w}_2 = 0.4 + (3/4) \times 0.4 = 0.7$.

3 Round: $0.7 \to 1$. Output $(0,\ 1)$.

Cost $(7/48)(0.4^2) + (1/3)(0.3^2)$ $= 4/75 \approx 0.053$.

Greedy, in the right metric: a best point, $(0,\ 1)$.



Greedy Repair Need Not Be Optimal

Each repair is the best continuous adjustment after a commitment.

The final discrete vector need not be globally best.



Greedy Repair Need Not Be Optimal

Each repair is the best continuous adjustment after a commitment.

The final discrete vector need not be globally best.

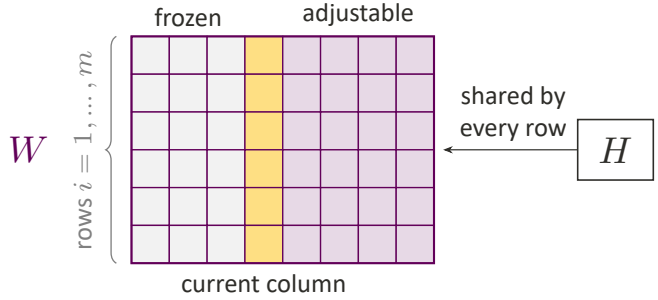
Same H and order, $w = (0.4,\ 0.3)$:

Choice	Expected squared output error
GPTQ returns $(0, 1)$	$23/300 \approx 0.0767$
Alternative $(1, 0)$	$18/300 = 0.0600$



Rows Separate, Work Is Shared

The shared expensive computation: the factorization and compensation coefficients derived from H.



In practice scales may differ by row, which changes the grids but not the shared H.



The GPTQ Guarantee

Theorem. With an integer grid and no clipping, GPTQ's error satisfies $e^\top H e \le \frac{1}{4}\operatorname{tr}(\Delta)$.

Proof. Left as an exercise.

Hint: with $r_i := \widehat{w}_i - \widetilde{w}_i$, the fresh rounding residual of step 2,

$$e^\top H e = \sum_{i=1}^{d} \Delta_{ii}\, r_i^2 \le \frac{1}{4}\operatorname{tr}(\Delta)\,,$$

since each rounding step misses by at most half a grid step: $|r_i| \le \frac{1}{2}$.

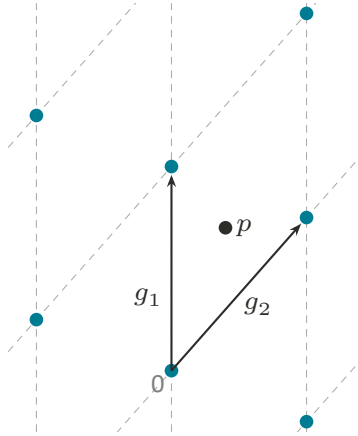


The Closest Lattice Point Problem

$G := [g_1 \cdots g_d]$, with linearly independent basis vectors.

$\mathcal{L} := \{Gz : z \in \mathbb{Z}^d\}$.

Given p, find $z \in \mathbb{Z}^d$ minimizing $\|Gz - p\|_2^2$.

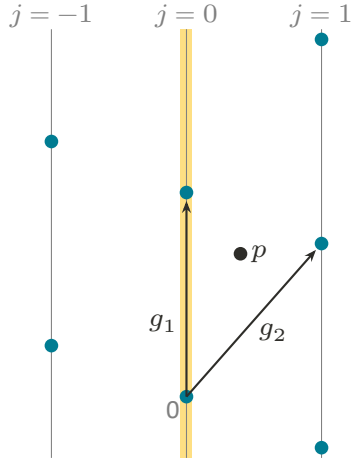


Babai's Nearest-Plane Algorithm

Babai (1986)

In two dimensions:

1. Choose the nearest line $jg_2 + \text{span}(g_1)$, $j \in \mathbb{Z}$.

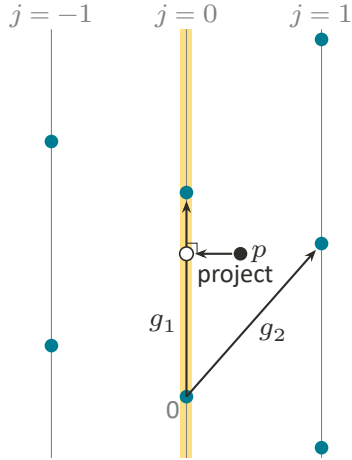


Babai's Nearest-Plane Algorithm

Babai (1986)

In two dimensions:

1. Choose the nearest line $jg_2 + \text{span}(g_1)$, $j \in \mathbb{Z}$.
2. Project onto that line.

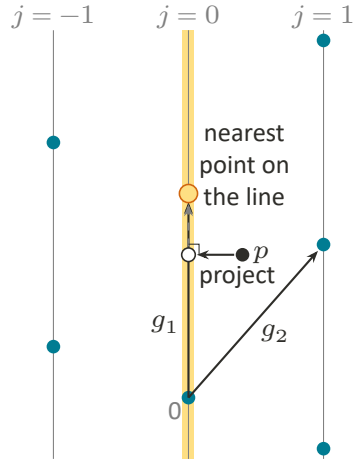


Babai's Nearest-Plane Algorithm

Babai (1986)

In two dimensions:

1. Choose the nearest line $jg_2 + \text{span}(g_1), j \in \mathbb{Z}$.
2. Project onto that line.
3. Choose the nearest lattice point along it.



GPTQ in Lattice Coordinates

$H = A^\top A$, $\widehat{w} = sz$:

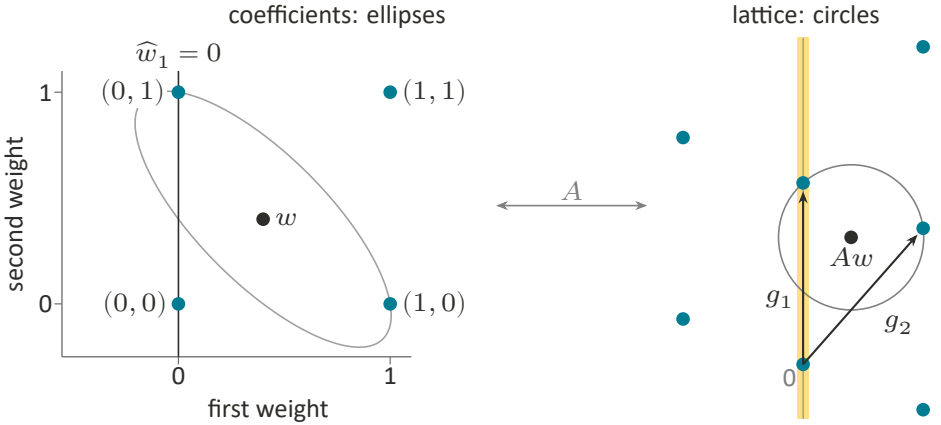
$$(\widehat{w} - w)^\top H(\widehat{w} - w) = \|sAz - Aw\|_2^2 .$$

Target $= Aw$. Lattice basis $=$ columns of sA.

GPTQ step	Geometric step
Commit a coefficient	Select a parallel lattice plane
Repair remaining real coefficients	Project onto that plane

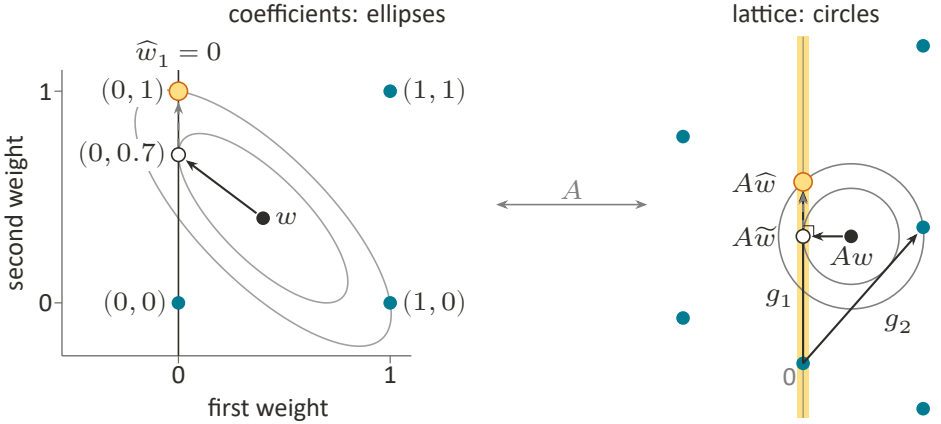


GPTQ in Lattice Coordinates



GPTQ in Lattice Coordinates

With matched ordering and no clipping, Babai's nearest-plane algorithm is the same as GPTQ (Chen et al., ICLR 2026).



What Happens in a Language Model?

OPT-175B, WikiText2 perplexity, original GPTQ experiments (2023).

Perplexity: $\exp$ of the average negative log-probability per token; lower is better.

Sixteen-bit reference: 8.34.

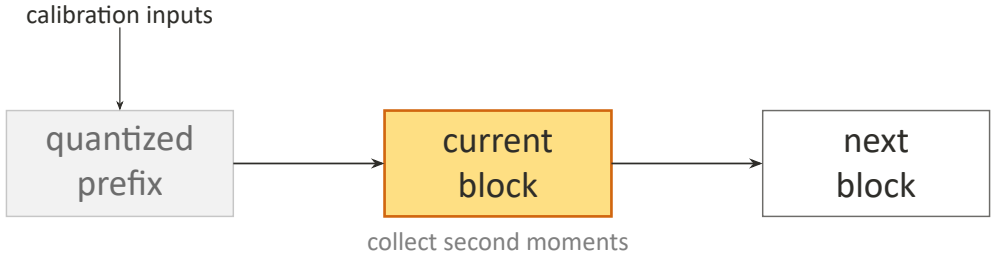
Weight bits	Nearest rounding	GPTQ
4	10.54	8.37
3	approximately 7,300	8.68

Source: Frantar et al., GPTQ (ICLR 2023), Table 3.



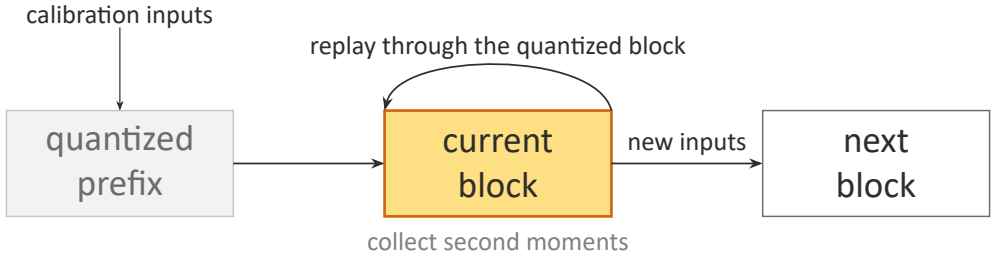
Preparing the Compressed Model

Calibration inputs; collect second moments; quantize the current block; replay it; pass its outputs onward.



Preparing the Compressed Model

Calibration inputs; collect second moments; quantize the current block; replay it; pass its outputs onward.



Historical example: OPT-175B, 128 text segments of 2,048 tokens, about 4.2 hours on one 80 GB A100.

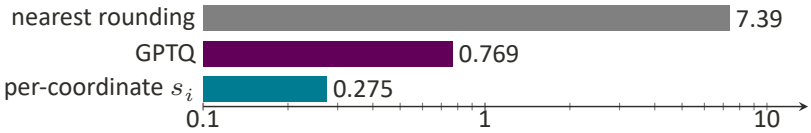


A Ladder of Three Costs

Lifar et al. (2026)

s: the grid spacing, here per coordinate (s_i, geometric mean $\overline{s}$).

Method, same total bits	Average cost
Nearest rounding	$(s^2/12)\ \mathrm{tr}(H)$
GPTQ, one spacing	$(s^2/12)\ \mathrm{tr}(\Delta)$
GPTQ, spacing s_i per coordinate	$(\overline{s}^{\,2}/12)\, d\, \det(H)^{1/d}$

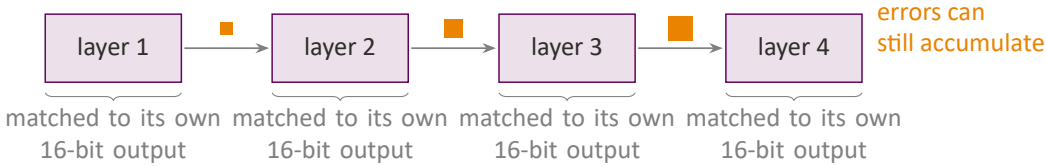


synthetic layer, $d = 64$

WaterSIC: within 0.255 bit of the best possible, at high rate.

Limits of GPTQ

Each layer is matched to its own 16-bit output: the objective is local.



1. Greedy commitments need not give the best discrete vector.
2. Calibration can differ from deployed inputs.
3. The local layer objective differs from complete-model quality.
4. Clipping can invalidate the half-step residual bound.



Five Questions for Any Tensor

	weights today	**activations** today	**keys and values** Lecture I
When is it known?	before deployment		at run time, then kept
Time to encode it?	hours, once		almost none: once per token
What must be preserved?	the layer's output on typical inputs		inner products (keys); averages (values)
What do values look like?	roughly bell-shaped		outliers in a few key channels
How does hardware use it?	read for every generated token		read at every later step

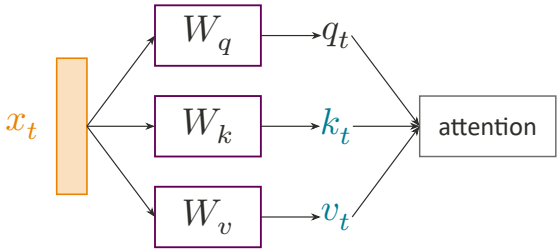
Weights are done. Now the other operand: activations.



Part II:
Quantizing Activations



Reminder: Activations



1. Multiplied immediately by a weight matrix.

2. Usually discarded after use.

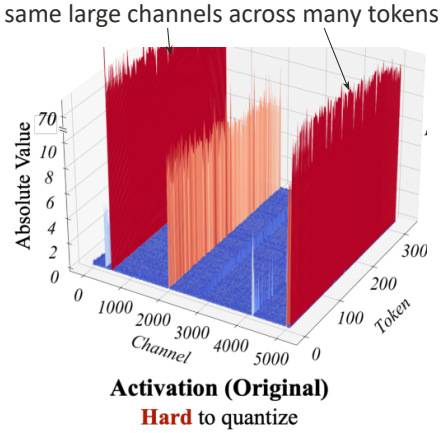
Activations are new at every token, and multiplied at once: their quantizer must be cheap.



Outlier Channels

Which entries should share a quantization scale: one token at a time, or one channel at a time?

The same channels are persistently large; the computational objective is different from the KV cache.

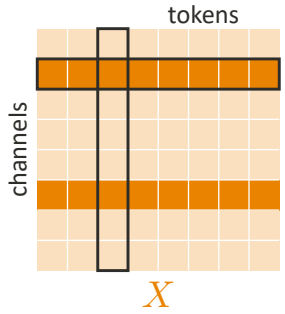


Source: Xiao et al., SmoothQuant (ICML 2023), Figure 4 (annotated): one linear layer of OPT-13B.

One Scale per Token or per Channel?

Per token

One scale for one activation vector. The outlier channel still sets the scale.



Per channel

One scale for one coordinate across tokens. Its scales lie along the inner dimension of WX.

Per-token scaling matches the hardware.
Per-channel scaling matches the statistics.



Dynamic Per-Token Quantization

$\tau_t := \max_j |x_{jt}| \,/\, C_{\max}\,, \qquad x_t \approx \tau_t\, Q(x_t/\tau_t)\,.$

x_{jt}: channel j of x_t; $C_{\max}$: the largest code, 127 for INT8.

Dynamic: choose τ_t from the current vector.

INT8. Equally spaced levels after scaling.

FP8. Exponent gives representable values at several magnitudes.

Dynamic tells us when the scale is chosen.
FP8 tells us which values are representable.



Why Not One Scale per Channel?

One scale per channel

$$x_j = \lambda_j \overline{x}_j\,,$$
$$(Wx)_i = \sum_j W_{ij}\,\lambda_j\,\overline{x}_j\,.$$

A different scale inside every sum.

One scale per token

$$Wx_t = \tau_t\, W(x_t/\tau_t)\,.$$

One scale factors outside the multiplication.

Per-channel scaling is accurate, but a naïve implementation complicates fast low-precision multiplication.



Why Not One Scale per Channel?

One scale per channel

$$x_j = \lambda_j \overline{x}_j\,,$$
$$(Wx)_i = \sum_j W_{ij}\,\lambda_j\,\overline{x}_j\,.$$

A different scale inside every sum.

One scale per token

$$Wx_t = \tau_t\, W(x_t/\tau_t)\,.$$

One scale factors outside the multiplication.

Bridge: absorb the fixed channel scales into W.

Per-channel scaling is accurate, but a naïve implementation complicates fast low-precision multiplication.

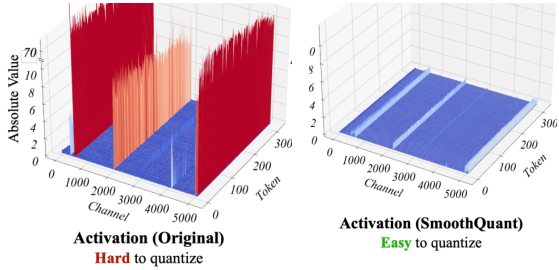


SmoothQuant

$$WX = (W\Lambda)(\Lambda^{-1}X)\,,$$ Λ is a fixed diagonal matrix estimated from calibration examples.

Activations: divide persistent outlier channels down.

Weights: multiply the matching weight columns up.



Use per-channel information offline; keep the runtime matrix multiplication hardware-friendly.

Source: Xiao et al., SmoothQuant (ICML 2023), Figure 4, panels 1–2 (OPT-13B).

AWQ: Same Identity, New Objective

SmoothQuant

Large activation channels are hard to quantize.

Rescale to make activations easier.

Target: W8A8.

AWQ

Errors in weights multiplied by large activations matter more.

Rescale to protect those weight channels.

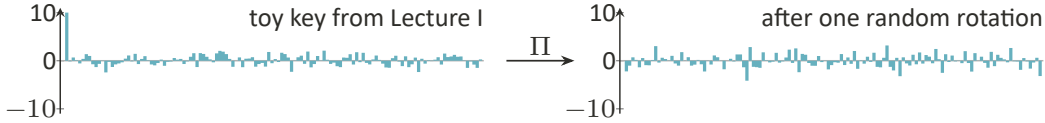
Target: W4A16.

AWQ is activation-aware, but it is primarily a weight-only quantization method.



Rotate: The Idea from Lecture I

$$WX = (W\Pi^\top)(\Pi X), \qquad \Pi^\top\Pi = I .$$



QuaRot: fixed randomized Hadamard rotations spread outlier energy across coordinates.

SpinQuant: learned rotations merged into the weights, plus fixed run-time Hadamards.

Rotations can remove outlier channels, but any transform applied at runtime is paid on every token.



Activation Quantization in Practice

Activations	Recipe	Supported in	Status
BF16 / FP16	W4A16	GPTQ, AWQ	baseline
dynamic FP8	W8A8	TensorRT-LLM; vLLM	mature
INT8 (SmoothQuant)	W8A8	TensorRT-LLM; vLLM	mature
NVFP4	W4A4	TensorRT-LLM; vLLM (Blackwell)	newer
rotation-based	W4A4KV4; W4A8	QuaRot, SpinQuant; Llama 3.2 (edge)	aggressive

A4 needs finer scales, transforms, or new hardware.

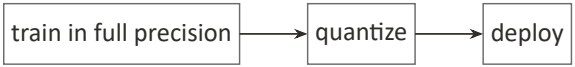


Part III: Quantization-Aware Training



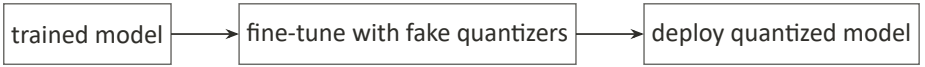
Let the Model See the Rounding

Post-training quantization (PTQ)



The trained model never sees its quantization errors.

Quantization-aware training (QAT)

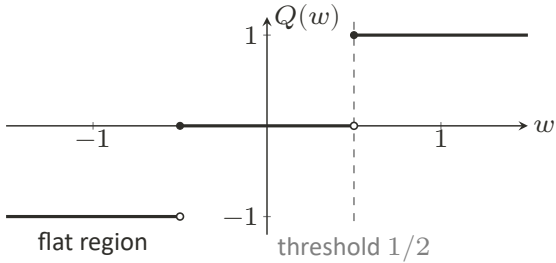


Forward pass uses rounded weights and activations.
Training keeps full-precision latent weights.

QAT changes the model so that the quantized computation works better.



Rounding Has No Useful Gradient



$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial Q(w)} Q'(w)\,,$$

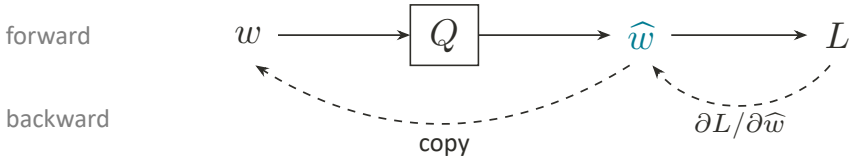
$Q'(w) = 0$ almost everywhere.

Ordinary gradient descent cannot cross the steps.

Putting a real rounding operator in the forward graph is not enough; the backward pass needs a surrogate.



The Straight-Through Estimator



$$\widehat{w} = Q(w) \qquad \frac{\partial L}{\partial w} \approx \frac{\partial L}{\partial \widehat{w}}$$

Update the latent w; deploy $Q(w)$.

The estimator is deliberately biased: it is a practical search direction, not the derivative of the staircase.

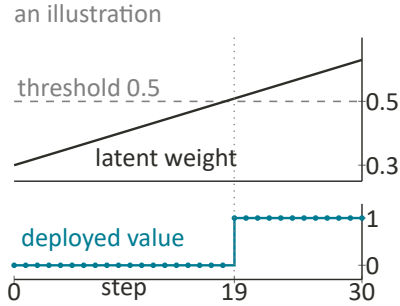


Straight-Through: What We Know

Bengio, Léonard, Courville (2013): introduced the heuristic.

Yin et al. (2019): with a well-chosen surrogate, descent in a two-layer network with binarized activations.

General theory for deep transformers remains incomplete.



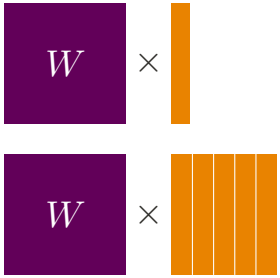
small updates accumulate until the deployed value changes

QAT is an effective optimization device, but the surrogate gradient does not have a complete general justification.

Part IV:
Memory, Compute, and Speed



One Model, Two Regimes



Decode: one or a few tokens. $y = Wx$. Little arithmetic per weight: often limited by memory bandwidth.

Prefill or large batch. $Y = WX_{\text{run}}$. Weights reused across many tokens: often limited by arithmetic throughput.

First-order model: time $\gtrsim \max\left(\frac{\text{bytes}}{\text{bandwidth}}, \frac{\text{operations}}{\text{compute rate}}\right)$

Weight-only compression pays when reading weights is the bottleneck; low-precision activations, when arithmetic is.



What Each Configuration Changes

Format	Bytes	Arithmetic	Pays when
W4A16	↓ weights	↔ 16-bit, after dequantizing	weight traffic dominates
W8A8	↓ both	↓ native 8-bit	prefill, large batches
W4A4	↓ both, block-scaled	↓ native 4-bit	hardware supports it
KV8	↓ cache, attention traffic	↔ linear layers unchanged	long context, large batches

These savings should not be multiplied mechanically.



Why Few Bits Are Not Enough

1 The bottleneck moves

Compute, cache traffic, or communication may dominate.

2 Unpacking costs time

Decoding, scaling, and conversion.

3 Hardware must multiply it

Needs an efficient kernel.

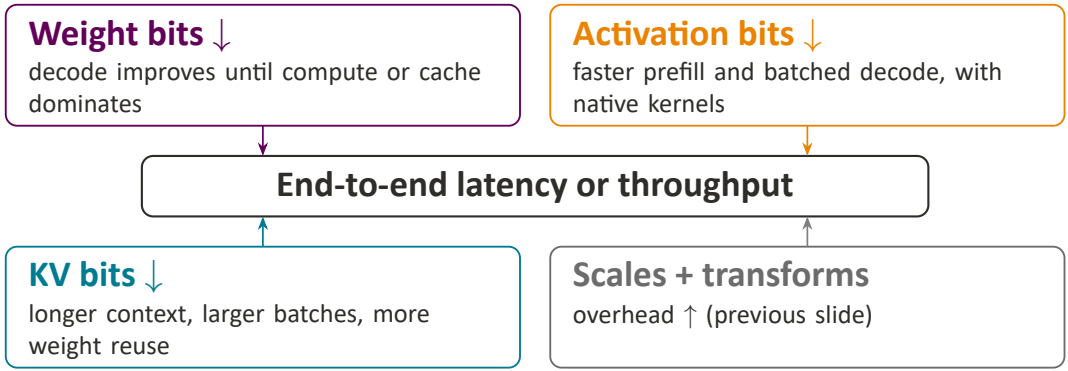
4 Runtime transforms are paid

Folded transforms are free.

Speed is set by the new bottleneck, not by bit width alone.



The Costs Interact



Quantization choices interact through memory, arithmetic, batch size, and overhead.

Two Lectures, Three Problems

	weights	activations	keys and values
When is it known?	before deployment	at run time, each layer and token	at run time, then kept
Time to encode it?	hours, once	almost none: every token	almost none: once per token
What must be preserved?	the layer's output on typical inputs	its effect on the next matrix multiplication	inner products (keys); averages (values)
What do values look like?	roughly bell-shaped	a few persistent outlier channels	outliers in a few key channels
How does hardware use it?	read for every generated token	an operand of fast matrix multiplication	read at every later step

Preserve the computation that consumes the tensor, not merely the individual entries.



Quantization: Four Ideas to Keep

1. Preserve the right thing.
2. Transform before rounding.
3. Train through rounding when needed.
4. Measure end to end.

A good quantizer preserves what the next computation needs, in a representation the machine can use efficiently.



Research 1: Bits Across Tokens

The cache stores one key per token, and today every token gets the same number of bits. With a fixed total, how should bits be spread across tokens?

Attention concentrates on a few tokens, but which ones is unknown when a key is written.

Eviction, dropping a token from the cache, is the case of zero bits. When is it better than keeping one bit?

If future attention were known, the best allocation is known. The open problem is the online one: bits chosen at write time, and lowered later as a token ages, without reading it back.

References: Equitz & Cover (1991) · KIVI · H2O · SnapKV · MiKV · RateQuant (2026).



Research 2: Rounding a Layer

GPTQ turned out to be a classical lattice algorithm. What do the more advanced lattice methods give for rounding a layer?

GPTQ was designed by engineers, independently of Babai's 1986 algorithm, and is the same algorithm. The lattice literature has stronger methods that are still unused.

Exact search where H has its few large eigenvalues; the simple greedy step on the rest.

Randomized roundings whose errors are nearly zero on average.

One codebook for every layer: a single, fixed set of rounding points that is nearly optimal whatever the layer's inputs look like. One is known to exist (within 0.11 bit); nobody knows how to build it.

References: Babai (1986) · Chen et al. (ICLR 2026) · WaterSIC (2026) · Klein (2000) · DiscQuant (COLT 2025) · Harbuzova, Ordentlich, Polyanskiy (2026).



Research 3: Rounding in Groups

Rounding weights in groups beats rounding them one at a time. How much of that gain survives when each weight must be recovered cheaply?

Rounding each weight on its own loses about a quarter bit even with entropy coding, and about half a bit at a fixed 2 to 4 bits.

The best codes treat many weights jointly, but a full codebook is far too large to store and search.

In between: codes over small groups (8 or 24 weights) that a table lookup can decode. How close can they get?

References: Makhdoumi, Huang, Médard, Polyanskiy (2013) · Conway & Sloane · QuIP# · QTIP (NeurIPS 2024) · NestQuant · Leech-lattice codes (2026).



Research 4: Bits a Model Stores

Does the information a model stores per parameter decide how far it can be quantized?

Morris et al. estimate that GPT-style models can memorize about 3.6 bits of information per parameter (3.51 in bf16, 3.83 in fp32).

By perplexity, 4-bit weights are nearly free, 3 bits cost some quality, 2 bits a lot; yet stored knowledge already falls at 4 bits (about 2 to 0.7 bits per parameter, Allen-Zhu and Li). Models trained on more tokens are harder to quantize.

If a model stores about 3.6 bits per parameter, rounding below that should lose information. Does storage predict where quality drops?

A place to prove it: a single-layer perceptron, whose capacity is known: 2 patterns per weight with real weights, about 0.83 with ± 1 weights.

References: Morris et al. (ICML 2026) · Kumar et al. (ICLR 2025) · Allen-Zhu & Li (ICLR 2025) · Ouyang et al. (ACL 2025) · Krauth & Mézard (1989) · Gutfreund & Stein (1990).



Questions?

Quantization I and II: vectors, then matrices.

