IQR-V4: Measured Widths, a Vocabulary-Weighted Head and Faster Kernels

Technical report on IQR-V4 for Qwen3.8-27B

Deadcode
radiance project

November 2026

Abstract

IQR-V4 is the fourth format of the IQR series for Qwen3.8-27B on AMD R9700 cards. It keeps V3's computed trellis code and its one f32 scale a row, and changes where the bits go and how the kernels read them. GPTQ runs in act order inside the column trellis; the projections fed by the attention heads take a bf16 scale a (row, 128) block from five bits up; the trunk's widths come from measured KL prices with one per-tensor exception, layer 63's `ffn_gate_up`; and the lm_head, a tenth of every entry's bytes, becomes vocabulary-weighted: four width classes of the computed trellis, the tokens that carry the probability on the widest, decoded in one launch that is faster than the int8 head it replaces. At the top of the ladder the int8 head is GPTQ'd instead of rounded. On the board's private split every V4 entry is ahead of its V3 twin of the same size, by 10–19% in KL from 2.5 to 5.4 bits a weight and 2–8% above; IQ4R-L-V4 is 24% below the official FP8 checkpoint at 55% of its size, and IQ2R-L-V4, at 2.8 bits, is below the 5-bit MXFP4 checkpoint. The kernels change as much. The one-token GEMV gives each weight tile a block whose waves split the input dimension, a 16-token tile without block barriers serves two to sixteen tokens, and three activation quantisers fold into the kernels that produce their input. Served end to end on one card, IQ4R-S-V4 decodes one stream 4% faster than the MXFP4 checkpoint at five times lower KL, and IQ3R-S-V4, still 38% below MXFP4's KL, is 4–20% faster at one to eight streams.

1 Background

IQR [1, 2, 3] quantises each linear layer of Qwen3.8-27B on one grid. The weight is rotated by a 128-point Walsh–Hadamard transform, every value is an int8 code times a scale, and the activations are rotated and quantised to int8 per (token, 128 inputs), so one int8×int8 matrix core serves every size. Codes are fitted by GPTQ [4] against calibration Hessians, and every format stays at its codes' width in VRAM: the GEMM decodes each weight where it reads it. IQR-V2 made the codes a tail-biting trellis [5, 7] down each column inside GPTQ; IQR-V3 computed a state's value from the state, a multiply-add, a mask and a byte sum, instead of reading a table, grew the trellis to 2^{16} states and replaced the per-block scales by one f32 a row.

V3 left three things on the table, and V4 is about them.

- **The bits were spent by a model.** V3's allocator priced each band of layers with a fitted model of how KL grows as a width falls. Measured, the prices at the 2/3-bit boundary differ from the model by up to 1.9× in either direction, and one tensor, layer 63's `ffn_gate_up`, costs a third of its whole band.
- **The vocabulary edges.** The embedding and the lm_head are about 10% of every entry's bytes. V3 stored the head as a 6-bit table form or as int8, the same width for all 248 320 rows, although the head's share of the KL weighs each row by its token's probability.
- **The GEMMs at decode.** V3's one-token GEMV gave each wave one 16-row tile over the whole input, so a GEMM was a few long waves spread unevenly over the card, and its end ran with most of the card idle. From two to sixteen tokens the narrow tile synchronised its block twice per unit, and three of the trunk's activation quantisers ran as kernels of their own, 128 launches a decode step.

2 The format

2.1 GPTQ in act order inside the trellis

V3 coded the columns in their stored order. V4 runs GPTQ in act order (`tcq_order=1`): the columns are coded in descending order of the Hessian's diagonal, so the error of the columns that matter most is fed forward into the

many that follow, and the codes are stored back in column order, so the kernels read the same layout. A row's trellis states run down the column, across the rows, so the column order does not change the code; it changes only GPTQ's feedback. On the FFN weights the tail-biting runs grow from 256 to 512 rows (`ffn_gate_up`) and to 1024 where the card's Viterbi stays fast enough (`tcq_rows`). On uniform bases of three to six bits these lower the KL by 0–2.3%; the first V4 entries, with the allocation below still V3's, were 0–7% below their V3 twins.

2.2 The head-fed projections

The attention output and the delta net's output projection (`attn_output`, `ssm_out`, 8% of the trunk) read the heads' outputs, whose channels differ in scale by more than the rotated FFN inputs do. V3 gave them a larger scale gain from five bits up and still saw them reach the code's resolution first. V4 gives them back a scale per (row, 128 inputs), bf16, from five bits up (`group=128 scale=bf16 dyn=1`), at 0.125 bpw on 8% of the weights: 0.010 bpw. On the board that lowers an entry's KL by 1–6% at four to six bits. On every other projection the same scale buys 5% of the excess for its 0.125 bpw, where a width step buys 16%, so they keep one scale a row.

2.3 Measured prices and one exception

The allocator splits every band of eight layers into its units (FFN up, FFN down, the attention inputs and the head-fed outputs) and puts each unit at a width, minimising the summed price at the entry's size. V4 measures the prices instead of modelling them: for the 2→3, 3→4 and 4→5-bit steps it quantises one unit at a time one width down over a uniform base and reads the KL on a selection corpus. Thirty-two units' measured prices predict the all-2-bit trunk to 2%, so the units add without strong compounding. The prices are noisy near 10^{-3}, so they are shrunk halfway (geometrically) toward the model's.

A single-tensor scan at the 2/3-bit boundary found one outlier: layer 63's `ffn_gate_up` costs $7.5 \cdot 10^{-3}$ KL at two bits, 35% of its whole band and five times any neighbour. V4 keeps that one tensor one width above its band (22 MB, 0.007 bpw). Table 1 shows the two changes at the low end of the ladder; from IQ4R-S up, where a unit's price is $\sim 3 \cdot 10^{-4}$ and inside the corpus' noise, the model's prices do as well and are kept.

entry	bpw	model prices	+ L63 one width up	measured prices + L63
IQ2R	2.47	0.06080	**0.05992**	0.06144
IQ2R-L	2.80	0.03941	–	**0.03682** (−6.6%)
IQ3R-S	3.16	0.02437	0.02337	**0.02241** (−8.0%)
IQ3R	3.47	0.01570	**0.01548**	0.01576
IQ3R-L	3.80	0.00991	–	**0.00921** (−7.1%)

Table 1. KL on the board's first private tier (t1) at the low end, all with the vocabulary-weighted head. Held out on the second tier the IQ2R-L, IQ3R-S and IQ3R-L gains are −5.5%, −6.4% and −4.2%. The ladder takes the best column of each row.

2.4 A vocabulary-weighted lm_head

The head's contribution to the KL is second order in the logit error, and each vocabulary row enters it weighted by its token's probability, $\frac{1}{2}\sum_v \mathbb{E}[p_v\, \delta z_v^2]$. On calibration text 90% of the probability mass sits on 15.8K of the 248K tokens, and a row at rank 128K weighs a thousand times less than one at rank 4K. A uniform head spends the same bits on both.

V4's head stores each row on one of four widths of the computed trellis, each width a class of its own weight (`output.kB`), with one f32 scale a row. The classes are chosen per run of 256 rows: the runs are sorted by their tokens' mean probability, and each takes the width that minimises its rows' output error $\delta W H \delta W^\top$, weighted by those probabilities, at a mean of 4.0, 4.5 or 5.0 bits (3–6, 4–7 and 4–7 bits). A width floor is necessary: with free widths the rarest rows fall to two bits and turn into false positives wherever the hidden state is large, a cost that grows like $e^{\delta z}$, not δz^2. Each half of the vocabulary is allocated on its own, so at two-card tensor parallelism both ranks hold the same rows of every class and the classes shard with the vocabulary.

`logits_gemm_vocab` decodes all classes in one launch, each block decoding its class's codes into int8 fragments. The head's input is rotated and held at 16 bits as two int8 planes, two matrix passes: int8 activations would cost the head $2.1 \cdot 10^{-5}$ KL, half its quantisation cost. Logits are written in vocabulary order: up to 32 tokens through a staging plane and a gather, above that at each row's id, the rows sorted by id within a class (a random-order store

cost 120 µs at 8 tokens). Table 2 gives its time. At DFlash's 8-token verification it saves 0.43 ms a step against the int8 head.

tokens	1	8	16	32	64
vocabulary-weighted head (4.4 b)	624	635	661	884	1557
int8 head	1020	1064	1076	1109	1410
6-bit table head (V3)	1133	1417	1444	1664	2218

Table 2. The lm_head of one rank (124 160 rows × 5120), µs.

On the board the weighted head is worth about a bit and a half of head at equal KL: at 5.0 b it is within $5 \cdot 10^{-5}$ of the int8 head's KL at 3.1 b less, and the 4.0–5.0 b heads lower the KL of the IQ4R-S to IQ6R-S entries by 3–10% against the same trunks with V3's heads.

2.5 The top of the ladder

From IQ6R-L up the entries keep V3's allocation (late bands on the int8 grid) on V4's bases, and an int8 head: the head's input is no longer the bottleneck there, and a weighted head's savings have nowhere better to go. That head is now GPTQ'd against its input Hessian instead of rounded to nearest. It costs no bytes; the mean KL falls 2–3% and the median 10–11%, and the held-out tier agrees (−2.4% mean, −10.5% median at IQ6R-L).

3 The kernels

3.1 One token: a block a weight tile

V3's GEMV gave each wave one 16-row tile and all of K. A GEMM was then a few hundred to a thousand long waves spread unevenly over the 64 compute units, and its last waves ran with most of the card idle. V4's GEMV gives each 16-row tile a block of four waves (eight for weights under 4096 rows), the waves striding K and their partial sums reduced in LDS, so every tile finishes at once and the card is full until the last round. On the 5120×6144 shape (the attention's q and gate) at one token that is 42.3 µs to 30.0 µs.

Three smaller changes cut the vector work per weight. The block's int32 partial sums accumulate into an f32 slot that starts at the bias $1.5 \cdot 2^{23}$, so the slot read as a float is the exact sum and needs no conversion (flushed every eight blocks, before the integer could leave the float's exact range); the activation byte is broadcast to the four bytes of a register by one `v_perm` instead of a multiply; and the 126 the code carries is subtracted by one `v_dot4` against `0x7E7E7E7E` instead of a quarter-rate multiply. The decoder also builds each column's code stream once per unit instead of once per row group, which alone takes 7.5% off the 3-bit form.

3.2 Two to sixteen tokens

From two tokens (one token for six- and seven-bit codes, where the GEMV runs out of registers) the computed trellis now leaves the GEMV for a 16-token WMMA tile of the same shape, a block a weight tile with its waves across K. Each wave decodes its own unit, transposes it into fragment order through its own LDS tile and reads the activation fragments straight from memory; the block meets only once, for the final reduction, which reuses the transpose tiles. With four streams that is 18.6% off IQ4R-S's step and 21.6% off an all-3-bit trunk's, end to end.

3.3 Quantisers folded into their producers

IQR's trunk reads int8 activations rotated by the 128-point transform. Three of them were produced by a kernel and then rotated and quantised by another: the gated product after `ffn_gate_up`, the delta net's convolution-and-recurrence output at decode, and the gated attention's gate. V4 folds each quantiser into its producer, the last block to finish a 128-column group running the transform and writing the codes and the scale; each fused kernel writes the same bytes as the pair it replaces. That is 64 + 48 + 16 launches a decode step, about 0.5 ms.

3.4 Prefill

At 2048 tokens the decode happens once per weight and the time is the int8 tile's. V4's prefill path is V3's: one f32 scale a row folds like FP8's block scale. The head-fed projections' bf16 (row, 128) scales fold per row and block, so a

V4 layer at five to seven bits takes 1–2% more than V3's at the same width (Table 3). Against FP8 the trunk is 2–8% behind and the int8 grid 19%.

	V2		V3		V4	
k	µs	/FP8	µs	/FP8	µs	/FP8
2	4630	1.247	3801	1.024	3801	1.024
3	4637	1.249	3876	1.044	3876	1.044
4	4636	1.249	3897	1.050	3943	1.062
5	4644	1.251	3932	1.059	3987	1.074
6	4667	1.257	3927	1.058	3990	1.075
7	4695	1.265	3954	1.065	4011	1.081
FP8 3712 µs (1.000), int8 grid 4419 µs (1.190)						

Table 3. Trunk GEMM time of one decoder layer at 2048 tokens, every projection at k bits, one rank of two, on V4's kernels (V4: the head-fed projections in their bf16 (row, 128) form from four bits).

3.5 Decode sizes

Table 4 times the 4-bit form against FP8's GEMM, the same container's weight on V3's kernels and on V4's, one rank of two, the median of seven interleaved rounds on one card. On V3's kernels the narrow forms lost their lead as the tokens grew: a 4-bit `ffn_down` took 0.85–0.87 of FP8's time at 8 and 16 tokens, and the small `ssm_out` shape was at or above FP8 from four tokens. On V4's kernels the time hardly moves from one token to sixteen, 0.52–0.54 of FP8 on `ffn_gate_up`, 0.55–0.57 on `ffn_down` and 0.61–0.66 on `ssm_out`'s shape: from four tokens up V4 is 1.15–1.26× as fast as V3 on `ffn_gate_up`, 1.39–1.53× on `ffn_down` and 1.48–1.74× on `ssm_out`. At one token the 2-bit form takes 0.32 of FP8's time on `ffn_gate_up` (V3 0.41); the head-fed bf16-scale form at five bits 0.67 (the f32-row form 0.64).

	`ffn_gate_up`			`ffn_down`		`ssm_out`	
M	FP8 µs	V3	V4	V3	V4	V3	V4
1	145.6	0.56	**0.52**	0.64	**0.55**	0.89	**0.61**
2	146.0	0.59	**0.53**	0.67	**0.56**	0.97	**0.65**
4	146.4	0.67	**0.53**	0.78	**0.56**	1.13	**0.65**
8	147.6	0.62	**0.54**	0.85	**0.57**	0.98	**0.66**
16	148.7	0.62	**0.54**	0.87	**0.57**	0.98	**0.65**

Table 4. The 4-bit computed trellis at M tokens, its time relative to FP8's, on V3's kernels and V4's (one rank: `ffn_gate_up` 17 408×5120, `ffn_down` 5120×8704, `ssm_out` 5120×3072).

4 Results

4.1 The ladder

Each entry is a mix of V4's per-width bases chosen at its V3 twin's size, scored on the board's private split, 1.05M positions the model writes itself, against the bf16 model's top-512 distribution (Table 5, Figure 1). Every V4 entry is ahead of its V3 twin: by 10–19% in mean KL from IQ2R to IQ5R, 6–8% at six bits, and 2% at seven and eight, where most of the trunk is on the int8 grid both formats share and the GPTQ'd head is the only change. The median moves more than the mean: −15 to −26% from IQ2R to IQ5R, and −8 to −12% from six bits up. IQ4R-L-V4 is 24% below the official FP8 checkpoint's KL (0.00389 at 8.77 bpw) at 55% of its size, and IQ2R-L-V4, at 2.8 bpw, is below the AWQ-tuned MXFP4 checkpoint (0.03824 at 4.99 bpw). Against EXL3-V2, a submitted ladder of EXL3's trellis [6] with per-tensor widths fitted to measured KL, V4 is ahead from 2.5 to 5 bpw (IQ4R-V4 0.00444 at 4.47 against 4.50 bpw's 0.00480), level at 5.4–5.5 (0.00154 against 0.00153), and behind from six bits up, by about 10% at six and 35–50% above: there the int8 code grid, not the allocation, is IQR's limit. The GEMM column is the entry's trunk GEMM time a decoder layer at 2048 tokens on V4's kernels (Table 3): V4's head-fed bf16 scales cost 0–3% against V3's mix of the same size.

			IQR-V4			vs V3			GEMM, µs	
entry	GB	bpw	KL	median	top-1	KL	Δ	Δ med.	V4	V3
IQ2R	8.31	2.470	0.06113	0.014693	91.23%	0.06807	−10%	−17%	3829	3829
IQ2R-L	9.40	2.797	0.03602	0.008274	93.29%	0.04241	−15%	−18%	3857	3857
IQ3R-S	10.62	3.157	0.02241	0.004972	94.75%	0.02584	−13%	−25%	3879	3879
IQ3R	11.67	3.472	0.01495	0.003057	95.74%	0.01698	−12%	−19%	3887	3885
IQ3R-L	12.79	3.804	0.00917	0.001928	96.61%	0.01129	−19%	−25%	3894	3892
IQ4R-S	13.93	4.142	0.00643	0.001369	97.12%	0.00762	−16%	−15%	3941	3900
IQ4R	15.04	4.473	0.00444	0.000859	97.69%	0.00512	−13%	−19%	3963	3914
IQ4R-L	16.11	4.793	0.00297	0.000616	98.07%	0.00342	−13%	−18%	3972	3925
IQ5R-S	17.05	5.071	0.00223	0.000465	98.33%	0.00249	−10%	−19%	3985	3930
IQ5R	18.27	5.433	0.00154	0.000312	98.60%	0.00182	−15%	−26%	3991	3930
IQ6R-S	20.38	6.063	0.00100	0.000210	98.86%	0.00106	−6%	−10%	4043	3931
IQ6R	21.65	6.439	0.00081	0.000173	98.96%	0.00088	−8%	−11%	3972	3952
IQ6R-L	23.33	6.939	0.00068	0.000143	99.08%	0.00072	−6%	−8%	4006	4006
IQ7R	24.95	7.420	0.00059	0.000120	99.15%	0.00060	−2%	−10%	4171	4171
IQ8R	27.03	8.040	0.00046	0.000106	99.22%	0.00047	−2%	−12%	4419	4419

Table 5. The IQR-V4 ladder on the board's private split (t1 and t2, 1.05M model-written positions), against its V3 twin of the same name; IQ6R-V4 is 0.07 bpw larger than IQ6R-V3. FP8's GEMM time is 3712 µs a layer.

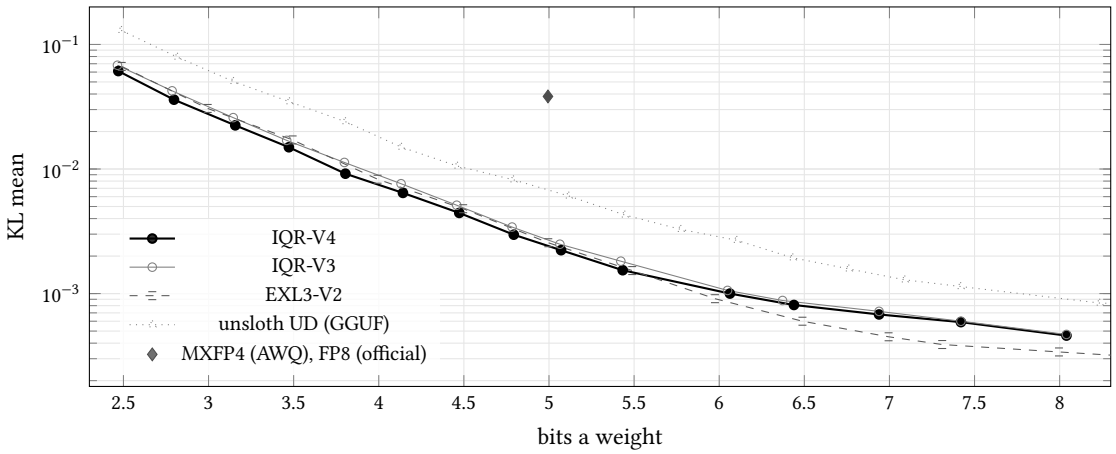


Figure 1. Mean KL against size on the board's private split.

4.2 Against MXFP4, end to end

The production container of the same model is AMD Quark's AWQ-tuned MXFP4 (4.99 bpw on disk, KL 0.0363 on t1). Table 6 serves it and two V4 entries with the same server at one-card tensor parallelism, the production flags without speculation, distinct 13K-token prompts at one to eight concurrent streams.

container	t1 KL	c1	c2	c4	c8	prefill tok/s
MXFP4 (Quark AWQ)	0.0363	25.85	26.44	27.81	31.29	3367
IQ4R-S-V4, V3 kernels	0.0067	27.63	27.95	29.38	32.56	3067
IQ4R-S-V4	0.0067	**24.73**	27.12	28.42	**31.22**	3073
IQ3R-S-V4, V3 kernels	0.0224	25.06	27.03	28.42	30.67	3099
IQ3R-S-V4	0.0224	**20.64**	**24.25**	**26.60**	**29.00**	3113

Table 6. Decode, ms a token a stream at c concurrent streams, and prefill, on one R9700. "V3 kernels": the same container on the GEMV and tile before Section 3.

IQ4R-S-V4 is smaller than MXFP4 (4.14 against 4.99 bits a weight) at five times lower KL. It decodes one stream 4.3% faster, eight streams level, and two and four streams 2–3% slower: at two to eight tokens V4's tile is about 3% slower per GEMM than MXFP4's GEMV, and its 3-bit layers run no faster there than its 4-bit ones. IQ3R-S-V4 is faster at every concurrency. Prefill is 8–9% behind: both tiles run at the card's 300 W limit, at the same clock, so their time follows their energy; IQR's int8 K loop moves twice the weight bytes of MXFP4's 4-bit tile (about 5%), and the decode into the int8 plane costs about 5% more.

5 What did not work

Most of V4's research rounds closed with numbers. Every reweighting of the Hessian lost to the plain calibration Gram: Fisher row and layer allocation, YAQA-style sketched Hessians, end-to-end row factors and uncertainty weights all followed the heavy tail of per-token gradient energy (10 of 65K tokens carry 30–60% of it). Gradient-free row allocation, finer activation groups, static smoothing, random-sign incoherence (the rotated weights are already Gaussian, excess kurtosis −0.05), neuron pruning, a conditional logit bias, a logit temperature in the head's scales, GPTQ damping below 0.01, a scale refit after GPTQ, quantisation-error propagation and 16-bit trunk activations each moved the KL by 0–2% or not at all. A 16-bit trunk, the one large lever left at the top, would double the prefill tile's matrix work and stays out of the format: from 6.5 bits up the int8 code grid, not the trellis, is IQR's limit.

6 Summary

IQR-V4 spends the bits where they were measured to pay: act-order GPTQ and longer runs, a block scale on the 8% of weights fed by the attention heads, widths from measured prices with one exception for layer 63, and a head whose rows get their width by their token's probability. Every entry of the ladder is ahead of its V3 twin, by 10–19% in KL up to 5.4 bits and 2–8% above, and V4 leads EXL3-V2 from 2.5 to 5 bits. The kernels close most of the decode gap V3 left: the GEMV fills the card, two to sixteen tokens run on a tile without block barriers, and three quantisers no longer launch on their own, so IQ4R-S-V4 decodes one stream faster than the MXFP4 container at a fifth of its KL. What is left is prefill, 8–9% behind MXFP4's tile at the power limit, and the int8 grid at the top of the ladder.

References

[1] Deadcode. IQR: GPTQ integer quantisation on a rotated int8 grid. Technical report on IQR-V1, radiance project, 2026.

[2] Deadcode. IQR-V2: trellis-coded GPTQ columns on a rotated int8 grid. Technical report on IQR-V2, radiance project, 2026.

[3] Deadcode. IQR-V3: a computed trellis code under one scale a row. Technical report on IQR-V3, radiance project, 2026.

[4] E. Frantar, S. Ashkboos, T. Hoefler, D. Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. ICLR, 2023.

[5] M. W. Marcellin, T. R. Fischer. Trellis coded quantization of memoryless and Gauss–Markov sources. IEEE Trans. Commun. 38(1):82–93, 1990.

[6] turboderp. ExLlamaV3 and the EXL3 format. github.com/turboderp-org/exllamav3, 2025.

[7] A. Tseng, Q. Sun, D. Hou, C. De Sa. QTIP: Quantization with trellises and incoherence processing. NeurIPS, 2024.