GPTVQ: The Blessing of Dimensionality for LLM Quantization

Anonymous Authors[1]

Abstract

Large language models (LLMs) necessitate huge DRAM footprint and memory bandwidth costs, severely limiting deployment on mobile devices. This work demonstrates that non-uniform quantization in one or more dimensions can significantly ease this memory bottleneck. We provide analysis and experimental results to show that the model size versus accuracy trade-off of neural network quantization markedly improves when increasing the quantization dimensionality. To exploit this, we propose GPTVQ: an efficient method that extends GPTQ to non-uniform and vector quantization (VQ). GPTVQ establishes state-of-the-art results in model size vs accuracy across a wide range of LLMs, including Llama-v2/v3 and Mistral. Furthermore, our method is fast: on a single H100 it takes between 3 and 11 hours to process Llamav2-70B. Finally, we show that VQ is practical, by demonstrating simultaneous reduction in DRAM footprint and latency on a VQ quantized LLM on a mobile class Arm® CPU, and a desktop Nvidia® GPU. Our source code is available in the supplementary material.

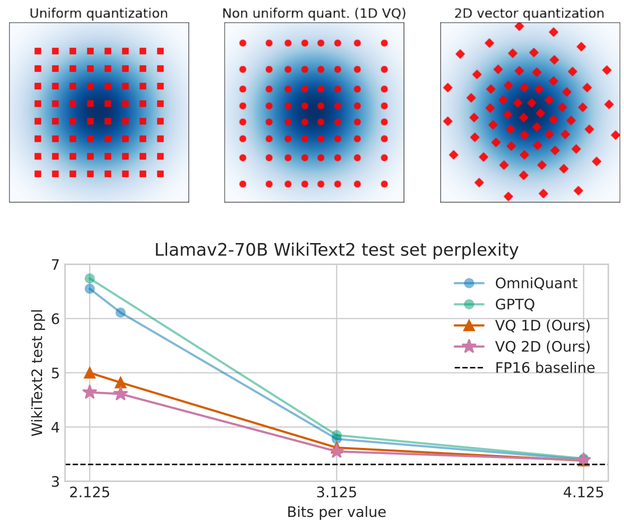

Figure 1. **Top:** Vector quantization more closely fits 2D normal data, compared to uniform and non-uniform grids. **Bottom:** GPTVQ compared to SOTA uniform quantization (Llamav2-70B).

1. Introduction

Large language models (LLMs) have made significant strides in enabling human-like natural language text generation with numerous applications, from general AI assistants like Open AI's GPT (Achiam et al., 2023), to more specialized tasks like coding companions (Roziere et al., 2023) and medical aides (Tu et al., 2024).

However, the impressive capabilities of LLMs require very large model sizes, which makes them challenging to deploy on mobile devices for two reasons. Firstly, the sheer size of LLMs occupy significant valuable DRAM footprint, which is hard to accomodate within the typical 8GB total capacity. Secondly, the bottleneck in LLM inference performance lies in weight movement, since their autoregressive nature requires the loading of every weight for each generated token. Reducing the stored model size directly relaxes both of these challenges.

While low-bit quantization has proven successful in reducing LLM weights down to 4 bits without substantial accuracy loss (Frantar et al., 2022; Lin et al., 2023; Shao et al., 2023), there are strong incentives to push LLM quantization much further. Moving beyond the uniform quantization methods employed in much of the prior research, we investigate the potential to achieve even greater compression by employing non-uniform quantization and subsequently expanding the dimensionality of the representational grid through vector quantization (VQ). In vector quantization Figure 1 shows how multiple weights are quantized together in VQ, achieving a more flexible quantization grid to align closely to the weight distribution.

We integrate our findings into a novel algorithm for post-training quantization called GPTVQ. This method allows fast non-uniform and vector quantization, improving the performance-size trade-off significantly compared to prior state-of-the-art.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

The contributions of this work are as follows:

- Our analysis and experimental results show that increasing the dimensionality of quantization improves the accuracy versus model size trade-offs for many LLMs.
- We propose a fast and accurate algorithm for post-training VQ compression, which achieves SOTA size vs accuracy trade-offs on a wide range of LLMs, while having a practical run time of only 3 to 11 hours on a 70B parameter model.
- We implement and benchmark VQ decompression on a mobile Arm® CPU and an Nvidia® GPU. While VQ leads to significant memory footprint reductions, our on-device timings also demonstrate that it leads to improved latency compared to a 4-bit integer baseline.

2. GPTVQ

Previous VQ methods, like (Stock et al., 2019), require end-to-end fine-tuning and hence do not scale to LLM-sized models. In this section, we introduce GPTVQ, a novel method for efficient and accurate vector-quantization of LLMs. We build on GPTQ (Frantar et al., 2022), a recent uniform quantization method which interleaves column-wise quantization with updates to the remaining (unquantized) weights, using information from the Hessian of the layer output reconstruction MSE. GPTQ provides excellent performance on uniform quantization of LLMs with up to hundreds of billions of parameters. Appendices H and I present further extensions to GPTVQ, including Codebook SVD and Blockwise Data Normalization.

2.1. Background: GPTQ

A large body of literature exists with methods to alleviate the effects of quantization noise on model accuracy, see (Gholami et al., 2022; Nagel et al., 2021) for recent surveys. A popular and effective approach in post-training quantization (PTQ), introduced by AdaRound (Nagel et al., 2020), is to modify weights to minimize a layer's output error as an approximation to the full network's loss:

$$\mathbb{E}\left[\mathcal{L}(\theta+\epsilon)-\mathcal{L}(\theta)\right] \approx \sum_{\ell} ||\mathbf{W}^{\ell}\mathbf{X}^{\ell} - \widehat{\mathbf{W}}^{\ell}\mathbf{X}^{\ell}||_F^2, \quad (1)$$

where $\mathbf{W}^{\ell}$ is the weight for layer ℓ, $\widehat{\mathbf{W}}^{\ell} = \mathbf{W}^{\ell} + \epsilon^{\ell}$ is the (quantized) approximation to this weight tensor, and $\mathbf{X}^{\ell}$ of shape $R \times N$ denotes the input data for layer ℓ from a calibration dataset, with N individual data points of dimensionality R along its columns.

GPTQ follows Optimal Brain Quantization (OBQ; (Frantar and Alistarh, 2022)), which uses the Hessian of Equation 1. This Hessian can be efficiently computed as $\mathbf{H}^{(\ell)} = \mathbf{X}^{(\ell)}\mathbf{X}^{(\ell)T}$. Like OBQ, GPTQ aims to minimize the Hessian-weighted error introduced by quantizing weights in $\mathbf{W}^{(\ell)}$:

$$E = \sum_{q} |E_q|_2^2 \qquad E_q = \frac{(\mathbf{W}_{:,q} - \text{quant}(\mathbf{W}_{:,q}))^2}{\left[\mathbf{H}^{-1}\right]_{qq}} \quad (2)$$

.

GPTQ extends OBQ in the following ways. First, GPTQ exploits the fact that $\mathbf{H}^{(\ell)}$ is shared over all rows of $\mathbf{W}^{(\ell)}$ by quantizing all weights in a column in parallel, from left to right. This obviates the need for independent Hessian updates for different rows. After quantizing a column q, all remaining (unquantized) columns $q' > q$ are modified with a Hessian-based update rule δ that absorbs the error introduced by quantizing column q on the layer's output:

$$\delta = -\frac{\mathbf{W}_{:,q} - \text{quant}(\mathbf{W}_{:,q})}{\left[\mathbf{H}^{-1}\right]_{qq}} \mathbf{H}_{:,(q+1):} \quad (3)$$

For further details on GPTQ we refer the reader to (Frantar et al., 2022).

2.2. The GPTVQ method

The GPTVQ method generalizes the GPTQ method for non-uniform and vector quantization.

Following the GPTQ framework we perform quantization of the weight tensor in a greedy manner starting from the first column. The details of the method are given in Algorithm 1. Given the VQ dimensionality d, we quantize d columns at a time. In the case of scalar quantization, the optimal Hessian-weighted quantization of a single columnn was achieved by rounding to nearest. However, in the case of vector quantization, simply choosing the nearest centroid might be suboptimal as error in each of d coordinates is weighted differently. If we denote the inverse of the diagonal part of the inverse Hessian as $\mathbf{D} = \text{diag}\left(1/[\mathbf{H}^{-1}]_{11}, \ldots, 1/[\mathbf{H}^{-1}]_{cc}\right)$, the following rule is used for choosing the optimal assignment j for a data point $\mathbf{x}^{(i)}$ and the corresponding subset of $\mathbf{D}^{(i)}$:

$$j = \arg\min_{m} \left(\mathbf{x}^{(i)} - \mathbf{c}^{(m)}\right)^T \mathbf{D}^{(i)} \left(\mathbf{x}^{(i)} - \mathbf{c}^{(m)}\right). \quad (4)$$

After quantizing d columns, we update the remaining weights using the update rule 3. We accumulate the update along d coordinates and apply it to the remaining weights as a single operation. To further minimize quantization error, we use several codebooks per layer, each assigned to a *group* of weights (see Algorithm 1). We use group sizes of at most 256 columns, to ensure codebook initialization can capture the updates of Eq. 3. E.g., a group of 2,048 weights is 8 rows by 256 columns.

Codebook initialization To initialize the codebook for a group of weights, we propose the following variant of the EM algorithm. Given the set of d-dimensional vectors $\mathbf{x}^{(i)}$, our goal is to find k centroid vectors $\mathbf{c}^{(m)}$ and the corresponding sets of assignments I_m, i.e. the list of indices of vectors assigned to the centroid m. The objective is the following sum of weighted distance functions:

$$\min_{\mathbf{I},\mathbf{c}^{(0),\dots,(k)}} \sum_{m=0}^{k} \sum_{i \in I_m} \left(\mathbf{x}^{(i)} - \mathbf{c}^{(m)}\right)^T \mathbf{D}^{(i)} \left(\mathbf{x}^{(i)} - \mathbf{c}^{(m)}\right), \tag{5}$$

where $\mathbf{D}^{(i)}$ is a $d \times d$ subset of $\mathbf{D}$ corresponding to the data point $\mathbf{x}^i$. E.g. for 2D vector quantization, these matrices are share among pairs of columns. For the case of $\mathbf{D}^{(i)}$ equal to identity, the clustering method is equivalent to K-means. The objective can be minimized using E- and M-steps as follows.

E-step: find the assignment j for each unquantized d-dimensionl vector $\mathbf{x}^{(i)}$ that minimizes the objective (4). Using this distance function assigns optimal centroids based on the data-aware loss.

M-step: find the centroid value $\mathbf{c}^{(m)}$ that minimizes

$$\mathbf{c}^{(m)} = \arg\min_{\mathbf{c}^{(m)}} \sum_{i \in I_m} \left(\mathbf{x}^{(i)} - \mathbf{c}^{(m)}\right) \mathbf{D}^{(i)} \left(\mathbf{x}^{(i)} - \mathbf{c}^{(m)}\right). \tag{6}$$

This objective is a quadratic form w.r.t $\mathbf{c}^{(m)}$. The optimal value is computed in a closed form as $\mathbf{c}^{(m)} = \left(\sum_{i \in I_m} \mathbf{D}^{(i)}\right)^+ \left(\sum_{i \in I_m} \mathbf{D}^{(i)} \mathbf{x}^{(i)}\right)$, where $(\cdot)^+$ is a Moore–Penrose pseudoinverse. During the vector quantization operation on line 4 in Algorithm 2, we use the assignment step defined in Equation 4 as well. Practically, we find no performance difference between using the inverse Hessian diagonal, or the full d-dim inverse sub-Hessian.

Codebook update After the procedure in Algorithm 1 is complete, we found that the output reconstruction error can be further reduced through a *codebook update*. Recall that, in line 4 of Algorithm 2, $\mathbf{Q}$ is incrementally constructed from the elements of $\mathbf{C}$. Since this construction constitutes a lookup of values in $\mathbf{C}$, the layer-wise objective can still be minimized w.r.t $\mathbf{C}$. The objective is a quadratic program and is convex:

$$\min_{\mathbf{C}_0,\dots,\mathbf{C}_N} ||\mathbf{WX} - \mathbf{QX}||_F^2, \tag{7}$$

where $\mathbf{Q}(\mathbf{C}_0, \dots, \mathbf{C}_N)$ is a look-up operation, reconstructing the quantized weights from the centroids. The gradient of $\mathbf{Q}$ w.r.t. $\mathbf{C}$ can be defined simply, as constructing Q only involves a look-up operation. In each GD step, the values in $\mathbf{C}$ are updated, and $\mathbf{Q}$ is reconstructed using the new values in $\mathbf{C}$, keeping the assignments fixed.

Total bits per value As a measure of total model size, we compute *bits per value* (bpv), given by $\log_2(k)/d + kdb_c/l$, where k is the number of centroids, d is the VQ dimensionality, b_c is the codebook bit-width, and l is the group size, i.e., the number of weights sharing a codebook. We choose values for k s.t. $\log_2(k)$ is an integer.

3. Experiments and results

In this section we evaluate GPTVQ and compare the performance of vector quantization in 1, 2 and 4 dimensions against uniform quantization baseline methods. We follow the experimental setup of (Shao et al., 2023) in terms of calibration dataset evaluation. Further details on experimental setup, datasets, and baselines can be found in Appendix A. Ablations on various model choices can be found in Appendix G.

Codebook overhead For a given bits per index b and VQ dimensionality d, we set group size l to reach an overhead of 0.125 bits per value for all values of b, and additionally consider an overhead 0.25 bits per value for $b = 2$. These are chosen to match the overhead incurred by a 16-bit quantization scale for the commonly used group size of 128 (e.g., (Frantar et al., 2022)) and the group size of 64 used by (Shao et al., 2023).

Main results Table 1 summarizes results for GPTVQ, where we report WikiText 2 perplexity and an average over zero-shot task scores for the PIQA, BoolQ, ARC-easy, ARC-challenge, HellaSwag and WinoGrande tasks. We include all Llama-v2 models, Mistral-7B-v0.1 and Mixtral-8x7B-v0.1. More results can be found in Appendix E: Table 7 and Table 8 contain individual scores for the zero-shot tasks, Table 5 contains WikiText2 perplexity for all Llama-v1 models, and Table 6 shows perplexity on 4 bit quantization. A separate comparison to AQLM can be found in Appendix B.1. Full VQ configurations can be found in Table 4.

Table 1 shows that non-uniform quantization using GPTVQ generally yields improved results over uniform PTQ methods. This gap becomes especially large at low bitwidths and for very large models. For example, comparing GPTVQ 2D on Llamav2-70B to OmniQuant W2@g128, we see an improvement of nearly 2 perplexity points. Furthermore, in nearly all cases, 2D VQ outperforms 1D VQ, while 4D VQ shows even more significant improvements.

3.1. On-device VQ inference evaluation and comparison

To investigate the effect of VQ quantized models on model DRAM footprint and latency, we implemented optimized kernels for both Arm® mobile CPU and Nvidia® GeForce RTX 3080 GPU.

Table 1. **Weight-only quantization results of Llama-v2/v3, Mistral, and Mixtral-MoE Models**. We report WikiText2 perplexity and average zero-shot accuracy; Models marked L2 denote Llama-v2, L3 denote Llama-v3, M denotes Mistral, and 8x7B denotes Mixtral-MoE 8x7B. Numbers marked in bold are SOTA or surpass it, numbers underlined are on par with or outperform at least one VQ variant. * Following (Huang et al., 2024), Llama3-8B zeroshot average omits BoolQ.

		WikiText2 perplexity ↓							Zeroshot avg acc. ↑				
		L2-7B	L2-13B	L2-70B	L3-8B	L3-70B	M-7B	8x7B	L2-7B	L2-13B	L3-8B*	M-7B	8x7B
FP16		5.47	4.88	3.31	6.1	2.9	5.25	3.84	70.5	73.2	68.6	75.7	75.9
2.125 W2 g128	RTN	4e3	122	27.3	2e3	5e5	1e3	4e3	36.9	42.1	36.0	37.8	38.3
	GPTQ	36.8	28.1	6.74	2e2	11.9	15.7	14.1	41.4	46.6	36.2	41.9	44.5
	AWQ	2e5	1e5	-	2e6	2e6	-	-	-	-	-	-	-
	OQ	11.1	8.26	6.55	-	-	-	-	-	-	-	-	-
	Ours 1D	12.2	**7.40**	**5.03**	**15.9**	**9.37**	**14.0**	**8.37**	**47.8**	**61.8**	**41.1**	**42.8**	**54.9**
	Ours 2D	**7.77**	**6.52**	**4.72**	**11.3**	**7.37**	**7.53**	**5.92**	**58.6**	**64.5**	**53.9**	**64.5**	**64.4**
	Ours 4D	**7.18**	**6.07**	**4.44**	**9.94**	**6.59**	**6.89**	**5.28**	**60.5**	**65.7**	**57.3**	**65.7**	**68.7**
2.25 W2 g64	RTN	432	26.2	10.3	-	-	71.5	156	42.4	46.4	-	44.8	46.9
	GPTQ	20.9	22.4	NAN	-	-	14.2	10.1	47.5	54.2	-	51.8	48.8
	AWQ	2e5	1e5	-	-	-	-	-	-	-	-	-	-
	OQ	9.62	7.56	6.11	-	-	-	-	-	-	-	-	-
	Ours 1D	10.1	**6.99**	**4.85**	**14.1**	**8.31**	**9.69**	**7.75**	**52.8**	**63.3**	**57.3**	**56.3**	**57.4**
	Ours 2D	**7.61**	**6.41**	**4.58**	**10.8**	**6.83**	**7.24**	**5.58**	**61.5**	**64.8**	**60.3**	**65.3**	**65.7**
	Ours 4D	**6.99**	**5.98**	**4.36**	**9.59**	**6.21**	**6.66**	**5.16**	**62.9**	**67.5**	**62.3**	**68.2**	**69.3**
3.125 W3 g128	RTN	6.66	5.51	3.97	27.9	11.8	6.15	5.18	67.3	70.8	40.2	71.8	72.4
	GPTQ	6.29	5.42	3.85	8.2	5.2	5.83	4.71	66.2	71.4	61.7	72.2	72.7
	AWQ	6.24	5.32	-	8.2	4.8	-	-	-	-	-	-	-
	OQ	6.03	5.28	3.78	-	-	-	-	-	-	-	-	-
	Ours 1D	**5.95**	**5.19**	**3.64**	**7.29**	**4.29**	**5.79**	**4.59**	**66.9**	71.4	**65.7**	71.0	**73.5**
	Ours 2D	**5.83**	**5.12**	**3.58**	**7.00**	**4.04**	**5.51**	**4.27**	**68.3**	71.2	**66.1**	**73.9**	**75.1**

The Arm® CPU kernel employs the table lookup (`TBL`) instruction to translate an index of (at most) 5 bits to an 8 bit integer, with two `TBL` instructions chained for 2D VQ. On GPU, we use native CUDA vector types to load and unload data quickly from GPU memory into the registers and back, such as `char4`/`uchar4`, and custom agglomerations of those, up to `char128`. The code for these kernels will be made available in the future.

We measure the time to transfer and unpack/decode the weights of a Llamav2-7B `gate_proj` layer (11008×4096), for VQ and to uniformly quantized data, and also FP16 on GPU. Furthermore, we integrate our Arm® kernel with a matmul operation for an end-to-end token generation experiment on Llamav2-7B quantized using 1D VQ.

Table 2 shows that for both data transfer and token generation, VQ can achieve significant footprint reductions, with strictly positive latency impact on Arm® CPU, and negligible to positive latency impact on Nvidia® GPU.

Table 2. **Measured VQ data transfer/decoding, and LLM token generation on mobile device.** Exp: experiment, Data Transfer (T) or Token Generation (G). Ptfm: platform, Arm® CPU or NVIDIA® GPU. Format: either Uniform or VQ. Rel. FP: relative footprint. Rel. lat: relative latency.

Exp	Ptfm	bpv	Format	d	Rel. FP ↓	Rel. lat. ↓
T	CPU	4	Unif	1D	1.00×	1.00×
T	CPU	8	Unif	1D	2.00×	1.93×
T	CPU	3	VQ	2D	0.75×	0.98×
T	CPU	2.75	VQ	2D	0.69×	0.96×
T	CPU	2.25	VQ	2D	0.56×	0.87×
G	CPU	3.125	VQ	1D	0.78×	0.96×
T	GPU	4	Unif	1D	1.00×	1.00×
T	GPU	8	Unif	1D	2.00×	1.47×
T	GPU	16	FP	1D	4.00×	2.72×
T	GPU	2.125	VQ	2D	0.53×	1.03×
T	GPU	2.125	VQ	4D	0.53×	0.71×
T	GPU	3.125	VQ	2D	0.78×	1.06×

4. Conclusions

In this work, we have shown that vector quantization in one or more dimensions progressively improves quantized large language model accuracy. We introduced GPTVQ, a fast method for post-training quantization of LLMs using VQ. GPTVQ achieves SOTA model size vs accuracy trade-offs on a wide range of LLMs and zero-shot tasks. Finally, we have shown that VQ can be efficiently on Arm® CPU and Nvidia® GPU platforms, with negligible to positive impact on token generation speed.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

David Arthur and Sergei Vassilvitskii. K-means++ the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, 2007.

Christopher M Bishop. Pattern recognition and machine learning. *Springer google schola*, 2:1122–1128, 2006.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

Minsik Cho, Keivan A Vahid, Saurabh Adya, and Mohammad Rastegari. Dkm: Differentiable k-means clustering layer for neural network compression. *arXiv preprint arXiv:2108.12659*, 2021.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

Juncan Deng, Shuaiting Li, Chengxuan Wang, Hong Gu, Haibin Shen, and Kejie Huang. LLM-codebook for extreme compression of large language models, 2024. URL `https://openreview.net/forum?id=nMbWsXPUVL`.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. *arXiv preprint arXiv:2401.06118*, 2024.

Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme model compression. *arXiv preprint arXiv:2004.07320*, 2020.

Elias Frantar and Dan Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning. *Advances in Neural Information Processing Systems*, 35:4475–4488, 2022.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL `https://zenodo.org/records/10256836`.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC, 2022.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE, 1993.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Wei Huang, Xudong Ma, Haotong Qin, Xingyu Zheng, Chengtao Lv, Hong Chen, Jie Luo, Xiaojuan Qi, Xianglong Liu, and Michele Magno. How good are low-bit quantized llama3 models? an empirical study. *arXiv preprint arXiv:2404.14047*, 2024.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Sakaguchi Keisuke, Le Bras Ronan, Bhagavatula Chandra, and Choi Yejin. Winogrande: An adversarial winograd schema challenge at scale. 2019.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Julieta Martinez, Jashan Shewakramani, Ting Wei Liu, Ioan Andrei Bârsan, Wenyuan Zeng, and Raquel Urtasun. Permute, quantize, and fine-tune: Efficient compression of neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15699–15708, 2021.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pages 7197–7206. PMLR, 2020.

Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv preprint arXiv:2308.13137*, 2023.

Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33:18098–18109, 2020.

Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. And the bit goes down: Revisiting the quantization of neural networks. *arXiv preprint arXiv:1907.05686*, 2019.

Xiaohu Tang, Yang Wang, Ting Cao, Li Lyna Zhang, Qi Chen, Deng Cai, Yunxin Liu, and Mao Yang. Lut-nn: Empower efficient neural network inference with centroid learning and table lookup. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, pages 1–15, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Tao Tu, Anil Palepu, Mike Schaekermann, Khaled Saab, Jan Freyberg, Ryutaro Tanno, Amy Wang, Brenna Li, Mohamed Amin, Nenad Tomasev, et al. Towards conversational diagnostic ai. *arXiv preprint arXiv:2401.05654*, 2024.

BigScience Workshop, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Table 3. **Perplexity, zeroshot average and decode latency comparison for GPTVQ and AQLM.**

		WikiText2 perplexity ↓			Zeroshot avg acc. ↑		GPU rel. latency ↓
		L2-7B	L2-13B	L2-70B	L2-7B	L2-13B	L2-7B
FP16		5.12	4.57	3.12	62.35	65.38	1×
AQLM	≈2	6.64	5.65	**3.94**	56.47	60.59	0.76×
Ours (4D)	2.125	6.70	5.71	4.20	56.45	**61.35**	0.26×
AQLM	≈2.25	**6.29**	**5.41**	-	**58.57**	61.58	N/A
Ours (4D)	2.25	6.52	5.62	4.12	58.08	**62.25**	N/A

A. Experimental setup

Models We use the Llama-1 (Touvron et al., 2023a), Llama-2 (Touvron et al., 2023b), and Llama-3 as well as Mistral-7B-v0.1 (Jiang et al., 2023) and Mixtral-MoE-8x7B-v0.1 (Jiang et al., 2024). Additionally, we run a single ablation on BLOOM-560M (Workshop et al., 2022).

Datasets Following Shao et al. (2023), we use 128 sequences of 2048 tokens from the WikiText2 (Merity et al., 2016) training set as calibration data for all experiments. We evaluate our models on token perplexity for the WikiText2 validation set for a sequence length 2048, as well as zero-shot language tasks: PIQA (Bisk et al., 2020), ARC-easy/-challenge (Clark et al., 2018), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), and WinoGrande (Keisuke et al., 2019). For Llama3, following (Huang et al., 2024), we omit BoolQ from the zeroshot average to allow fair comparison to the zeroshot results in (Huang et al., 2024). For all evaluation tasks except WikiText2 perplexity we use the LLM-evaluation-harness (Gao et al., 2023).

Baselines We compare GPTVQ to various uniform quantization methods with different group sizes, at the same overall bits-per-value (bpv). We include Round-to-Nearest (RTN) and several recent state-of-the-art PTQ approaches for LLMs: GPTQ (Frantar et al., 2022), AWQ (Lin et al., 2023), and OmniQuant (Shao et al., 2023). We take AWQ and OmniQuant baseline numbers from (Shao et al., 2023), all Llama3 baseline numbers from (Huang et al., 2024), and generate all other baseline numbers ourselves. In Appendix B.1 we provide a detailed comparison to AQLM (Egiazarian et al., 2024), recent work that applies VQ to LLMs in a different manner.

B. Related work

Vector quantization A number of works propose vector quantization of CNN weights (Cho et al., 2021; Fan et al., 2020; Gong et al., 2014; Martinez et al., 2021;?; Stock et al., 2019; Wu et al., 2016). The most common approach is to reshape the weights of convolutional or fully connected layers into a matrix, and then apply K-means clustering directly on the columns. Typically, the clustering is applied on scalars or vectors of dimension 4 or higher. Some of these works consider data-aware optimization of the quantized weights. Most often, a variant of the EM algorithm is used in order to update centroids and assignments (Gong et al., 2014; Stock et al., 2019). An alternative approach is using a differentiable K-means formulation, which enables fine-tuning using SGD with the original loss function in order to recover the network accuracy (Cho et al., 2021; Fan et al., 2020; Tang et al., 2023).

LLM quantization Applying DNN quantization approaches to recent LLMs often poses significant computational challenges. Therefore, even uniform post-training quantization methods must be optimized for scalability (Frantar et al., 2022). Since vector quantization approaches have even higher computational complexity, applying them to LLM weights compression may be expensive. The most similar to our work is a method (Deng et al., 2024), which uses gradient-based layer sensitivities to update the codebooks and a reduced complexity LoRA-based approach (Hu et al., 2021) to partially recover the accuracy.

Hessian-based compression methods Several classical works suggest second-order approximation of the neural network loss function for accurate unstructured pruning (Hassibi et al., 1993; LeCun et al., 1989). A more recent line of work extends this family of methods to PTQ (Frantar and Alistarh, 2022; Frantar et al., 2022; Singh and Alistarh, 2020).

Algorithm 1 GPTVQ algorithm: Quantize a weight tensor $\mathbf{W} \in \mathbb{R}^{r \times c}$ given the inverse Hessian $\mathbf{H^{-1}}$, the block size B, VQ dimensionality d, the number of centroids k, and the group size l

0: $N_b \leftarrow \frac{c}{B}$ {the number of blocks}
0: $m \leftarrow \frac{l}{r}$ {the number of columns in a group}
0: $\mathbf{Q} \leftarrow \mathbf{0}_{r,c}$
0: $\mathbf{E} \leftarrow \mathbf{0}_{r,c}$
0: $N_g \leftarrow \frac{rc}{l}$ {the number of groups/codebooks}
0: $\mathbf{C}_i \leftarrow \mathbf{0}_{d,k}, i = 1, \ldots, N_g$
0: $\mathbf{H}^{-1} \leftarrow$ Cholesky$(\mathbf{H}^{-1})^T$
0: **for** $i = 0, B, 2B, \ldots, N_b B$ **do**
0:     **if** i % m = 0 **then**
0:         $g \leftarrow \frac{i}{m}$ {the group index}
0:         $\mathbf{C}_g \leftarrow$ init_codebook $[\mathbf{W}_{:,i:i+m-1}]$
0:     **end if**
0:     $\mathbf{W}_{:,i:i+m-1} \leftarrow \textsc{QuantGroup}(\mathbf{W}_{:,i:i+m-1})$
0:     $\mathbf{W}_{:,(i+B)} \leftarrow \mathbf{W}_{:,(i+B)} - \mathbf{E} \cdot [\mathbf{H}^{-1}]_{i:(i+B),(i+B):}$
0: **end for**=0

B.1. Comparison to AQLM

Additive Quantization for Language Models (Egiazarian et al., 2024) (AQLM) is a recent method that also uses vector quantization to compress LLMs to very low effective bit widths and achieves impressive bits per value vs accuracy results, as shown in Table 3 (due to differences in evaluation protocol, we can't compare to (Egiazarian et al., 2024) directly in Table 1). While both GPTVQ and AQLM employ VQ for LLM compression, our methods differ in several significant ways, which affects inference deployment and compression time, as detailed in this section.

AQLM uses larger vector dimension d, with d=8, scale their codebooks exponentially in d, similar to us. E.g., for 2-bit results AQLM uses codebooks with 2^{15} or 2^{16} 8-dimensional entries, where each entry is stored in FP16. While the authors have shown that these configurations can be employed on Nvidia® GPUs, codebooks of these sizes would be harder to employ efficiently on Arm® platforms. This is caused by the fact that many calls to the (5-bit) `TBL` instruction would be required, leading to significant additional latency during inference time. For example, decoding a single 16-bit index to an 8-bit FP16 would require $2 \times 8 \times 2^{11}$ 5-to-8-bit lookup tables (LUTs), where each lookup requires $2 \times 8 \times 11$ instructions to decode. Even on GPUs, our configurations have a clear edge over AQLM, as seen in Table 3 and in comparing Table 2 to Table 4 in (Egiazarian et al., 2024).

The full AQLM algorithm requires significant time to compress models. Compressing Llamav2-7B requires 24 hours on an A100, while GPTVQ takes between 30 minutes and 3 hours on a single H100 GPU. This is due to the fact that AQLM requires an expensive beam search and block-wise fine-tuning to achieve good accuracy, which add significantly to compression time. It should however be noted that our method becomes significantly slower for higher quantization dimensionality, mainly due to the EM codebook initialization.

The long runtime of AQLM is caused in part by a block-wise fine-tuning step. This step allows the model to correct intra-layer effects of quantization error. While GPTVQ has no mechanism to correct intra-layer error effects, its results are competitive with AQLM. AQLM without the additional fine-tuning step (i.e. Table 7 in the Appendix of (Egiazarian et al., 2024)), achieves a perplexity of 8.18 for the WikiText2 test set on Llamav2-7B, a degradation of nearly 1.5 points compared to 6.70 for GPTVQ under the same conditions.

C. GPTVQ Algorithm

Algorithm 2 QuantGroup: VQ quantization for a group of weights $\mathbf{W} \in \mathbb{R}^{r \times l}$ given the inverse Hessian $\mathbf{H^{-1}}$, the block size B, VQ dimensionality d

0: **function** QUANTGROUP($\mathbf{W}$)
0:     **for** $j = 0, d, 2d, \ldots, l$ **do**
0:         $P = j, \ldots, j + d - 1$
0:         $\mathbf{Q}_{:,P} \leftarrow$ VQ_quant $[\mathbf{W}_{:,P}, \mathbf{C}_g]$
0:         $\mathbf{E}_{:,P} \leftarrow (\mathbf{W}_{:,P} - \mathbf{Q}_{:,P}) [\mathbf{H}^{-1}]_P$
0:         $\mathbf{W}_{:,d-1:B} \leftarrow \mathbf{W}_{:,d-1:B} - \sum_{p=0}^{d-1} \mathbf{E}_{:,j+p} [\mathbf{H}^{-1}]_{p,d-1:i}$
0:     **end for**
0: **end function**=0

Table 4. **VQ configurations.** Group shape ($r \times c$) indicates (rows×columns)

bpv	d	b	group size	group shape	codebook bw
2.125	1D	2	256	(1×256)	8
2.125	2D	2	2,048	(4×256)	8
2.125	4D	2	65,536	(256×256)	8
2.25	1D	2	128	(1×128)	8
2.25	2D	2	1,024	(4×256)	8
2.25	4D	2	32,768	(128×256)	8
2.75	2D	2.5	2,048	(4×256)	8
3	2D	2.5	512	(2×256)	8
3.125	1D	3	8,192	(32×256)	8
3.125	2D	3	32,768	(128×256)	8
4.125	1D	4	1,024	(4 ×256)	8
4.125	2D	4	65,536	(256×256)	8

D. VQ Configurations

E. Extended results

F. Mean and standard deviation over multiple runs

Table 9. **Mean and standard deviation over 10 random seeds**. Setting used: Llamav2-7B, 2D VQ, 8-bit codebook.

BPV	Mean and Std. Dev.
3.125	5.82 ± 0.01
4.125	5.59 ± 0.01

G. Hyperparameter ablations

EM initialization To find seed centroids for EM initialization, we compare k-Means++ (Arthur and Vassilvitskii, 2007) to a quick and effective initialization method dubbed *Mahalanobis initialization*. In the latter method, we initialize EM for a matrix of N d-dimensional points $\mathbf{X}$ by first sorting all points by Mahalanobis distance (Bishop, 2006) to the mean of the data, then sampling k points spaced $\lfloor \frac{N}{k-1} \rceil$ apart from the sorted list. Intuitively, this method ensures that points are sampled at representative distances from the mean. Table 10 shows perplexity after GPTVQ for different EM initialization seed methods, and find that Mahalanobis initialization performs comparably to k-Means++, at increased speed.

EM iterations We explore the effect of the number of EM initialization iterations on the final perplexity of GPTVQ. Table 11 shows that even up to 100 iterations, results keep improving slightly, therefore we use 100 iterations as default.

Table 5. **Weight-only quantization results of Llama-1, Llama-2, Mistral, and Mixtral-MoE Models**. We report WikiText2 perplexity in this table; lower is better Models marked 'L1' or 'L2' denote Llama-v1 and Llama-v2, respectively. M denotes Mistral and 8x7B denotes Mixtral-MoE 8x7B.

		L1-7B	L1-13B	L1-30B	L1-65B
FP16		5.68	5.09	4.10	3.53
2.125 bpv (W2@g128)	RTN	1.9e3	781.20	68.04	15.08
	GPTQ	44.01	15.60	10.92	9.51
	AWQ	2.6e5	2.8e5	2.4e5	7.4e4
	OmniQuant	9.72	7.93	7.12	5.95
	GPTVQ 1D (ours)	16.29	**6.93**	**6.04**	**5.19**
	GPTVQ 2D (ours)	**9.64**	**6.58**	**5.63**	**4.91**
2.25 bpv (W2@g64)	RTN	188.32	101.87	19.20	9.39
	GPTQ	22.10	10.06	8.54	8.31
	AWQ	2.5e5	2.7e5	2.3e5	7.4e4
	OmniQuant	8.90	7.34	6.59	5.65
	GPTVQ 1D (ours)	16.64	**6.78**	**5.97**	**5.05**
	GPTVQ 2D (ours)	9.90	**6.43**	**5.56**	**4.86**
	GPTVQ 4D (ours)	**8.76**	**6.33**	**5.42**	**4.74**
3.125 bpv (W3@g128)	RTN	7.01	5.88	4.87	4.24
	GPTQ	6.55	5.62	4.80	4.17
	AWQ	6.46	5.51	4.63	3.99
	OmniQuant	6.15	5.44	4.56	3.94
	GPTVQ 1D (ours)	6.60	**5.34**	**4.48**	**3.85**
	GPTVQ 2D (ours)	**6.32**	**5.31**	**4.38**	**3.79**

Table 6. **Weight-only 4 bit quantization results of Llama-1, Llama-2, and Mistral-7B models**. We report WikiText2 perplexity in this table; lower is better. Models marked 'L1' or 'L2' denote Llama-v1 and Llama-v2, respectively. M-7B denotes Mistral.

		L1-7B	L1-13B	L1-30B	L1-65B	L2-7B	L2-13B	L2-70B	M-7B
FP16		5.68	5.09	4.10	3.53	5.47	4.88	3.31	5.25
4.125 bpv (W4@g128)	RTN	5.96	5.25	4.23	3.67	5.72	4.98	3.46	5.42
	GPTQ	5.85	5.20	4.23	3.65	5.61	4.98	3.42	5.35
	AWQ	5.81	5.20	4.21	3.62	5.62	4.97	-	-
	OmniQuant	**5.77**	5.17	4.19	3.62	**5.58**	**4.95**	-	-
	GPTVQ 1D (ours)	5.96	**5.15**	**4.18**	**3.60**	5.62	**4.97**	**3.39**	**5.32**
	GPTVQ 2D (ours)	5.94	5.20	**4.18**	3.64	**5.59**	**4.94**	**3.38**	**5.32**

Codebook compression Compared to FP16 codebooks, quantizing the entries to INT8 allows the group size to be reduced by half at the same overhead. We find that 8 bit quantization does not harm accuracy, while the smaller group size improves accuracy, as discussed in Appendix H.

Codebook update Table 12 includes an ablation on including codebook updates as described in Section 2.2. We find that, in all cases, updating the codebook after running Algorithm 2 improves final perplexity, at the expense of moderately increased (though still reasonable) run time. We thus include codebook update in all training runs.

Method runtime GPTVQ can quantize large language models efficiently. Exact runtime depends on model, quantization setting (groupsize, bitwidth, vq dimension), and several hyperparameters (EM iterations, codebook update iterations). As An indication of realistic run-times on a single H100: Llamav2-7B takes between 30 minutes and 1 hour, while Llamav2-70B takes between 3 and 11 hours.

H. Further codebook compression

While we find that 8 bit quantization of codebooks provides best results for the same overhead, we explore a different approach to codebook compression in this section.

For the case where $d = 1$, we could further compress the codebook $\mathbf{C}$ by stacking all codebooks for multiple blocks (e.g. all blocks in a tensor) and rank-reducing the resulting matrix. For a single tensor, $\mathbf{C}$ has shape $N_G \times k$, where N_G is the number of groups in the corresponding weight tensor, k is the number of centroids per codebook. We first sort values in each codebook in $\mathbf{C}$, and reassign the indices in $\mathbf{I}$ accordingly. Then, we perform SVD on $\mathbf{C}$, leading to matrices $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V}$, of shapes $N_G \times k$, $k \times k$ and $k \times k$, respectively. $\mathbf{U}' = \mathbf{U}\mathbf{\Sigma}$, and reduce the rank of this matrix to r, yielding a $N_G \times r$ shaped matrix $\mathbf{U}''$. We also reduce the rank of $\mathbf{V}$ accordingly, yielding $r \times r$ matrix $\mathbf{V}'$. Then, we perform gradient descent (GD) on the loss of equation 7, but with respect to the codebook tensor factors $\mathbf{U}''$ and $\mathbf{V}'$. In each GD step, $\widehat{\mathbf{C}}$ is created as $\widehat{\mathbf{C}} = \mathbf{U}''\mathbf{V}'^T$, and the rest of the codebook up procedure as described earlier is followed. Lastly, only the codebook tensor factor $\mathbf{U}''$ is quantized, as $\mathbf{V}'$ gives very little overhead. During inference, $\widehat{\mathbf{C}}$ is quantized per codebook after construction.

For higher dimensions, Tucker factorization could be employed. However, in this case there is no natural ordering in which to sort the elements of each codebook.

In table 15 we compare the effect of either rank reducing by 50%, or quantizing the codebook to 8-bit (our default approach), to keeping the codebook in FP16 and increasing the group size. In all three settings the overhead of the codebook is the same. We see that, for the same overhead, quantization gives best results. For this reason, and because codebook SVD does not easily apply to $d > 1$, we have not explored codebook SVD further, and instead use INT8 quantization as our default approach.

I. Blockwise data normalization

In order to lower the error of vector quantization, we apply blockwise data normalization to the data before the codebook initialization. For each group corresponding to a new codebook we perform element-wise division $\mathbf{W}_i \oslash \mathbf{S}_i$ of the weight sub-matrix matrix $\mathbf{W}_i$ by the corresponding scales $\mathbf{S}_i$. The scale is computed block-wise for every sub-row of $\mathbf{W}_i$, e.g. for a block size of 16, 32, or 64.

Given a set of blocks (sub-rows) $\mathbf{w}^{(i)}$, the scale $s^{(i)}$ for each of them is computed as $s^{(i)} = \max_j |w_j^{(i)}|$. In order to minimize the overhead, the scales are quantized to 4-bit integer.

We found that it is beneficial to perform quantization in log-scale to capture several orders of magnitudes in weights. The quantized scales are computed as $s_{int}^{(i)} = \lceil \frac{\log_2[s^{(i)}] - z}{a} \rfloor a$, where a is the quantization scale shared among the group of weights. In order to accurately represent zero in log-space which corresponds to unit scaling, we use the floating point offset z. In practice the value of z is shared within the columns of $\mathbf{W}$ and thus has negligible overhead. Finally the scaled sub-row is normalized as $\mathbf{w} \cdot 2^{-as_{int} - s_0}$, where $s_0 = \log_2(z)$. The scaled data is used for codebook initialization. The inverse scaling is applied at VQ decoding step.

Table 7. **LM-eval results of quantized Llama-v2 7B and 13B, and Llama-v3 8B models.**

	#Bits	Method	PIQA	ARC-e	Arc-c	BoolQ	HellaSwag	Winogrande	**Avg.↑**
Llama-v2-7B	FP16		79.11	74.58	46.25	77.74	75.99	69.14	70.47
	2.125 bpv (W2@g128)	RTN	51.09	27.95	25.00	41.13	26.57	49.88	36.94
		GPTQ	54.84	30.64	25.09	53.43	33.09	51.54	41.44
		VQ-1D	**61.21**	**38.76**	**24.66**	**62.78**	**45.78**	**53.83**	**47.84**
		VQ-2D	**71.33**	**57.41**	**32.94**	**65.60**	**59.85**	**64.72**	**58.64**
		VQ-4D	**73.34**	**60.44**	**34.39**	**65.50**	**63.99**	**65.04**	**60.45**
	2.25 bpv (W2@g64)	RTN	58.76	36.66	24.83	41.87	40.38	51.93	42.40
		GPTQ	60.83	39.02	25.17	59.33	45.82	55.49	47.61
		VQ-1D	**64.80**	**49.33**	**28.24**	**65.87**	**53.37**	**54.93**	**52.76**
		VQ-2D	**72.36**	**63.47**	**35.41**	**72.14**	**60.92**	**64.72**	**61.50**
		VQ-4D	**73.99**	**64.73**	**36.77**	**71.19**	**64.84**	**65.75**	**62.88**
	3.125 bpv (W3@g128)	RTN	76.77	70.50	42.92	71.71	73.96	67.64	67.25
		GPTQ	77.37	68.14	40.70	71.04	72.50	67.25	66.16
		VQ-1D	**77.86**	**68.64**	**40.96**	**73.85**	**72.29**	**67.80**	**66.90**
		VQ-2D	**77.64**	**73.15**	**43.17**	**74.22**	**72.61**	**69.06**	**68.31**
Llama-v2-13B	FP16		80.52	77.53	49.23	80.52	79.38	72.14	73.22
	2.125 bpv (W2@g128)	RTN	58.43	32.32	25.51	47.86	39.40	48.86	42.06
		GPTQ	59.52	40.15	27.65	57.06	41.56	53.43	46.56
		VQ-1D	**73.23**	**64.10**	**35.75**	**71.38**	**60.71**	**65.43**	**61.77**
		VQ-2D	**75.24**	**68.27**	**38.99**	**69.91**	**65.81**	**68.98**	**64.53**
		VQ-4D	**75.46**	**71.93**	**42.92**	**67.86**	**69.26**	**66.93**	**65.73**
	2.25 bpv (W2@g64)	RTN	61.59	41.58	25.43	49.79	48.24	51.85	46.41
		GPTQ	70.13	56.65	31.57	51.10	56.62	58.88	54.16
		VQ-1D	**72.36**	**67.63**	**37.37**	**74.13**	**62.89**	**65.27**	**63.28**
		VQ-2D	**74.97**	**67.63**	**40.53**	**69.24**	**67.11**	**69.30**	**64.80**
		VQ-4D	**76.66**	**69.87**	**43.00**	**74.68**	**70.81**	**69.69**	**67.45**
	3.125 bpv (W3@g128)	RTN	78.89	74.28	46.76	77.25	76.51	70.80	70.75
		GPTQ	79.33	75.84	47.01	78.90	77.16	70.40	71.44
		VQ-1D	**78.94**	**75.04**	**46.76**	**79.42**	**75.85**	**72.45**	**71.41**
		VQ-2D	**79.27**	**74.33**	**46.67**	**77.40**	**77.21**	**72.45**	**71.22**
Llama-v3-8B	FP16		79.9	80.1	50.4	-	60.2	72.8	68.6
	2.125 bpv (W2@g128)	RTN	53.1	24.8	22.1	-	26.9	53.1	36.0
		GPTQ	53.9	28.8	19.9	-	27.7	50.5	36.2
		VQ-1D	**56.58**	**35.10**	**18.26**	**60.00**	**38.25**	**57.06**	**41.05**
		VQ-2D	**69.48**	**62.58**	**29.01**	**72.29**	**43.05**	**65.51**	**53.93**
		VQ-4D	**71.93**	**69.19**	**32.68**	**69.45**	**45.62**	**67.17**	**57.32**
	2.25 bpv (W2@g64)	**VQ-1D**	**71.16**	**70.24**	**34.04**	**74.13**	**45.71**	**65.27**	**57.29**
		VQ-2D	**74.27**	**71.30**	**37.54**	**69.24**	**49.07**	**69.30**	**60.30**
		VQ-4D	**75.68**	**72.60**	**41.04**	**74.68**	**52.22**	**69.69**	**62.25**
	3.125 bpv (W3@g128)	RTN	62.3	32.1	22.5	-	29.1	54.7	40.2
		VQ-1D	**77.31**	**77.90**	**43.43**	**79.42**	**57.28**	**72.45**	**65.68**
		VQ-2D	**77.80**	**76.68**	**45.14**	**77.40**	**58.16**	**72.45**	**66.05**

Table 8. **LM-eval results of quantized Mistral-7B and Mixtral-8x7B models.**

	#Bits	Method	PIQA	ARC-e	Arc-c	BoolQ	HellaSwag	Winogrande	**Avg.↑**
Mistral-7B	FP16		82.10	79.59	53.92	83.58	81.07	73.88	75.69
	2.125 bpv (W2@g128)	RTN	53.05	29.42	26.62	38.56	29.26	49.57	37.75
		GPTQ	57.73	35.65	26.62	46.06	36.06	49.49	41.93
		VQ-1D	**55.22**	**35.94**	**25.51**	**54.01**	**34.35**	**52.01**	**42.84**
		VQ-2D	**73.78**	**69.02**	**37.80**	**76.57**	**64.52**	**65.35**	**64.51**
		VQ-4D	**75.90**	**71.63**	**41.98**	**69.85**	**68.59**	**66.46**	**65.73**
	2.25 bpv (W2@g64)	RTN	60.72	38.47	27.56	44.83	46.10	51.07	44.79
		GPTQ	65.83	46.21	30.20	62.11	50.64	55.56	51.76
		VQ-1D	**67.41**	**59.01**	**33.79**	**67.74**	**53.80**	**55.96**	**56.28**
		VQ-2D	**74.86**	**69.23**	**40.53**	**74.07**	**65.93**	**67.40**	**65.34**
		VQ-4D	**76.61**	**73.15**	**42.41**	**77.95**	**69.48**	**69.30**	**68.15**
	3.125 bpv (W3@g128)	RTN	80.79	74.62	48.46	80.00	78.66	68.19	71.79
		GPTQ	79.82	75.51	49.40	81.22	77.34	70.17	72.24
		VQ-1D	**78.84**	**75.29**	**47.87**	**79.57**	**75.32**	**69.30**	**71.03**
		VQ-2D	**81.12**	**78.70**	**51.02**	**82.39**	**78.05**	**72.06**	**73.89**
Mixtral-8x7B	FP16		83.46	73.74	55.89	84.74	82.45	75.30	75.93
	2.125 bpv (W2@g128)	RTN	51.90	27.27	25.85	47.98	27.07	49.64	38.29
		GPTQ	59.79	35.44	27.30	52.08	41.80	50.83	44.54
		VQ-1D	**68.93**	**50.93**	**33.02**	**62.51**	**52.52**	**61.17**	**54.85**
		VQ-2D	**76.39**	**57.87**	**38.91**	**74.95**	**67.03**	**71.03**	**64.36**
		VQ-4D	**78.13**	**65.57**	**46.42**	**78.59**	**72.40**	**71.11**	**68.70**
	2.25 bpv (W2@g64)	RTN	62.08	38.68	28.41	54.46	44.40	53.12	46.86
		GPTQ	66.05	42.93	28.58	50.12	49.59	55.41	48.78
		VQ-1D	**69.42**	**50.55**	**36.09**	**64.95**	**59.51**	**63.93**	**57.41**
		VQ-2D	**77.42**	**62.12**	**42.66**	**72.39**	**70.74**	**68.90**	**65.71**
		VQ-4D	**79.16**	**67.68**	**48.04**	**76.09**	**73.43**	**71.11**	**69.25**
	3.125 bpv (W3@g128)	RTN	81.50	68.77	50.60	80.92	79.71	72.93	72.40
		GPTQ	80.85	69.32	52.05	81.35	78.40	74.43	72.73
		VQ-1D	**80.90**	**71.34**	**52.73**	**84.83**	**77.62**	**73.64**	**73.51**
		VQ-2D	**82.59**	**72.94**	**54.86**	**84.46**	**80.61**	**74.82**	**75.05**

Table 10. **Effect of EM initialization**. Setting used: Llamav2-7B, 2D 3-bit VQ, blocksize 2048.

Lookup method	BPV	Setting	PPL	Time (s)
1D 3B 1024	3.125	Mahalanobis	6.05	605
		K++	6.16	3328
2D 3B 16384	3.125	Mahalanobis	5.65	756
		K++	5.63	3168
1D 4B 2048	4.125	Mahalanobis	5.86	1272
		K++	5.88	2116
2D 4B 65536	4.125	Mahalanobis	5.59	3816
		K++	5.57	6644

Table 11. **Effect of number of EM interations**. Setting used: BLOOM-560m 2D 3-bit VQ with blocksize 4096, perplexity on WikiText2 test set.

EM iterations	PPL
10	24.49
30	24.18
50	24.12
75	24.11
100	24.09

Table 12. **Effect of codebook fine-tuning on final PPL for Llamav2-7B**.

d	b	gs	Update	PPL	Runtime (s)
1	2	512	N	43.14	625
			Y	14.02	1857
	3	1024	N	6.05	712
			Y	6.01	1916
2	2	2048	N	8.64	723
			Y	8.21	1335
	3	8192	N	5.93	1585
			Y	5.88	2195

Table 13. **Effect of scaling block size on perplexity for Llamav2-7B**. d: VQ-dimension; b: VQ bitwidth per dimension; gs: block size; Codebooks are quantized to 8 bits.

d	b	gs	Scaling BS					
			None	128	64	32	16	8
1	2	512	14.01	16.74	2744.9	480.8	15.36	13.79
	3	1024	6.02	5.97	6.00	5.87	5.82	5.72
2	2	2048	8.23	8.38	8.04	7.97	7.56	6.89
	3	8192	5.91	5.82	5.78	5.73	5.74	5.66

Table 14. **Effect of scaling on perplexity for different models**. Configurations with equal overhead with or without the scaling are considered. d: VQ-dimension; b: VQ bitwidth per dimension; gs: block size; Codebooks are assumed to be quantized to 8 bit.

d	b	gs	Scale	Llamav2-7B	Llamav2-13B	Mistral-7B	Mixtral-8x7B
1	2	256	N	14.01	7.34	15.03	8.56
		512	Y	171.29	7.44	87.60	8.11
	3	512	N	5.98	5.21	5.76	4.60
		1024	Y	6.01	5.17	5.77	4.59
2	2	2048	N	8.23	6.69	10.98	6.73
		4096	Y	8.49	6.50	10.28	6.37
	3	8192	N	5.91	5.19	8.63	4.52
		16384	Y	5.56	5.11	5.53	4.30

Table 15. **Choices in experimental setup leading to comparable bits per value**. d: VQ-dimension; b: VQ bitwidth per dimension; gs: block size; Q: 8-bit codebook quantization yes/no; SVD: codebook SVD yes/no. BPV: bits per value.

d	b	gs	Q	SVD	BPV	PPL
1	2	512	N	N	2.125	14.01
		256	Y	N	2.125	11.57
		256	N	Y	2.125	44.99
	3	1024	N	N	3.125	6.01
		512	Y	N	3.125	5.98
		512	N	Y	3.125	5.98
2	2	4096	N	N	2.125	8.37
		2048	Y	N	2.125	8.23
	3	16384	N	N	3.125	5.93
		8192	Y	N	3.125	5.87