SINQ: Sinkhorn-Normalized Quantization

Lorenz K. Muller Philippe Bich Jiawei Zhuang Ahmet Celik Luca Benfenati Lukas Cavigelli

Computing Systems Lab, Huawei Technologies, Switzerland · ICML 2026 · github.com/huawei-csl/SINQ

The Problem

A fast, calibration-free quantizer that closes the gap to calibrated & non-uniform methods, by adding one extra scale vector.

Post-training quantization (PTQ) is the standard way to deploy LLMs cheaply. But at $\leq$ **4 bits** perplexity degrades: a few *outlier* weights share a scale with their whole row/column, wrecking precision for everyone else.

The cleanest, fastest recipe (**calibration-free, uniform** quantization) suffers most. Calibrated (AWQ/GPTQ) and non-uniform (NF4) methods help, but add quantization-time cost, overfitting risk, and inference look-ups.

Contributions

- Column-wise weight std. dev. *predicts* input activation scales $\Rightarrow$ **calibration-free pseudo-activation-awareness**.
- A **dual-scale** parameterization + a fast **Sinkhorn–Knopp** algorithm that balances rows & columns.
- Beats SoTA calibration-free baselines; composes with calibration (AWQ) and non-uniform levels (NF4); **negligible overhead**, drop-in for any linear layer.

Key Idea: Dual-Scale Quantization

A single scale vector cannot isolate a 2-D outlier. With scales along **both** axes we trade off error between the outlier's row and column:

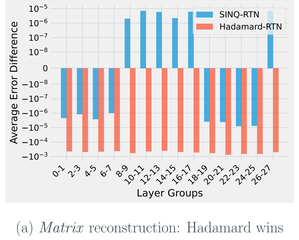
Single scale: $\mathbf{W} \approx \vec{s} \odot (\mathbf{Q} + \vec{z})$ **Dual scale (ours):** $\mathbf{W} \approx \vec{s} \odot (\mathbf{Q} + \vec{z}) \odot \vec{t}$

Concretely, for a large outlier W_{ij}, raising the row scale s_i and lowering the column scale t_j shrinks its quantization error and **redistributes the slack** across row i and column j, a degree of freedom that single-scale quantization simply does not have.

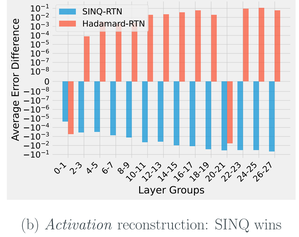
The second scale $\vec{t}$ costs only M extra parameters and, at inference, folds into the **activations** ($\vec{x} \odot \vec{t}$).

Weights vs. Activations

SINQ minimizes *activation* error, not *weight* error: the opposite of Hadamard rotation, and what matters end-to-end.



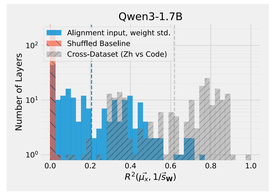
(a) *Matrix* reconstruction: Hadamard wins



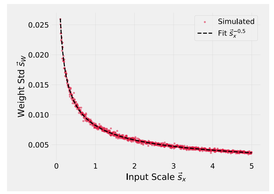
(b) *Activation* reconstruction: SINQ wins

Why It Works: Pseudo-Activation-Awareness

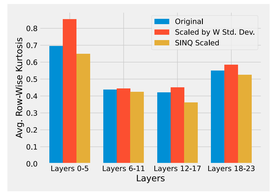
The reciprocal column-wise weight std. dev. is a strong predictor of the *average input magnitude* $\vec{\mu}_x$, exactly the quantity AWQ needs calibration data to estimate. We recover it **for free** from the weights.



(a) High R^2 between $\vec{\mu}_x$ and $1/\vec{s}_\mathbf{W}$



(b) Adam training drives $\vec{s}_\mathbf{W} \propto |\vec{s}_x|^{-1/2}$



(c) SINQ avoids new row outliers

$$\vec{s}_\mathbf{W} \propto \frac{1}{\sqrt{|\vec{s}_x|}} \qquad \text{(weight updates} \sim \text{outer product of inputs \& gradients)}$$

Naively scaling by $1/\vec{s}_\mathbf{W}$ creates *new* row outliers (panel c, red). SINQ scales columns toward activation-awareness while keeping rows tame.

The SINQ Algorithm

A dampened **Sinkhorn–Knopp** iteration rescales rows and columns toward a common target std. dev. τ, minimizing the **matrix imbalance** I before round-to-nearest:

$$I(\mathbf{W}) = \frac{\max\left(\max_i \sigma_i^{\text{row}},\ \max_j \sigma_j^{\text{col}}\right)}{\min\left(\min_i \sigma_i^{\text{row}},\ \min_j \sigma_j^{\text{col}}\right)}$$

- Multiplicative scales $\vec{s}, \vec{t}$; the clamp damps each step to avoid overshoot.
- Snapshot the scales achieving the **lowest imbalance**.
- Apply any standard quantizer (INT4 / NF4 / ...) to the balanced matrix.
- Recover $\vec{s}, \vec{t}$; **$\sim$1.1$\times$ RTN runtime** (vs. HQQ >2×, AWQ >30×).

SINQ Normalization

In: $\mathbf{W} \in \mathbb{R}^{m \times n}$, iters K, bits b, clamp $[c_-, c_+]$
Out: $\mathbf{Q}$, $\vec{z}$; scales $\vec{s} \in \mathbb{R}^m$, $\vec{t} \in \mathbb{R}^n$

$\tau \leftarrow \min\left(\min_i \sigma_i^{\text{row}}, \min_j \sigma_j^{\text{col}}\right)$ *target*
$\vec{s} \leftarrow \mathbf{1}_m, \quad \vec{t} \leftarrow \mathbf{1}_n, \quad I^\star \leftarrow \infty$
for $k = 1, \ldots, K$:
  $\hat{\mathbf{W}} \leftarrow \mathbf{W} \oslash \vec{s} \oslash \vec{t}$ *apply scales*
  if $I(\hat{\mathbf{W}}) < I^\star$: $I^\star \leftarrow I(\hat{\mathbf{W}})$, $\vec{s}^\star, \vec{t}^\star \leftarrow \vec{s}, \vec{t}$
  $\vec{t} \leftarrow \vec{t} \cdot \text{clamp}_{[c_-, c_+]}(\sigma^{\text{col}}(\mathbf{W})/\tau)$
  $\vec{s} \leftarrow \vec{s} \cdot \text{clamp}_{[c_-, c_+]}(\sigma^{\text{row}}(\mathbf{W})/\tau)$
$\hat{\mathbf{W}} \leftarrow \mathbf{W} \oslash \vec{s}^\star \oslash \vec{t}^\star$
$\mathbf{Q}, \vec{z}, \vec{s}_q \leftarrow \text{RoundToNearest}(\mathbf{W}, b)$
return $\mathbf{Q}$, $\vec{z}$, $\vec{s}_q \cdot \vec{s}^\star$, $\vec{t}^\star$

Fast & (Almost) Free at Inference

End-to-end decode (SGLang, W4A16 speedup over FP16):

	Qwen3-14B	Qwen3-32B
FP16	48 tps	21 tps
AWQ	2.4×	2.9×
SINQ	2.4×	2.8×

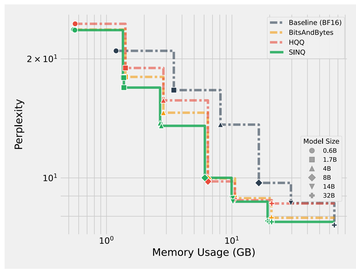
Kernel overhead of the 2nd scale (gemlite W4A16):

B	D	OH
1	1024	1.8%
1	2048	1.5%
64	1024	0.8%

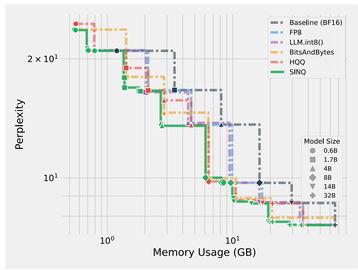
The no-overhead variant folds $\vec{t}$ into preceding layers for **identical** inference speed, and also improves GGUF / `llama.cpp` accuracy at full speed.

Results: Memory–Perplexity Pareto

Because Qwen3 ships in many sizes, we can show the bf16 Pareto front across the whole memory range, from $\sim$1.5 GB to 65 GB. At equal memory **SINQ** matches or beats calibration-free baselines almost everywhere. Panel (b) adds 6- and 8-bit points, with `LLM.int8()` as a reference.



(a) 4-bit methods



(b) Mixed 4/6/8-bit

Perplexity (WikiText2) ↓

Qwen3, uniform PTQ. Original BF16: *16.67 / 8.64 / 7.60* (1.7B/14B/32B).

	Calibration-free	1.7B	14B	32B
3-bit	RTN	32.43	10.50	30.78
	HQQ	32.10	10.73	9.09
	SINQ	**22.39**	**9.33**	**8.79**
4-bit	RTN	18.74	8.95	8.92
	HQQ	18.96	8.78	8.62
	SINQ	**17.14**	**8.76**	**7.74**

	Calibrated	1.7B	14B	32B
3-bit	GPTQ	32.21	9.54	9.03
	Had+GPTQ	24.70	9.61	8.51
	A-SINQ	**22.30**	**9.31**	**8.45**
4-bit	GPTQ	19.70	8.81	7.80
	AWQ	16.90	8.78	7.79
	A-SINQ	**16.66**	**8.71**	**7.78**

QA: Flip-Rate & Accuracy (4-bit)

Method	Flip-rate % ↓ 14B	32B
RTN	4.13	5.26
HQQ	4.11	5.33
SINQ	3.46	**3.60**
AWQ[c]	3.20	3.69
A-SINQ[c]	**3.18**	**3.60**

Method	Accuracy % ↑ 14B	32B
BF16	*73.3*	*75.6*
HQQ	**72.8**	74.6
SINQ	72.5	**75.2**
AWQ[c]	72.6	74.9
A-SINQ[c]	72.7	75.0

[c] uses calibration. Flip-rate is a more reliable metric than accuracy.

Takeaways

- Weight structure alone yields **activation-awareness**: no calibration data.
- One extra scale + a few Sinkhorn iterations **halve the perplexity gap** to BF16.
- Composes with calibration (AWQ) and non-uniform levels (NF4).
- **Fast to quantize, fast to run, trivial to apply** to any architecture.
- Validated on Qwen3, Llama, Phi, DeepSeek-V3 and MoE models.

Code: `github.com/huawei-csl/SINQ`