Language Model Quantization I
Quantizing Vectors

Amin Saberi

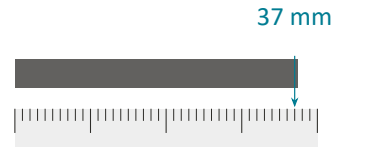
MS&E 319

Autumn 2026

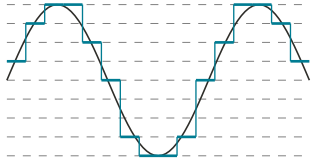


Quantization

Replace a continuous quantity by one of finitely many values.



(a) a length, read to the nearest millimetre



(b) Spotify Lossless: 24 bits per sample, 44,100 samples per second



(c) 8 billion weights: 16 GB at 16 bits, 4 GB at 4 bits



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.

W4 A16 KV8 weights in 4 bits, activations in 16, cache in 8

Today: the cache. Next lecture: weights and activations.



The Plan for Two Lectures

Lecture I (today): quantizing vectors, applications to quantizing the KV cache



not known in advance

encode in real time

preserve inner products

Lecture II: quantizing matrices, and what makes it fast



weights: known in advance, with sample inputs: round to preserve the layer's output



activations: outlier channels: rescale and rotate



Five Questions for Quantization

	weights Lecture II	**activations** Lecture II	**keys and values** today
When is it known?			at run time, then kept
Time to encode it?			almost none: once per token
What must be preserved?			inner products (keys); averages (values)
What do values look like?			outliers in a few key channels
How does hardware use it?			read at every later step

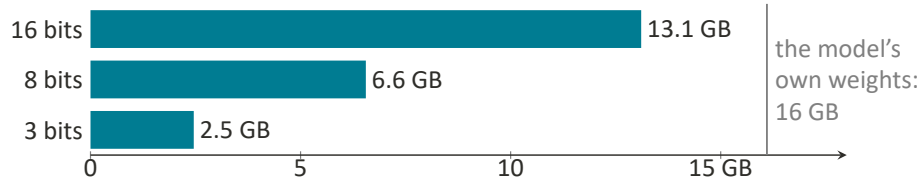
We will discuss the right-hand column today.



Example: Llama 3.1 8B

The size of KV cache:

$$\underbrace{2}_{\text{keys and values}} \times \underbrace{32}_{\text{layers}} \times \underbrace{8}_{\text{key-value heads}} \times \underbrace{128}_{\text{head dimension}} \times \underbrace{100{,}000}_{\text{tokens}} \times \underbrace{16 \text{ bits}}_{\text{bits}} \approx 13 \text{ GB}$$



One conversation. A server holds one cache per user.

Can we store each number in 3 bits instead of 16?

Numbers: counted from the published configuration of Llama 3.1 8B.



How Numbers Are Stored

A float is a sign, an exponent and a fraction. FP32 uses 32 bits.



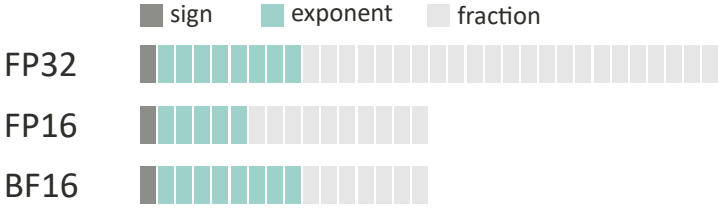
$$x = (-1)^{\text{sign}} \times (1.\text{fraction})_2 \times 2^{\text{exponent}-127}.$$

Largest: $(2 - 2^{-23}) \times 2^{127} \approx 2^{128}$. Smallest positive: 2^{-126}.



How Numbers Are Stored

Fewer bits: the same formula, other splits of the bits.



about 7 digits; up to 3.4×10^{38}

about 3 digits; up to 65,504

about 2 digits; the range of FP32

Exponent bits set the range; fraction bits set the precision.

Models ship in 16 bits. What do 8 and 4 bits look like?



Fewer Bits: FP8, INT8, INT4

Micikevicius et al. (2022)

A b-bit format is a list of at most 2^b values.

FP8 (E4M3)

1 sign, 4 exponent, 3 fraction bits; 8 values per power of two



INT4 with a scale

value = scale × integer; INT8: the same with 256 values





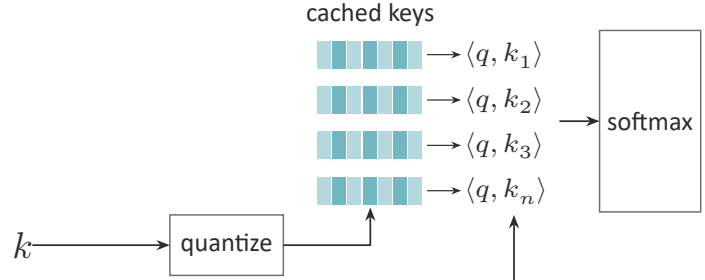
Common Quantization Configurations

Quantity	Established	More aggressive	Algorithmic ideas
Weights	FP8 / INT8	INT4, NVFP4	GPTQ, AWQ, quantization-aware training
Activations	FP8 / INT8	NVFP4	dynamic scaling, SmoothQuant, outlier handling
KV cache	FP8	NVFP4 / INT4	KIVI, Hadamard rotations, TurboQuant

Sources: LLM Compressor and NVIDIA (formats); vLLM, SGLang, TensorRT-LLM (cache); each method's paper.



Today's Problem



online: encoded when it is produced
data-oblivious: no data to tune on
few bits: b about 2 to 4 per coordinate
inner products: $\langle q, k_i \rangle$ for later queries

The keys are compressed. The query is not.



Today's Problem

Four requirements

- [ ] **Online.** Each key is encoded once, when it is produced: generation adds one token at a time, and the next step reads it.
- [ ] **Data-oblivious.** No calibration data, no training: keys depend on each user's input, which is not known in advance.
- [ ] **Few bits.** About 2 to 4 bits per coordinate instead of 16: the cache is larger than the model at long contexts.
- [ ] **Inner products.** Keep $\langle q, k_i \rangle$ accurate for queries not yet seen: attention reads the keys only through these scores.



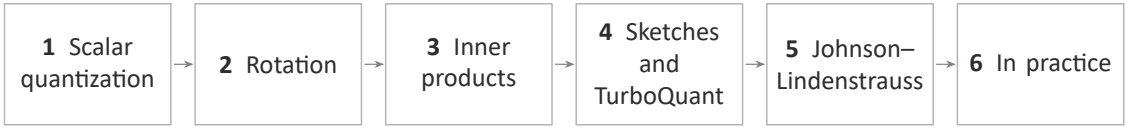
What You Will Learn

1. Lloyd–Max quantizers: optimal levels for a known density.
2. Random rotations: every coordinate becomes nearly Gaussian.
3. Bias: error-optimal quantizers shrink inner products.
4. Unbiased one-bit sketches (QJL) and TurboQuant.
5. Johnson–Lindenstrauss: random projections keep distances.
6. Reported results, and how deployed systems differ.



Roadmap

Six steps to a compressed cache.



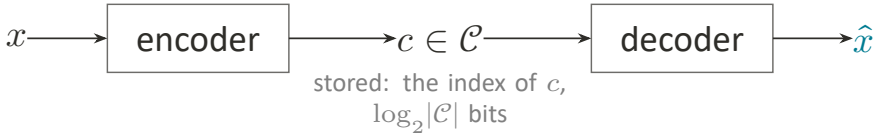


What Is a Quantizer?

Store $x \in \mathbb{R}^d$ in a fixed number of bits: only finitely many outputs.

A **quantizer** is a map $Q : \mathbb{R}^d \to \mathcal{C}$, with $\mathcal{C}$ a finite set: the **codebook**.

The reconstruction of x is $\hat{x} := Q(x)$.



How do we say whether Q is good?



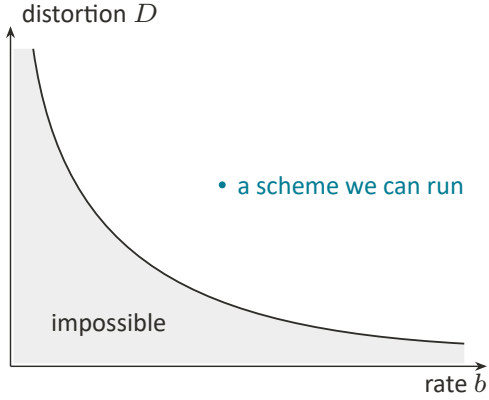
Rate and Distortion

Rate: bits per coordinate.

$$b := \frac{1}{d} \log_2 |\mathcal{C}| \,,$$

Distortion: expected squared error.

$$D := \mathbb{E} \, \|x - Q(x)\|_2^2 \,.$$



For a given rate: how small can D be, and at what cost?

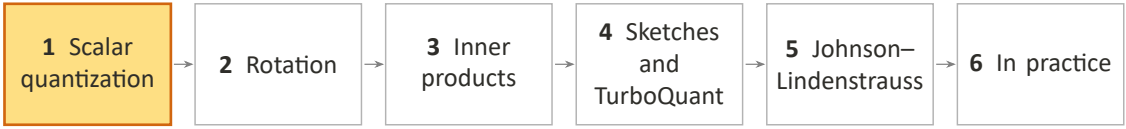


Optimal Quantization: Algorithmic Ideas



Roadmap

Six steps to a compressed cache.

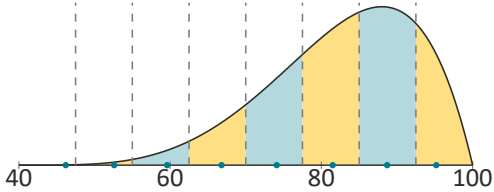


First: what is the best quantizer for one number?

Start with One Number

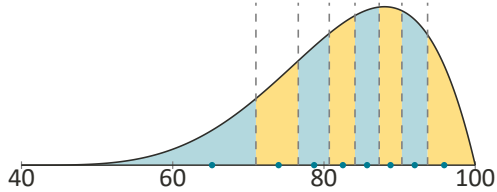
The scores of a class, to be turned into 8 grades, A+ to C.
Which cutoffs make the grades distort the scores the least?

equal width: 7.5 points each



$D = 4.36$

percentiles: an eighth of the class each



$D = 3.93$

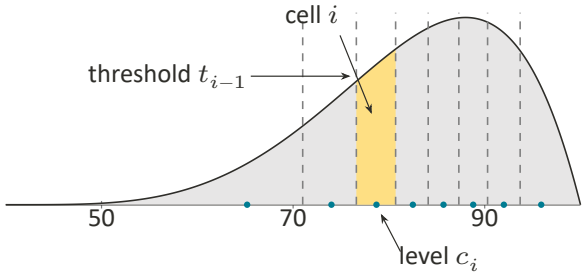
each grade stands for the mean score of its students; D in squared points

Is either one the best?



The Problem, Formally

X has density f. Choose $N := 2^b$ levels $c_1 < \cdots < c_N$ and thresholds t_i between them; cell i is $(t_{i-1}, t_i]$.



f: the density of the scores

$$D = \sum_{i=1}^{N} \int_{t_{i-1}}^{t_i} (x - c_i)^2 \, f(x) \, dx \, .$$

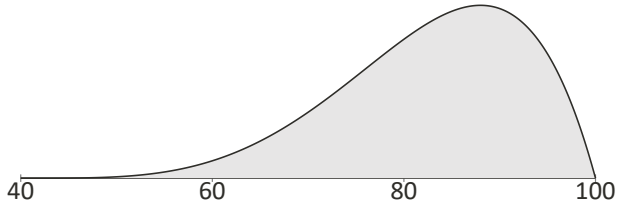
cell i — squared error

The problem: choose the levels and thresholds that minimize D.



Finding the Best Levels

How do we find the levels and thresholds with the least D, for any density f?



the scores again, or any other density

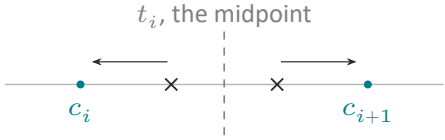
Hint: at the best cutoffs, what happens if we move one a little?



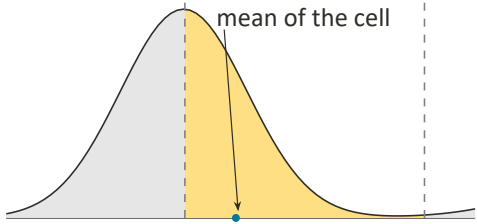
Optimal Quantizer: Two Conditions

Lloyd (1957); Max (1960)

Nearest level: each threshold is the midpoint of its two levels.
Centroid: each level is the mean of its cell.



$$t_i = \frac{c_i + c_{i+1}}{2},$$



$$c_i = \mathbb{E}[X \mid X \in (t_{i-1}, t_i]].$$

Each condition assumes the other. How do we satisfy both?



The Lloyd–Max Algorithm

Alternate the two conditions

Lloyd (1957); Max (1960)

Start from any levels $c_1 < \cdots < c_N$. Repeat:

1. Thresholds to midpoints: $t_i \leftarrow \frac{c_i + c_{i+1}}{2}$.
2. Levels to centroids: $c_i \leftarrow \mathbb{E}\big[X \mid X \in (t_{i-1}, t_i]\big]$.

Stop when D no longer falls.

Neither step can increase D. It works for any density.

For the scores, from percentiles: D falls from 3.93 to 2.36.

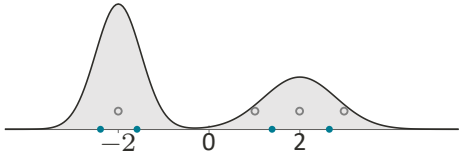
Numbers: simulated: L1-F15.py.



Lloyd–Max Algorithm's Optimality

Theorem (Fleischer 1964; Trushkin 1982; Kieffer 1983). If $\log f$ is concave, one set of levels meets both conditions: the optimum.

same algorithm, different start



not log-concave: two answers,
$D = 0.145$ (teal) and $D = 0.198$ (gray)

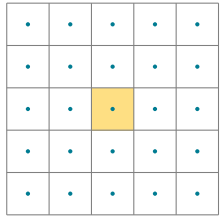
$f = 0.6\,\mathcal{N}(-2, 0.5^2) + 0.4\,\mathcal{N}(2, 0.8^2)$:
two bumps, so $\log f$ is not concave

With two bumps, Lloyd–Max can stop short.

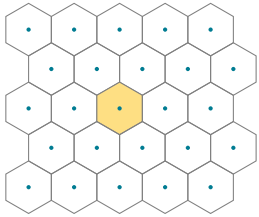


Quantizing Coordinates Jointly

One codebook for the whole vector does better: rounder cells.



one coordinate at a time



jointly: rounder cells, less error

But it needs 2^{bd} points: $2^{384} \approx 10^{115}$ for $d = 128$, $b = 3$.

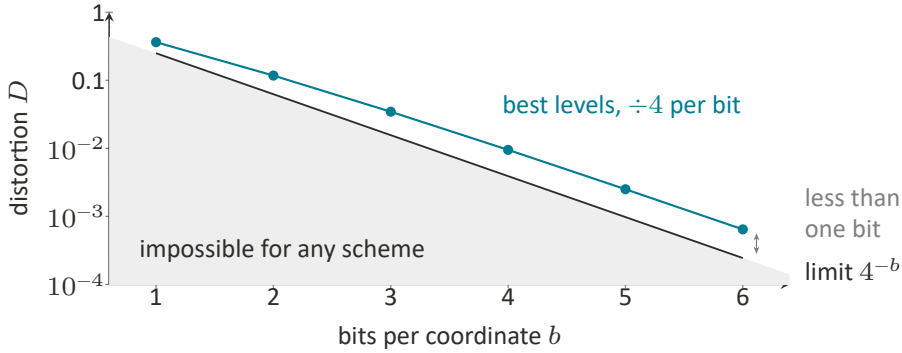
Joint codes help, and cannot be built.



A Lower Bound for Gaussians

Any scheme at rate b bits per coordinate has distortion at least $\sigma^2\, 4^{-b}$, for a Gaussian.

D_b: the distortion of the best b-bit levels for $\mathcal{N}(0,1)$, the teal dots

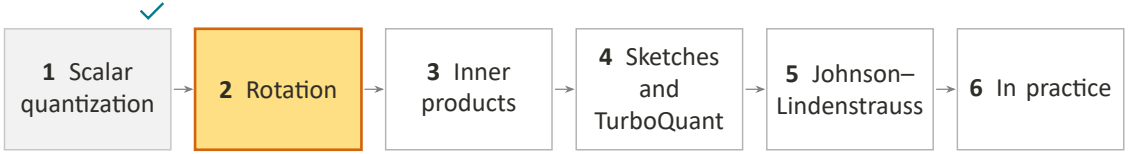


All of this needs a known density. Are real keys like that?



Roadmap

Six steps to a compressed cache.

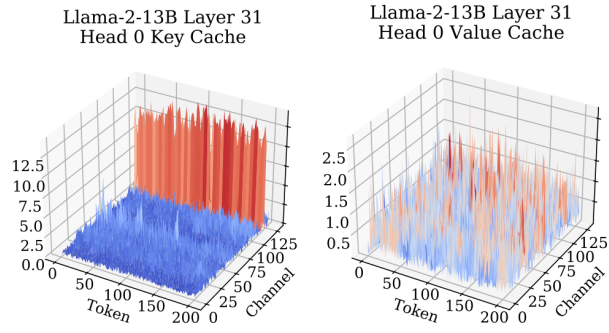




Real Keys Are Spiky

Liu et al., KIVI (2024)

In the key cache, a few fixed channels are far larger than the rest.



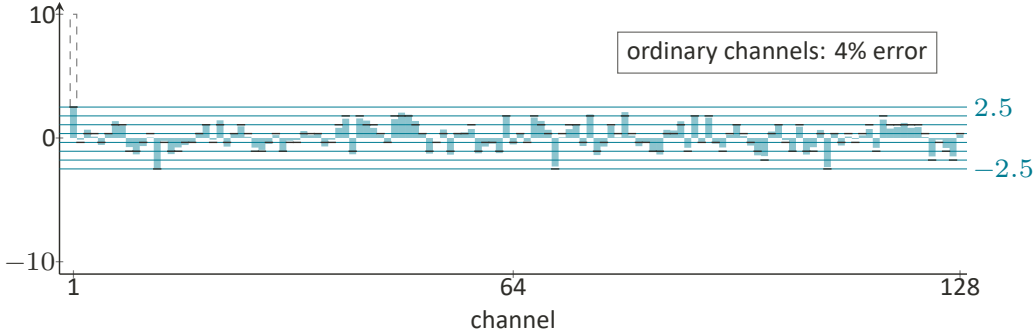
What happens when we quantize a vector like this?

Source: Liu et al., KIVI (2024), Figure 2: Llama-2-13B, layer 31, keys and values.

Why Spikes Hurt

A toy key: one channel at 10, 127 channels of size about 1.

3 bits per coordinate, equally spaced levels.



No fixed grid serves both. How would you fix this?

Numbers: simulated: check_lecture1_draft_numbers.py; lecture1_slide_numbers.py.



How Would You Fix This?

Spiky keys: how would you fix this?

Per-channel scales: need statistics over many tokens. Not online.

Clip the large values: the error lands on the largest channels.

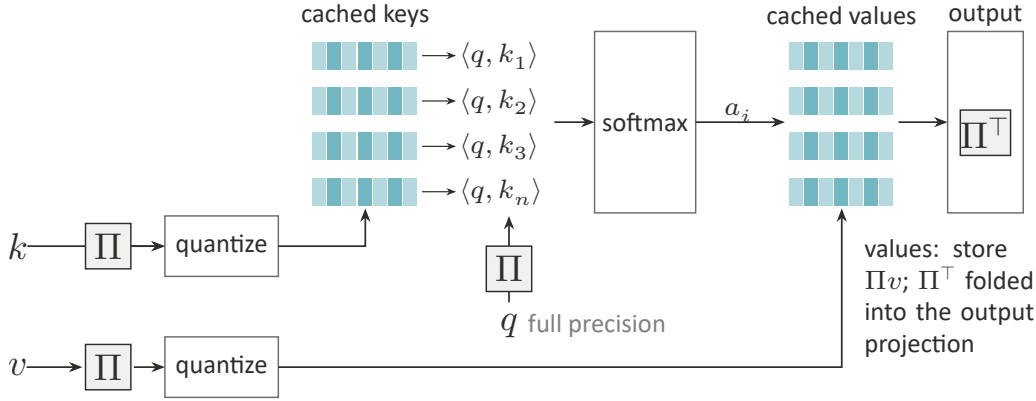
More bits for large channels: must know which. Calibration data.

Can we change coordinates so that no channel is special?



Idea: Rotate Keys, Queries and Values

One random rotation Π ($\Pi^\top \Pi = I$), drawn once: $\langle \Pi q, \Pi k \rangle = \langle q, k \rangle$.



Same function. What do the rotated keys look like?

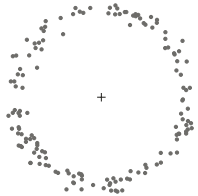


Where Do Gaussian Vectors Live?

Draw many vectors $g \sim \mathcal{N}(0, I_d)$, in d dimensions.

Where do they lie?

- ☐ (a) near the origin
- ☐ (b) spread everywhere
- ☐ (c) on a thin sphere



$\|g\|_2^2$ is a sum of d terms of mean 1, so $\|g\|_2 \approx \sqrt{d}$.

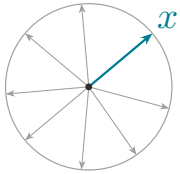
Gaussian vectors lie on a sphere of radius $\sqrt{d}$, with a small error.



Rotated Entries Are Gaussian

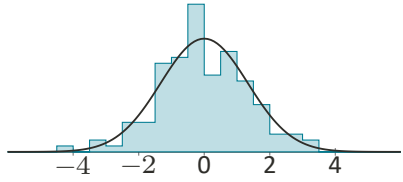
Claim. For any fixed x and a random rotation Π,
(i) Πx is a random point on the sphere of radius $\|x\|_2$;
(ii) therefore each entry of Πx is $\mathcal{N}(0, \|x\|_2^2/d)$, with a small error.

Πx: radius $\|x\|_2$



$\Rightarrow$

approximately Gaussian

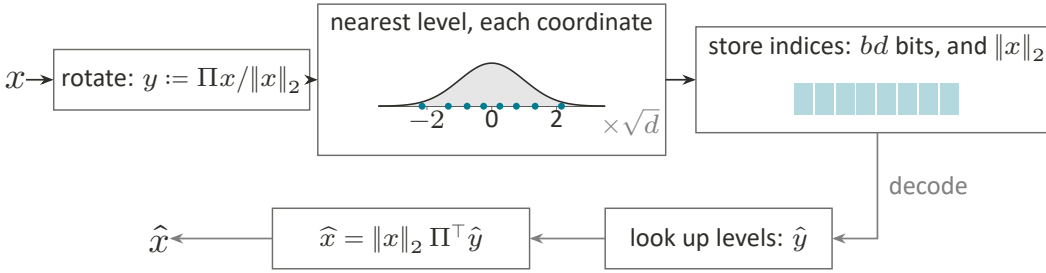


$\approx \mathcal{N}(0, \|x\|_2^2/d)$



Rotate, Then Quantize Coordinates

The levels are computed once, offline, by the Lloyd–Max algorithm for one density: a random unit vector's coordinate, $\approx \mathcal{N}(0, 1/d)$.



No data needed: the levels exist before the first key.



The Hadamard Transform

A rotation that is fast to apply. For d a power of 2, start from $H_1 := 1$:

$$H_{2k} := \frac{1}{\sqrt{2}} \begin{pmatrix} H_k & H_k \\ H_k & -H_k \end{pmatrix}, \qquad H_4 = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}.$$

Randomized: $\Pi := H_d \operatorname{diag}(\xi)$, with $\xi \in \{\pm 1\}^d$ random signs.

Hadamard rotation: a cheap and discrete approximation of random rotation.



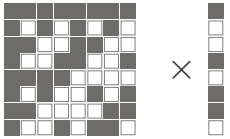
Rotations in Practice

QuaRot (2024): Hadamard on q, k at run time; 4-bit cache.

SpinQuant (2024): learned rotations.

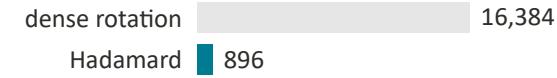
TurboQuant (2025): random rotation; 3.5-bit cache.

RaBitQ (2024): the same device, in vector search.



a Hadamard matrix times
a diagonal of random signs

operations per vector, $d = 128$

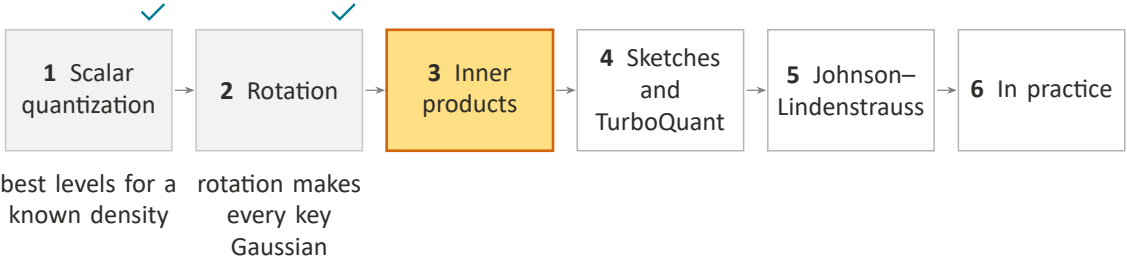


Hadamard: no entry is large.
Exactly Gaussian: a uniformly random rotation.

This is what deployed methods do.

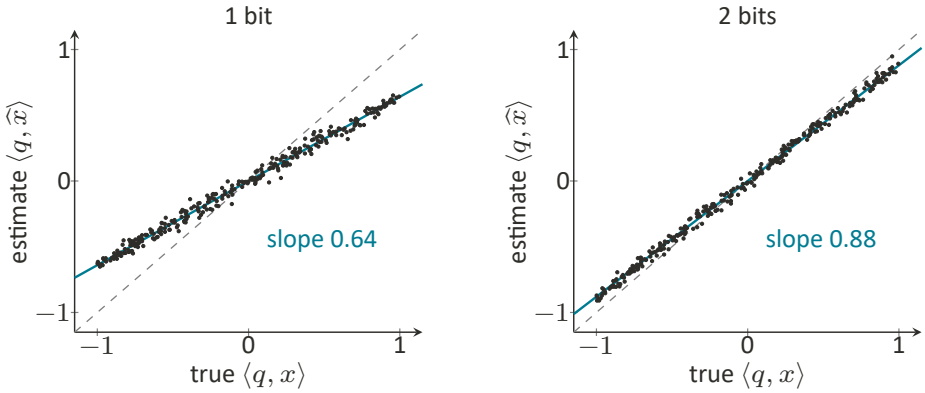
Source: Ashkboos et al. (2024); Liu et al. (2024); Zandieh et al. (2025); Gao and Long (2024).

Where We Stand



We made $\|x - \hat{x}\|_2^2$ small. Is that what attention needs?

An Experiment



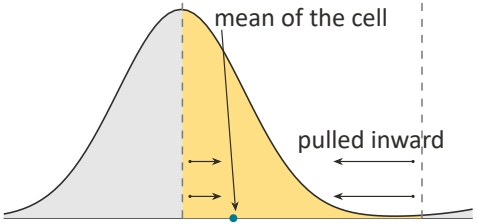
The estimates are shrunk toward zero, all by one factor. Why?

Numbers: simulated: lecture1_slide_numbers.py, $d = 128$, 300 pairs per panel.



Quantizing Creates a Bias

Each level is the mean of its cell, so every point in the cell is replaced by that mean.



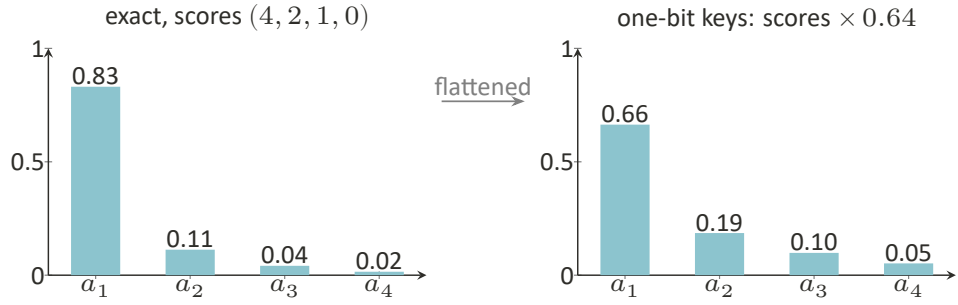
$$\mathbb{E}\langle q, \hat{x} \rangle = (1 - D_b)\, \langle q, x \rangle \,.$$

The spread inside each cell is lost: that is the bias.



Bias Flattens Attention

The resulting bias impacts the attention layer.



How do we remove the bias?



Two Ways to Remove the Bias

Rescale by the known factor: unbiased, more variance.

1 $x \rightarrow$ rotate, quantize: b bits $\rightarrow \widehat{x} \rightarrow$ divide by the shrinkage $\rightarrow$ estimate

Or quantize with $b - 1$ bits and spend the last bit on an unbiased description of what is left.

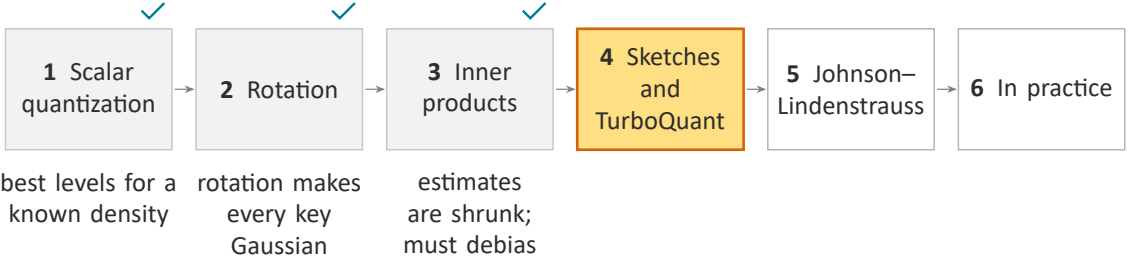
2 $x \rightarrow$ rotate, quantize: $b - 1$ bits $\rightarrow \widehat{x} \rightarrow$ residual $r := x - \widehat{x}$ $\rightarrow$ 1 bit per coordinate, unbiased: ?

The second route needs a new tool.



Roadmap

Six steps to a compressed cache.





Unbiased Inner Products

Keep q in full precision and store x in a few bits.
We want $\langle q, x \rangle$ from what is stored.

An estimate $\widehat{\ell}$ of $\langle q, x \rangle$ is **unbiased** if $\mathbb{E}[\widehat{\ell}] = \langle q, x \rangle$ for every q.

Rotate, then quantize is biased: $\mathbb{E}\langle q, \widehat{x} \rangle = (1 - D_b)\langle q, x \rangle$.

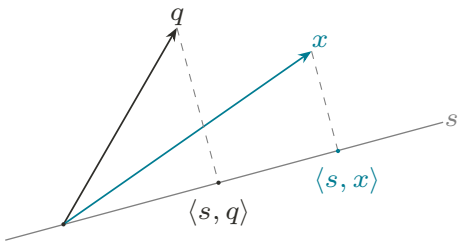
What random measurement of x is unbiased for every q?



Gaussian Vectors to the Rescue Again

Draw $s \in \mathbb{R}^d$ with independent $\mathcal{N}(0,1)$ entries. Then

$$\mathbb{E}[\langle s, q\rangle \langle s, x\rangle] = \langle q, x\rangle .$$



projections on a random direction

Proof. $\mathbb{E}[\langle s, q\rangle \langle s, x\rangle]$
$= q^\top \mathbb{E}[ss^\top]\, x = q^\top x.$



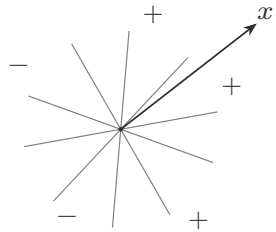
From Numbers to Bits

Zandieh et al. (2024)

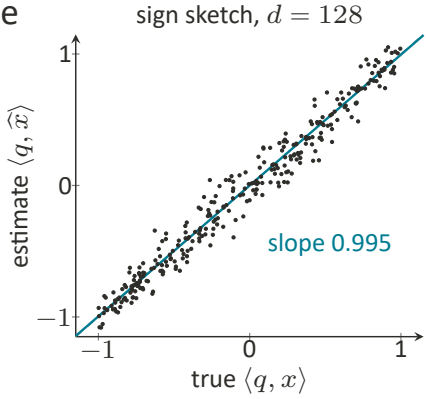
Draw d such vectors, the rows of S.

Keep only the sign of each entry of Sx: one bit for each dimension.

With q in full precision, $\langle q, x \rangle$ can be estimated without bias (QJL).



a *sketch*: a short random summary of x

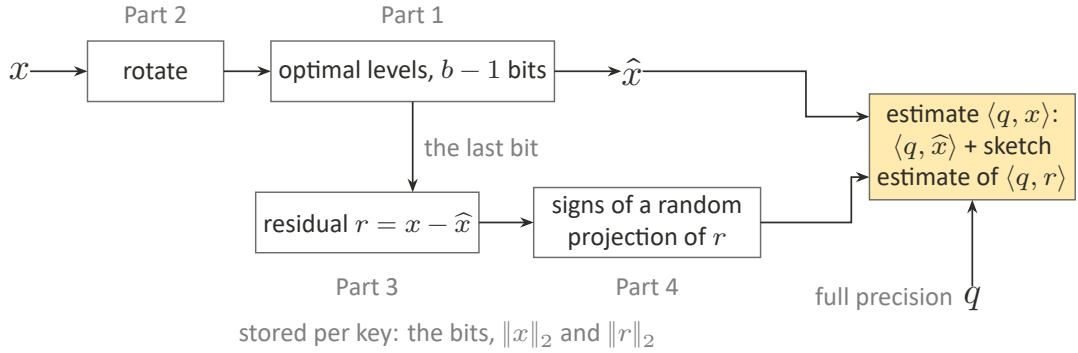


Numbers: simulated: lecture1_slide_numbers.py, $d = 128$, 300 pairs.

TurboQuant

Zandieh et al. (2025); Gao and Long (2024)

b bits per coordinate: $b-1$ for the quantizer, 1 for the residual.



Unbiased for every key and query; the error falls by four per bit.

What About the Values?

Values enter attention only through the output, a weighted average:

$$o := \sum_i a_i \, v_i \, .$$

Its weights are known when the values are read: no unseen query.
So keep each v_i close in squared error.

Rotate, then quantize shrinks each value: $\mathbb{E}[\hat{v}_i] = (1 - D_b)\, v_i$.

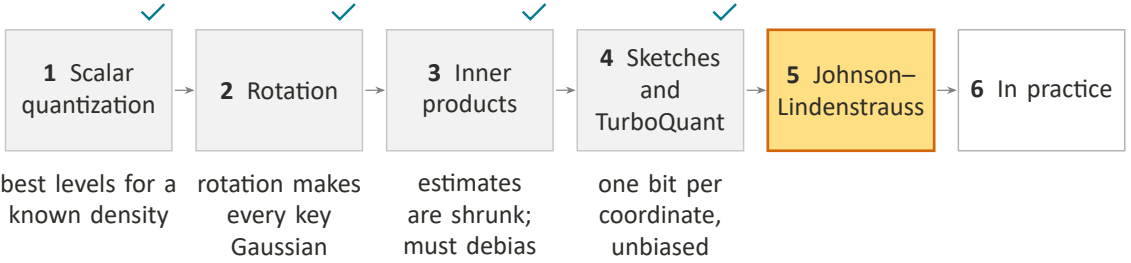
One known factor, which can be undone for free. Values have no outlier channels (KIVI).

Values need small squared error, not unbiased inner products.



Roadmap

Six steps to a compressed cache.

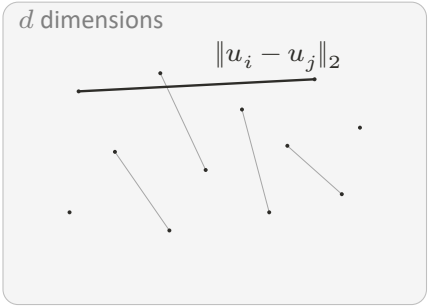


TurboQuant Is Inspired by the Johnson–Lindenstrauss Lemma



How Many Dimensions for Distances?

n vectors in d dimensions, with their distances and inner products. How many dimensions are needed to approximately preserve those?



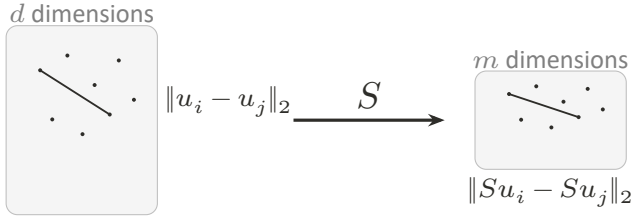
- ☐ (a) d
- ☐ (b) about $\sqrt{d}$
- ☐ (c) a number that does not depend on d

The answer does not depend on d.



$O(\log n)$ Dimensions Are Enough

Any n vectors, in any dimension d: about $\log(n)/\varepsilon^2$ numbers each keep all pairwise Euclidean distances within a factor $1 \pm \varepsilon$.



The dimension depends on n and ε, not on d.



The Johnson–Lindenstrauss Lemma

Multiply every vector by one random $m \times d$ Gaussian matrix S.

If $m \geq C \log(n)/\varepsilon^2$, for an absolute constant C, all pairwise distances are preserved within a factor $1 \pm \varepsilon$, with high probability.

$$(1-\varepsilon)\,\|u_i - u_j\|_2^2 \leq \|Su_i - Su_j\|_2^2 \leq (1+\varepsilon)\,\|u_i - u_j\|_2^2\,.$$

From here on, S has m rows, not d, and entries $\mathcal{N}(0, 1/m)$.



Why the Lemma Holds

One vector. For a fixed unit u, $\|Su\|_2^2$ is an average of m independent terms with mean 1, so

$$\mathbb{P}\big(|\|Su\|_2^2 - 1| > \varepsilon\big) \le 2e^{-m\varepsilon^2/8}\,.$$

All pairs. Apply it to each normalized $u_i - u_j$ (S is linear); there are fewer than $n^2/2$. By the union bound, all succeed with probability $1-\delta$ once

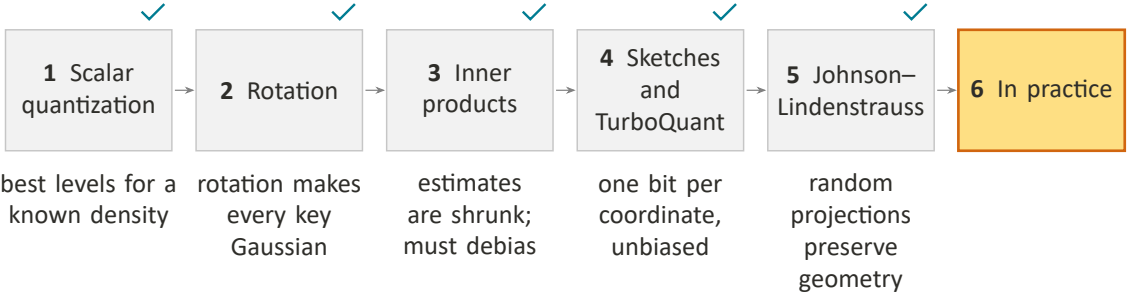
$$m \ge \frac{8\,(2\ln n + \ln(1/\delta))}{\varepsilon^2}\,.$$

An average concentrates, and a union bound pays only $\log n$.



Roadmap

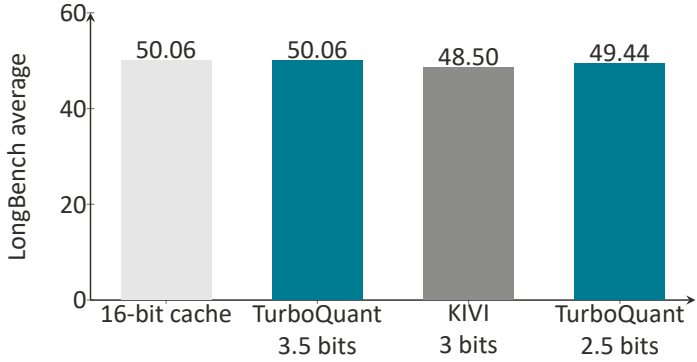
Six steps to a compressed cache.



Does it work, and what do deployed systems use?

TurboQuant: Reported Results

LongBench, a long-context benchmark, on Llama 3.1 8B.



No measured loss at 3.5 bits: a cache 4.6 times smaller.

Fractional rates: some channels get one more bit.

Source: Zandieh et al., TurboQuant (2025), Table 1 (Llama-3.1-8B-Instruct).



Why 3.5 Bits?

The rate is an average over the 128 channels of a key:
outlier channels get one bit more than the others.

For example, 64 channels at 4 bits and 64 at 3 bits:

$$\frac{64 \times 4 + 64 \times 3}{128} = 3.5\,.$$

an illustration: the paper does not give its 3.5-bit split

Two stored lengths per key add $2 \times 16/128 = 0.25$ bit per coordinate.

3.5 bits is an average over channels, not one grid.



A Family of Methods

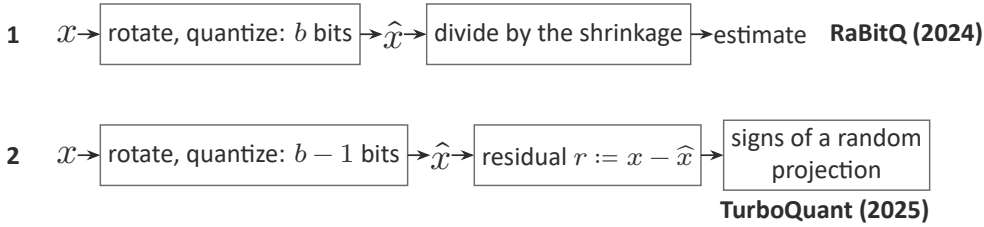
Gao and Long (2024); Zandieh et al. (2025)

Rotate, quantize each coordinate, correct the inner product.

RaBitQ (2024), from vector search, corrects by rescaling.

TurboQuant corrects with a residual sketch.

The comparison between them is disputed.

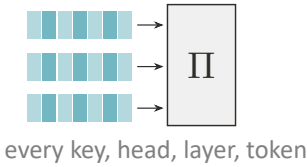


Which correction is better? Homework.

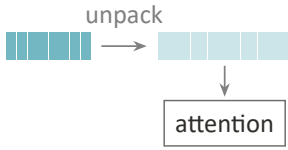


Reality Check

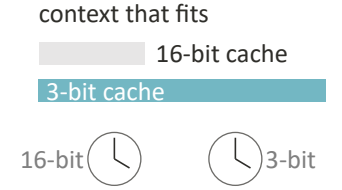
Deployed systems often use 8 bits, not 3.



the rotation is
paid on every key



codes must be unpacked
before attention

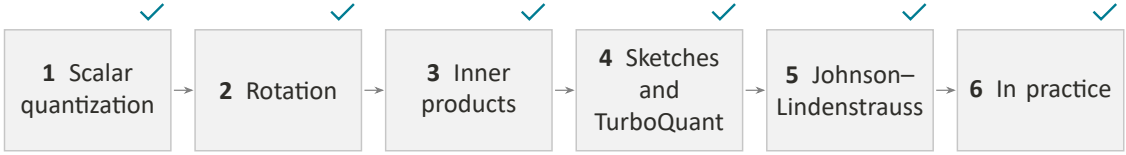


a smaller cache buys
longer contexts,
not faster tokens

More research is needed!

Summary

Few bits per key, at once, no data, inner products kept.





Summary

Five results to remember

1. Best levels: thresholds at midpoints, levels at centroids.
2. A random rotation makes every key's entries nearly Gaussian.
3. Error-optimal levels shrink inner products by $1 - D_b$.
4. Signs of random projections estimate $\langle q, x \rangle$ without bias.
5. About $\log(n)/\varepsilon^2$ dimensions keep the distances of n points.



Next Lecture

	weights Lecture II	activations Lecture II	keys and values today
When is it known?			at run time, then kept
Time to encode it?			almost none: once per token
What must be preserved?			inner products (keys); averages (values)
What do values look like?			outliers in a few key channels
How does hardware use it?			read at every later step

Weights come with sample inputs. Distortion of what?



Questions?

Next: Lecture II, quantizing matrices.

