any4: Learned 4-bit Numeric Representation for Large Language Models

Mostafa Elhoushi
(m.elhoushi@ieee.org)
Jeff Johnson
(jhj@meta.com)

Fundamental AI Research (FAIR) at Meta

ICML International Conference On Machine Learning 2025

Quantization / weight distribution mismatch

Datacenter inference: DRAM/HBM increasingly dominate power consumption in many cases, better to be arithmetic than memory bound. Reduce data size and make the bits you are moving around have more meaning?
Edge devices: Smaller models allow use of smaller and/or slower memories (lower power) while still maintaining performance targets.

4-bit quantization yields significant speedups while maintaining sufficient accuracy (unlike most 1-3 bit implementations). Training using high dynamic range floating-point arithmetic, varying scale factors, and outliers offer challenges to lossy compression via quantization.

Efficient inference often uses uniform quantization (int4) or hardware-supported non-uniform quantization (fp4) which **do not match weight distribution (quasi-Gaussian)**. Quantization grouping (shared scale/offset) helps but is just an affine transformation. Distribution mismatch increases quantization error and thus lowers accuracy.

Make dequantization values match weight distribution?
Can you learn dequantization values for specific weight matrices?
Can you implement learned dequantization efficiently in hardware?

any4: learned 4-bit quantization from weights

fp4, NormalFloat4 (NF4) [1], AbnormalFloat4 (AF4) [2] are non-uniform distributions (latter two assume Gaussian) but are fixed a priori, not learned from data (fp4 directly in hardware itself).

Weight pre-processing (e.g., AWQ [3], GPTQ/OPTQ [4], QuIP [5], etc.) can smooth outliers and yield improvements with int4, but losses still exist with int4 uniform distribution.

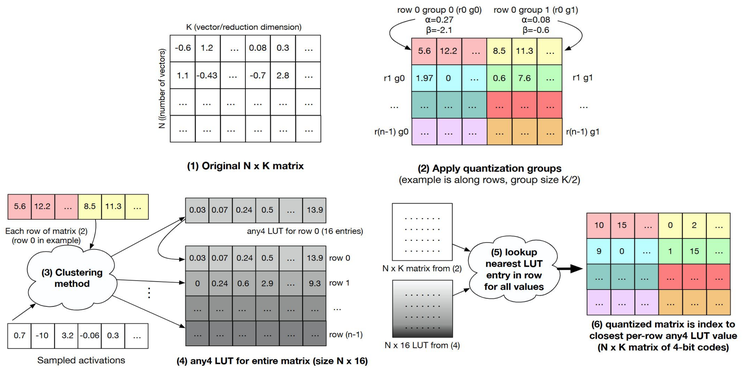
Solution (any4): use per-row data clustering to learn 4-bit (de)quantization values per each row (or column) of a matrix (scalar k-means). Also takes into account expected activation data distribution (sampled activations used in clustering). Any arbitrary int4 code to float mapping can be provided.

Per-row LUT used for dequantization within efficient GPU kernel implementation (**tinygemm**). Orthogonal technique to existing weight pre-processing methods (GPTQ, Hadamard, etc.), just another technique to map floating-point weights to/from int4 codes.

Low overhead (32 bytes/row for bfloat16/float16 LUT containing 16 per-row reproduction values):
N x 4096 matrix: overhead of **0.015625 bits/weight (~1.5%)**
N x 8192 matrix: overhead **0.0078125 bits/weight (~0.78%)**

github.com/facebookresearch/any4

Learning any4 quantization codes (any4 LUTs)



Quantization is applied after standard quantization grouping along the reduction dimension (adjusting values with shared scale and zero offset to lie within [0, 15], group sizes 32/64/128/256 supported). Uniform int4 quantization simply rounds to nearest integer at this stage, but any4 clusters post-quantization group weights along each row which allows for non-uniform/arbitrary distributions. Sampled activations are used in the optimization process to reduce the expected error post-quantization for actual seen usage.

any4 quantization accuracy

	Perplexity ↓				Tasks ↑					
	WikiText-2	C4	PTB	CodeParrot	HumanEval Pass@1	MBPP Pass@1	MMLU	HellaSwag	GSM8K	BBH
Llama3.2 1B										
FP16	9.76	12.77	16.56	3.49	16.46%	21.4%	36.1%	47.7%	6.60%	31.1%
INT4	11.89	15.74	20.32	4.08	9.76%	11.4%	30.1%	44.7%	3.18%	26.2%
FP4	13.01	17.11	21.89	4.28	8.54%	5.8%	29.3%	43.6%	2.27%	23.3%
NF4	10.99	14.63	18.78	3.82	**13.4%**	13.8%	**33.3%**	45.8%	2.65%	26.8%
ANY4	**10.63**	**13.95**	**17.94**	**3.71**	11.0%	**18.6%**	32.9%	**46.7%**	**3.71%**	**29.0%**
Llama3 8B										
FP16	6.14	8.93	10.59	2.54	29.3%	41.4%	62.0%	60.1%	50.7%	62.8%
INT4	6.87	9.89	11.37	2.83	23.2%	35.4%	59.6%	58.6%	40.6%	58.5%
FP4	7.10	10.22	11.81	2.89	22.0%	36.8%	57.1%	58.5%	35.0%	53.2%
NF4	6.63	9.52	11.14	2.72	**23.2%**	**39.2%**	60.7%	59.1%	41.1%	59.0%
ANY4	**6.51**	**9.40**	**11.07**	**2.68**	21.3%	**39.2%**	**61.0%**	**59.5%**	**41.7%**	**59.2%**
Llama3 70B										
FP16	2.86	6.77	8.16	1.91	17.7%	60.8%	75.4%	66.3%	80.6%	82.4%
INT4	3.63	7.97	8.86	2.21	18.3%	45.0%	73.0%	**66.2%**	73.9%	78.4%
FP4	3.94	7.76	8.99	2.17	**22.0%**	50.8%	71.9%	65.6%	75.3%	77.9%
NF4	3.43	7.67	8.84	2.15	18.9%	39.6%	73.7%	66.1%	75.9%	79.3%
ANY4	**3.20**	**7.01**	**8.33**	**1.99**	17.1%	**57.4%**	**75.1%**	66.1%	**78.5%**	**81.8%**

As any4 is adapted to weights and sampled activations, it generally obtains lower perplexity / higher accuracy than other 4-bit codes.

	Quantization Algorithm	Numeric Format	WikiText-2 Perplexity ↓		Numeric Format	WikiText-2 Perplexity ↓		Numeric Format	WikiText-2 Perplexity ↓
Llama3 8B									
	FP16		6.1						
4-bits	RTN	INT4	6.9	3-bits	INT3	17.1	2-bits	INT2	1.9E3
	GPTQ	INT4	**6.5**		INT3	8.2		INT2	2.1E2
	AWQ	INT4	6.6		INT3	8.2		INT2	1.7E6
	QuIP	INT4	**6.5**		INT3	**7.5**		INT2	**85.1**
	RTN	ANY4	**6.5**		ANY3	8.0		ANY2	1.0E3
Llama3 70B									
	FP16		2.9						
4-bits	RTN	INT4	3.6	3-bits	INT3	11.8	2-bits	INT2	4.6E5
	GPTQ	INT4	3.3		INT3	5.2		INT2	**11.9**
	AWQ	INT4	3.3		INT3	4.8		INT2	1.7E6
	QuIP	INT4	3.4		INT3	4.7		INT2	13.0
	RTN	ANY4	**3.2**		ANY3	**4.6**		ANY2	253.8

any4 by itself is competitive with additional weight pre-processing techniques prior to quantization (GPTQ, AWQ, QuIP), but any4 is an orthogonal technique and could be applied in conjunction with these and other pre-processing techniques (future work).

tinygemm: a latency-optimized GPU GEMM library implementing any4

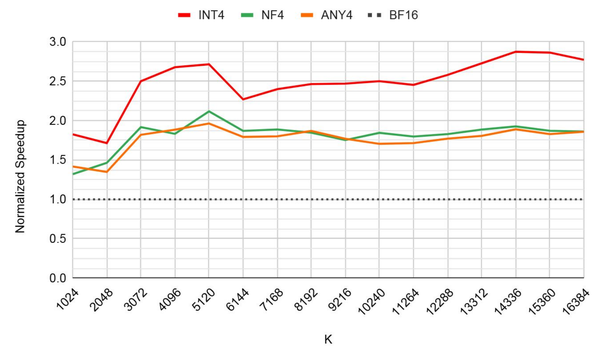
GEMM libraries mainly focus on throughput for large matrix multiplications which can better achieve near peak arithmetic throughput. Small batch (e.g., bs=1) inference in LLMs frequently involves (1 x K) x (K x N) GEMVs with low arithmetic intensity. Memory latency, not arithmetic throughput or memory bandwidth, is often limiting factor.

tinygemm is designed for this small batch / latency sensitive domain. Its int4 kernels are part of pytorch core (since late 2023) and used for small batch int4 inference by compiler/torchao as needed.

Much GEMM machinery added recently (e.g., wgmma) does not work well in this case as by definition we would only be using 1/N - ε/N of the tensor core (TC) throughput for larger N. TC is still more efficient than scalar FMA + reductions (pre-TC GEMMs). Using *smaller* TC tile sizes (e.g., Ampere+ m16n8k16 for fp16/bf16) *increases* utilization.

any4 is an extension to int4 tinygemm that uses small LUTs held in registers or shared memory for dequantization. Our paper discusses "row-wise" any4, but "matrix-wise" any4 is also available which can implement NF4/AF4 or emulate any other 4-bit scalar representation (e.g., fp4) on platforms not supporting it.

tinygemm does not use async loads or shared memory transposition, matrices are directly laid out in "tensor core" format in memory, packed so that all memory accesses are warp contiguous 16B loads straight from gmem to tensor core registers for latency (the reason why pytorch added "tensor core" memory layout). Small batch = no weight reuse.



Performance relative to bf16 baseline for (1 x K) x (K x K) GEMV

References

[1] Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), Advances in Neural Information Processing Systems, volume 36, pp. 10088–10115 (2023).
[2] Yoshida, Davis. NF4 Isn't Information Theoretically Optimal (and that's Good), arXiv 2306.06965 (2023).
[3] Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., and Han, S. Awq: Activation-aware weight quantization for llm compression and acceleration. MLSys (2024).
[4] Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. OPTQ: Accurate quantization for generative pre-trained transformers.Eleventh International Conference on Learning Representations (2023).
[5] Chee, J., Cai, Y., Kuleshov, V., and Sa, C. D. QuIP: 2- bit quantization of large language models with guarantees. Thirty-seventh Conference on Neural Information Processing Systems (2023).

Take a photo to learn more: