QuIP# Even Better LLM Quantization with Hadamard Incoherence and Lattice Codebooks

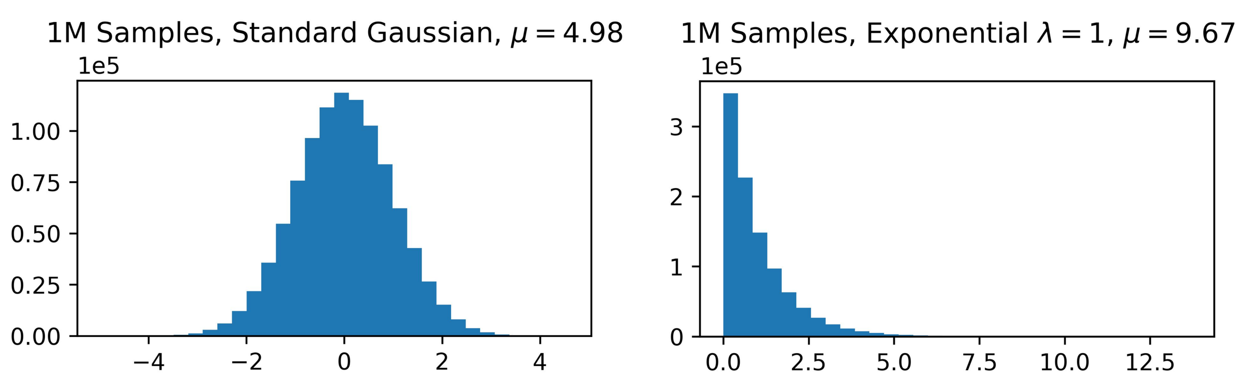
Albert Tseng* albert@cs.cornell.edu
Jerry Chee* jerrychee@cs.cornell.edu
Qingyao Sun qs234@cornell.edu
Volodymyr Kuleshov kuleshov@cornell.edu
Christopher De Sa cdesa@cs.cornell.edu

QuIP# achieves **state-of-the-art 2 bit weight-only LLM quantization** through **better incoherence processing** with the random Hadamard transform, E8 lattice-based **fast vector quantization**, and **fine-tuning**. QuIP# is **over 3X faster than FP16**.

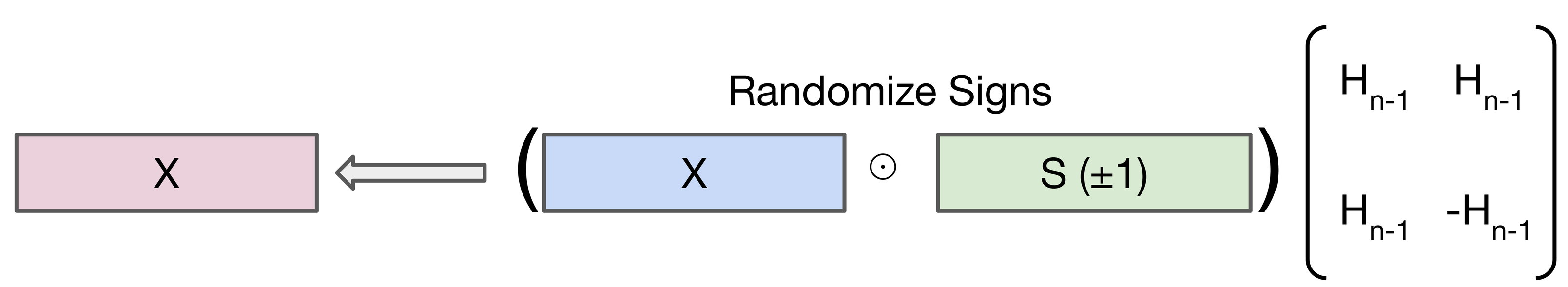
Incoherence Processing with the Random Hadamard Transform

LLM weights have "outliers," making them hard to quantize. We can solve this by multiplying each weight matrix $W \in \mathbb{R}^{m \times n}$ with random orthogonal matrices. This concentrates their entries, making them **incoherent** (small μ).

$$\mu = \max_{ij} \frac{|W_{ij}|}{\|W\|_F}\sqrt{mn}$$



QuIP# introduces the **random Hadamard transform (RHT)** for incoherence processing. The RHT performs $X \leftarrow VSX$, where V is a Hadamard matrix and S is a random sign vector. Hadamard matrices are recursively defined, so the RHT can be performed in **O(n log n)** time with **minimal storage overhead**.



The RHT gives a **log dependence** for μ on the matrix size, improving QuIP[1]'s $\log^2$ dependence and $O(n\sqrt{n})$ runtime.

$$\mu_W = 2\log\left(\frac{4mn}{\delta}\right) \qquad \mu_H = \sqrt{2\log\left(\frac{2n^2}{\delta}\right)}$$

Vector-Quantizing Incoherent Matrices with BlockLDLQ

Like existing works, we quantize linear layers independently by minimizing a "proxy error" that represents the expected output activation error.

$$\mathbb{E}_{x\sim D}\, \mathrm{tr}\left((W-\hat{W})xx^T(W-\hat{W})^T\right) \qquad H = \mathbb{E}_{x\sim D}\left[xx^T\right]$$

To minimize this, QuIP# introduces the BlockLDLQ algorithm, a direct extension of LDLQ to vector quantization. BlockLDLQ iteratively rounds g columns together with linear feedback A from already-rounded columns.

$$\hat{W}_k = Q\left(W_k + (W_{:k-1} - \hat{W}_{:k-1})A_k\right)$$

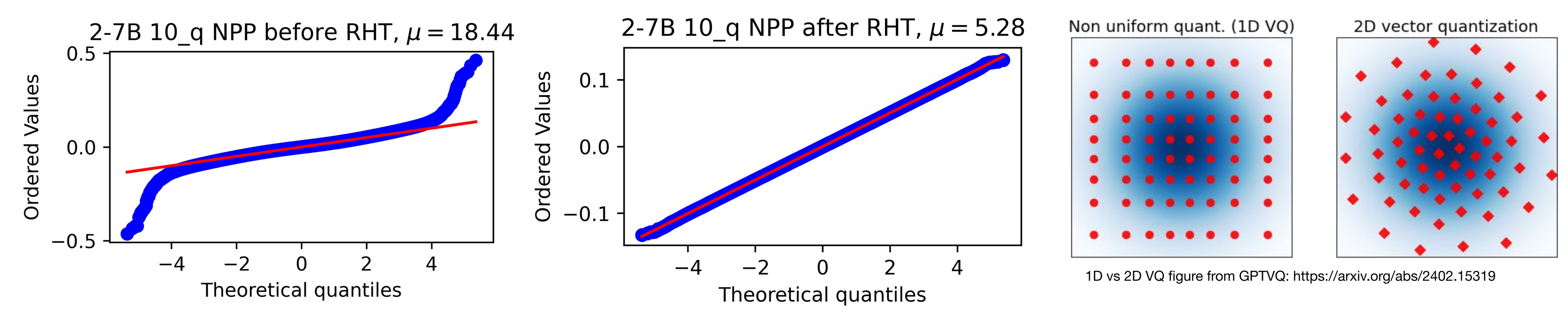
If we set A to the g-block LDL decomposition of H, we can bound the error of BlockLDLQ. Note that this bound depends on the incoherence μ – reducing μ (such as with the RHT) directly improves the quantization error.

$$\mathbb{E}_{x\sim D}\, \mathrm{tr}\left((W-\hat{W})H(W-\hat{W})^T\right) \le \frac{gm\mu^2\sigma^2}{n}\mathrm{tr}(H^{1/2})^2$$

1. QuIP: 2-Bit Quantization of Large Language Models With Guarantees. J. Chee, Y. Cai, V. Kuleshov, C. De Sa. NeurIPS 2023.
2. Extreme Compression of Large Language Models via Additive Quantization. V. Egiazarian, A. Panferov, D. Kuznedelev, E. Frantar, A. Babenko, D. Alistarh. ICML 2024.
3. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. E. Frantar, S. Ashkboos, T. Hoefler, D. Alistarh. ICLR 2023.

Fast Vector Quantization with E8-Based Codebooks

The RHT makes W's entries approximately i.i.d Gaussian. QuIP# uses this shaping by **vector quanting weights.** Vector Quantization (VQ) quantizes d numbers together to a codebook. This codebook can be shaped to the source distribution, reducing distortion.

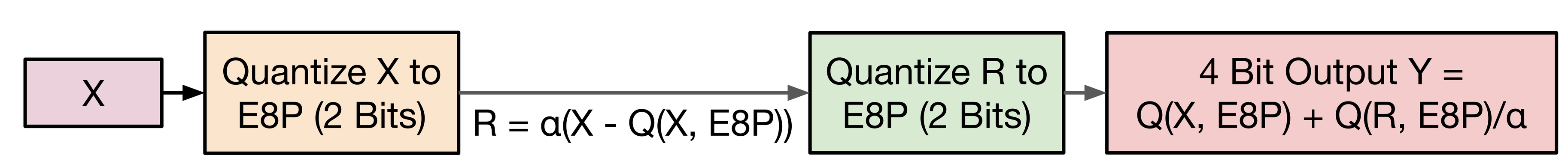


Increasing the VQ dimension reduces error. However, direct k-bit d-dim VQ needs $O(2^{kd}d)$ space and time, making **high-dimensional VQ intractable.** Furthermore, for fast inference, the codebook must fit in GPU L1 cache (<100KB), limiting d (<4).

Codebook	FP16	**E8P (8D, QuIP#)**	4D VQ (D4 Lattice)	2 Bit Int (1D, QuIP)
Wiki2 PPL (no FT) ↓	3.12	**4.16**	4.41	5.90

QuIP# solves this with a novel 2 bit 8D codebook, **E8P,** based on the **highly symmetric E8 lattice**. E8 achieves the **optimal 8D unit-ball packing**, and E8's symmetry makes E8P **1000X smaller** than a naive 8D codebook (1KiB vs 1MiB). E8P can also be decoded in **<4 instructions per weight**, making QuIP# fast.

To hit higher bitrates, QuIP# uses **residual vector quantization (RVQ).** RVQ repeatedly quantizes the quantization residual, **exponentially decreasing BlockLDLQ's error**. Since E8P is uniform, the residuals are also ball-shaped, letting us recursively use E8P in RVQ.



Why Post Training Quantization (PTQ)?

LLMs have billions of parameters that can take up over a terabyte of memory. Unfortunately, **small batch autoregressive inference is memory bound**, meaning that we can **only generate tokens as fast as we can read in weights.**

PTQ can accelerate inference by quantizing weights to smaller data types and thus compressing models. This directly **reduces memory footprint** and **increases theoretical inference speed**. Quantized LLMs also require less hardware to run and are more information-efficient than native LLMs.

Fine-Tuning (FT)

QuIP# uses a novel fine-tuning algorithm to efficiently capture **inter-layer interactions**. First, QuIP# **tunes layers before quantization** to correct for quantization error from prior layers. Then, it tunes unquantized parameters **end-to-end.** QuIP# needs **<1000 samples and 100 GPU hours** for a 70B model and can **halve quantization error.**

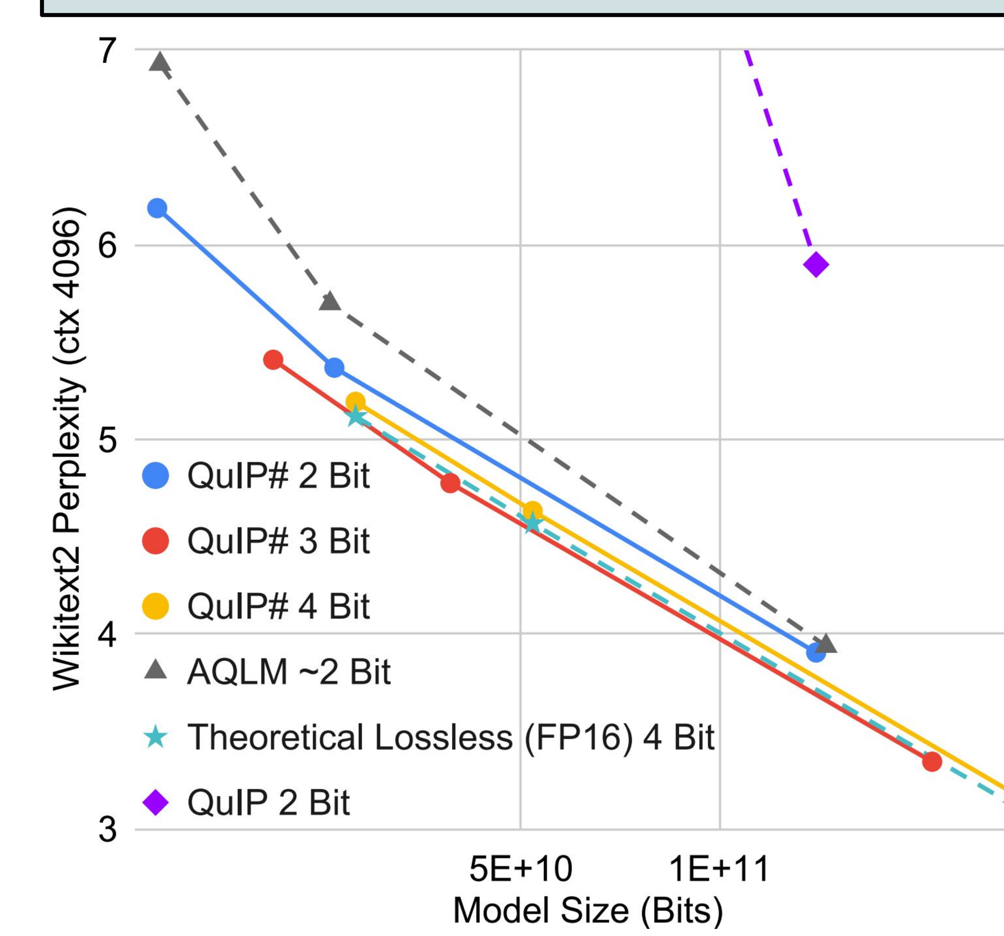
PPL ↓	2-7B	
Method	Wiki2	C4
FP16	5.12	6.63
2 Bit no FT	8.22	11.0
2 Bit FT	**6.19**	**8.16**

QuIP# is SOTA in Quality and Inference Speed

QuIP# is the first PTQ method where **3 bits scales better than 4 bits.** QuIP# is over **3X faster than FP16** inference and **5X faster than AQLM[2]**'s direct-VQ approach.

Prompt: How much wood could a woodchuck chuck if a woodchuck could chuck wood?

2 Bit QuIP# Llama 2 70B Chat: The answer to this tongue-twister is a bit of a mystery. Woodchucks, also known as groundhogs, are rodents that burrow in the ground and are known for their ability to move large amounts of soil. However, there is no scientific evidence to suggest …



PPL ↓		2-7B		2-13B		2-70B	
Mthd.	Bits	W2	C4	W2	C4	W2	C4
FP16	16	5.12	6.63	4.57	6.05	3.12	4.97
QuIP	4	-	-	4.76	6.29	3.58	5.38
QuIP#	**4**	**5.19**	**6.75**	**4.63**	**6.13**	**3.18**	**5.02**
GPTQ[3]	3	8.06	10.6	5.85	7.86	4.40	6.26
QuIP	3	-	-	5.12	6.79	3.87	5.67
AQLM	3	5.46	7.10	4.83	6.37	3.36	5.17
QuIP#	**3**	**5.41**	**7.14**	**4.78**	**6.35**	**3.35**	**5.15**
QuIP	2	-	-	13.5	16.2	5.90	8.71
AQLM	2	6.93	8.84	5.70	7.59	3.94	5.72
QuIP#	**2**	**6.19**	**8.16**	**5.35**	**7.20**	**3.91**	**5.71**

0-shot Acc. ↑		2-7B			2-13B			2-70B		
Mthd.	Bits	ArcC	PiQA	Wino	ArcC	PiQA	Wino	ArcC	PiQA	Wino
FP16	16	40.0	78.5	67.3	45.6	73.5	69.6	51.1	81.1	77.0
QuIP	4	-	-	-	44.9	79.0	69.7	47.0	80.3	76.0
QuIP#	**4**	**40.5**	**78.4**	**67.6**	**45.5**	**78.9**	**69.9**	**50.6**	**81.4**	**77.1**
QuIP	3	-	-	-	41.5	76.9	69.6	46.3	80.0	74.6
QuIP#	**3**	**39.2**	**77.3**	**66.5**	**44.0**	**78.4**	**69.1**	**50.9**	**81.4**	**76.4**
QuIP	2	19.4	54.6	51.8	23.5	62.0	52.8	34.0	74.8	67.5
QuIP#	**2**	**34.6**	**75.1**	**64.9**	**39.5**	**77.3**	**67.7**	**48.7**	**80.3**	**75.9**

bs=1 Speed tok/s (HF, 4090)		
Method	2-7B	2-70B
FP16	33.1	OOM
AQLM 2 Bit	20.6	8.27
QuIP# 2 Bit	**106.3**	**25.9**

QuIP# bs=1 Speed %MemBW (FA, 4090)		
Model Size	2 Bits	4 Bits
7B	29.6%	40.9%
70B	**56.8%**	OOM

HF = HuggingFace and FA = FlashAttention inference engines.