ReALLM: A GENERAL FRAMEWORK FOR LLM COMPRESSION AND FINE-TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce ReALLM, a novel approach for compression and memory-efficient adaptation of pre-trained language models that encompasses most of the post-training quantization and fine-tuning methods for a budget of < 4 bits. Pre-trained matrices are decomposed into a high-precision low-rank component and a vector-quantized latent representation (using an autoencoder). During the fine-tuning step, only the low-rank components are updated. Our results show that pre-trained matrices exhibit different patterns. ReALLM adapts the shape of the encoder (small/large embedding, high/low bit VQ, etc.) to each matrix. ReALLM proposes to represent each matrix with a small embedding on b bits and a neural decoder model $\mathcal{D}_\phi$ with its weights on b_ϕ bits. The decompression of a matrix requires only one embedding and a single forward pass with the decoder. Our weight-only quantization algorithm yields the best results on both commonsense reasoning tasks (C4, WikiText-2) for a budget of 3 bits *without* any training. With a budget of 2 bits, ReALLM achieves state-of-the-art performance on understanding tasks (ARC, PiQA, Winogrande, MMLU) as well as generation tasks (TruthfulQA) after fine-tuning on a single partition of C4 dataset. Additionally, ReALLM is practical in terms of inference latency and memory.

1 INTRODUCTION

Large Language Models (LLMs) based on transformer architectures (Vaswani et al., 2017) have gained significant interest, particularly with open-source models like LLaMA (Touvron et al., 2023), Falcon (Almazrouei et al., 2023), and Gemma (Team et al., 2024). These models can be used for inference or local fine-tuning, but full fine-tuning is expensive due to high GPU memory requirements. For instance, fine-tuning LLaMA-65B needs over 780 GB of memory (Dettmers et al., 2023a).

To address memory constraints, quantization of model weights, activations, and gradients is used. Quantization-Aware Training (QAT) is common in computer vision (Courbariaux et al., 2015; Liu et al., 2020; Gholami et al., 2022), but training LLMs from scratch is impractical. Post-training quantization (PTQ) is a viable alternative (Dettmers et al., 2022; Frantar et al., 2022), with recent studies focusing on scalar quantization (SQ) and some exploring vector quantization (VQ) (Tseng et al., 2024; Egiazarian et al., 2024).

Combining quantization with Parameter Efficient Fine-Tuning (PEFT) methods like LoRA (Hu et al., 2021) improves efficiency (Dettmers et al., 2023a). However, PTQ faces challenges due to outliers in LLM weights, leading to quantization errors (Kim et al., 2023; Dettmers et al., 2023b).

This paper introduces ReALLM - for **Re**sidual **A**utoencoder **LLM** - a method for LLM PTQ and fine-tuning. Pre-trained LLM matrices are decomposed into a 16-bit remainder and a compressed part using a VQ autoencoder (Van Den Oord et al., 2017). Only low-rank components are fine-tuned, while quantized elements remain static. Our approach adapts to matrix patterns, leveraging computer vision techniques to exploit structured patterns in pre-trained weight matrices, as illustrated in Figure 2 and Figure 3. This exploitation of patterns aligns with findings from Dettmers et al. (2023b); Heo et al. (2024), where matrix sensitivity patterns are analogous to our Figure 2. Our contributions are the following:

- We introduce ReALLM, a method that employs a novel autoencoder and a residual pipeline to efficiently compress pre-trained LLM matrices.

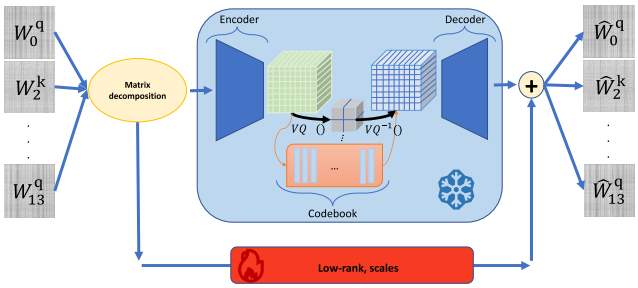


Figure 1: ReALLM; during the fine-tuning step only low-rank and scales are updated

- We demonstrate that state-of-the-art PTQ approaches (Lin et al., 2023; Shao et al., 2023; Tseng et al., 2024; Egiazarian et al., 2024) and fine-tuning methods (Hu et al., 2021; Dettmers et al., 2023a; Guo et al., 2023; Li et al., 2023; Liao and Monz, 2024) are special cases of ReALLM.
- We propose a preprocessing step involving scaling and column permutations of matrices to mitigate quantization errors from outliers, adapting the autoencoder scheme to matrix patterns.
- Our approach achieves the best reported results for 3 and 2-bit Post-Training Quantization (PTQ) by fine-tuning end-to-end with block-wise error reduction.

2 Related work

LLMs adapters With the advent of high-performance open-source LLMs, full fine-tuning has become impractical, leading to the development of parameter-efficient fine-tuning (PEFT) methods. Notable techniques include prefix tuning (Li and Liang, 2021), selective fine-tuning (Guo et al., 2021), and Low Rank Adapter (LoRA) (Hu et al., 2021). LoRA retains pre-trained matrices while adding a low-rank component, significantly reducing the number of tunable parameters. Our work employs DoRA (Liu et al., 2024), which enhances fine-tuning by decomposing weights into magnitude and direction, leading to improved performance with minimal computational effort.

Quantization LLM compression primarily uses quantization techniques. Early methods like ZeroQuant (Yao et al., 2022) and nuQmm (Park et al., 2022) rounded weights to the nearest quantization level. Later developments handled outliers through higher bitwidth quantization (Xiao et al., 2023; Dettmers et al., 2022; Kim et al., 2023; Dettmers et al., 2023b). Methods similar to ReALLM combine quantization with low-rank decomposition (Dettmers et al., 2023a; Guo et al., 2023; Li et al., 2023; Liao and Monz, 2024). QLoRA (Dettmers et al., 2023a) integrates PEFT and quantization, while LoftQ (Li et al., 2023) and LQ-LoRA (Guo et al., 2023) minimize quantization errors using SVD of pre-trained weights. ApiQ (Liao and Monz, 2024) optimizes both LoRA components and quantization parameters for the entire model. Recent studies combine weight and activation quantization for enhanced efficiency (Liu et al., 2023; Nrusimha et al., 2024).

Block/layer-wise tuning PTQ (Frantar et al., 2022) introduced a strategy using a large-scale solver to minimize layer-wise quadratic error, crucial for low bit-width quantization (Tseng et al., 2024; Egiazarian et al., 2024). QuIP# (Tseng et al., 2024) applies random rotations and optimal lattice quantizers to pre-trained matrices, while AQLM (Egiazarian et al., 2024) uses adaptive codebooks and blockwise fine-tuning. Both methods achieve stable results in compressing LLMs to 2 bits per parameter.

3 Method

Low-rank/sparse decomposition Starting from a pre-trained LLM matrix $W \in \mathbb{R}^{p\times q}$, W is decomposed into a residual component $R \in \mathbb{R}^{p\times q}$ and a quantized matrix Q (represented with b bits per coordinate). Only the residual matrix is retained with high bit accuracy and further optimized in

the fine-tuning phase using a small calibration dataset. Any efficient matrix decomposition can fit into the residual part: butterfly (Dao et al., 2019), sparse outliers (Dettmers et al., 2023b; Lin et al., 2023), etc. We use a low-rank component $R = L_1(L_2)^t$, analogous to the *data-free* method in Guo et al. (2023). The aim is to identify Q, L_1, and L_2 that solve:

$$\min_{Q,L_1,L_2} \|W - (Q + L_1(L_2)^t)\|.$$

QLoRA Dettmers et al. (2023a) provides a suboptimal solution by setting $L_1 = 0$.

Mixed-autoencoder configuration An autoencoder in ReALLM is parameterized by neural networks ψ and ϕ, with $\mathcal{E}_\psi : \mathbb{R}^{p\times q} \to \mathbb{R}^{e_0\times e_1\times e_2}$ and $\mathcal{D}_\phi : \mathbb{R}^{e_0\times e_1\times e_2} \to \mathbb{R}^{p\times q}$, where $e_0e_1e_2 \ll pq$. Previous works focused on applying the same quantization strategy directly to the pre-trained matrix. ReALLM adapts the autoencoder to the type and shape of the matrix, using HNeRV (Chen et al., 2023) for efficient training (see Figure 1).

Vector Quantization (VQ) ReALLM uses a data-free vector quantization method based on k-means to store the embedding $\mathcal{E}_\psi(W)$ with few bits. The embedding is divided into buckets of dimension d, and codewords are optimized using k-means clustering. The total number of bits required is $bd \cdot \frac{e_0e_1e_2}{d}$, with additional memory for the codebook.

Quantization pre-processing Before quantization, appropriate scaling is performed to handle outliers. ReALLM uses column permutations to create spatial regularity, clustering outliers to improve compression (see Algorithm 2). The memory overhead for storing permutations is negligible (see Figures 2 and 3).

ReALLM: a new LLM format ReALLM represents each matrix of size $p \times q$ with a small embedding of size $e_0 \times e_1 \times e_2$ on b bits and a neural decoder model $\mathcal{D}_\phi$ with c parameters on b_ϕ bits. This format speeds up the decoding step compared to diffusion-based approaches. The set of hyper-parameters for ReALLM includes the rank r, the shape of the latent representation (e_0, e_1, e_2), the number of bits and the bucket dimension in the VQ (b, d), and the number of parameters and bits of the decoder (c, b_ϕ).

4 Experimental Validation

We conducted thorough ablation studies on LLaMA-2-7b (Touvron et al., 2023) and Mistral-2-7b (Jiang et al., 2023), testing ReALLM on LLaMA-2 and LLaMA-3 models (7B and 13B parameters) across various language generation tasks. We compared ReALLM with other quantization methods for 3 and 2 bits per coordinate. On an Nvidia A40 GPU (with 46GB memory), the entire computation (PTQ + fine-tuning) takes 90 hours for a LLaMA2-7B model.

Memory and Latency Analysis Our primary objective is to maximize accuracy for a given model size. We report in Appendix D the memory used by the quantized model and the on-the-fly dequantization memory overhead, the prefilling and the generation acceleration at constant GPU RAM on A40 GPU.

Language Generation Tasks For continual language modeling, we train on a single partition of the C4 (Raffel et al., 2020) dataset for half an epoch and use a sequence length of 4096 for training only. We use a sequence length of 2048 for both WikiText-2 (Merity et al., 2016) and C4 evaluation. Our main baselines are LQ-LoRA (Guo et al., 2023), QuIP# (Tseng et al., 2024), and AQLM (Egiazarian et al., 2024). In Table 1, we evaluate the perplexity of ReALLM on the respective validation datasets of C4 and WikiText-2 for a single run. Our *data-free* version of ReALLM achieves state-of-the-art metrics for 3 bit quantization. However, for a budget of 2 bits, quantization errors are larger, and our results show that fine-tuning (both block-wise and end-to-end) is needed to further improve performance.

Few-Shot Tasks Following HuggingFace's Open LLM Leaderboard[1], we measure zero-shot accuracy on ARC (Clark et al., 2018), PiQA (Tata and Patel, 2003), and Winogrande (Sakaguchi et al., 2021), via the LM Evaluation Harness (Gao et al., 2021). We report results in Table 2 and compute

[1] https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

Table 1: Perplexity (↓) on the validation dataset for LLaMA2-7B and LLaMA2-13B, with a sequence length of 2048.

Method	Calib.	#seq.	#bits	C4		WikiText-2	
				7B	13B	7B	13B
LLaMA2			16	6.97	6.46	5.47	4.48
GPTQ	C4	128	3	7.89	7.00	6.29	5.42
AWQ	Pile	192	3	7.84	6.94	6.24	5.32
Omniquant	Wiki.	128	3	7.75	6.98	6.03	5.28
LQ-LoRA	Wiki.+C4	10k	3	7.88	–	6.48	–
LoftQ	Wiki.	128	3	–	–	5.63	5.13
ApiQ[PTQ]	Wiki.	128	3	7.84	6.88	6.19	5.18
QuaRot[A16W3]	Wiki.	128	3	–	–	6.09	5.37
QuIP# (no ft)	NA	NA	3	7.85	6.98	6.19	5.34
QuIP#	RedP.	6k	3	7.32	6.72	5.79	**5.10**
ReALLM (no ft)	NA	NA	3	7.72	6.91	6.10	5.27
ReALLM	C4	1024	3	**7.27**	**6.69**	**5.77**	5.14
LoftQ	Wiki.	128	2	–	–	7.85	7.69
ApiQ	Wiki.	128	2	–	–	7.46	6.29
QuIP#	RedP.	6k	2	8.35	**7.45**	6.66	5.74
AQLM	RedP.	4096	2	8.56	7.51	**6.64**	**5.65**
ReALLM	C4	1024	2	**8.28**	7.50	6.69	5.72

Table 2: Accuracy (↑) in LM Eval (acc, not acc_norm) for LLaMA-2.

Task	LLaMA-2-7B				LLaMA-2-13B			
	LLaMA-2 16-bits	AQLM 2-bits	QuIP# 2-bits	ReALLM 2-bits	LLaMA-2 16-bits	AQLM 3-bits	QuIP# 3-bits	ReALLM 3-bits
ARC-challenge	43.52	33.55	34.63	**35.15**	48.32	43.63	44.02	**47.01**
ARC-easy	76.26	62.79	64.60	**68.56**	78.48	73.51	72.45	**75.96**
PiQA	78.07	73.54	75.12	**75.73**	80.01	77.78	78.40	**78.67**
Winogrande	69.22	64.61	64.89	**66.46**	72.13	67.56	69.13	**70.96**
Average	66.77	58.62	59.81	**61.47**	69.74	65.62	66.00	**68.15**

Table 3: Accuracy (↑) in LM Eval (acc, not acc_norm) for 2bits quantization of Llama3-8B.

Task	Llama3-8B	ReALLM	Quip#	QuaROT	LoftQ	RTN	GPTQ	AWQ	QuIP	DB-LLM	PB-LLM	BiLLM	OmniQuant	SmoothQuant
ARC-c	50.4	**38.2 ± 1.4**	30.8	24.6	20.4	21.9	20.5	21.3	21.3	28.2	17.5	17.7	19.3	20.0
ARC-e	80.1	**74.9 ± 0.8**	57.1	57.9	26.1	24.7	25.0	29.0	29.0	59.1	31.7	36.0	36.1	26.3
Piqa	79.9	**76.3 ± 0.9**	67.5	65.5	53.8	53.1	52.8	52.9	52.9	68.9	52.5	56.1	59.0	54.6
Winogrande	72.8	**65.8 ± 1.6**	62.4	60.6	47.8	51.1	49.6	51.7	51.7	60.4	50.4	51.0	51.9	50.3

Table 4: 5-shot evaluation on MMLU.

Base-model	Llama3-8b	Llama3-8b	ReALLM Llama2-7b	ReALLM Llama2-7b	AWQ Llama2-7b	RTN Llama2-7b	QA-LORA Llama2-7b	ReALLM Llama2-7b	AWQ Llama2-7b	QA-LORA Llama2-7b
#bits	16	16	2	3	3	3	3	2	2	2
MMLU	66.6	45.3	46.5±0.6	**44.8±0.6**	44.7	42.33	41.11	**35.1±0.6**	27.4	33.2

the average on the 4 mentioned tasks. For all LLM sizes, ReALLM provides a notable advantage with respect to AQLM (Egiazarian et al., 2024) and QuIP# (Tseng et al., 2024). We include evaluations on 5-shots MMLU and ThruthfulQA to better demonstrate the performance of our compressed LLM on both Llama2-7b and Llama3-8b in Tables 3, 4 and 10. ReALLM outperforms all other tested quantization methods.

5 Conclusion

We present ReALLM, a weight-only PTQ method that achieves state-of-the-art results on LLMs at 2, and 3 bits budget. Our (low-rank) fine-tuning approach enables one to fine-tune language models with 13 billions parameters on a *single* GPU with less than 40 GB of RAM.

Large context sequence lengths result in large KV-cache memory consumption during inference, and PTQ is a promising approach for compressing KV-cache activations (Hooper et al., 2024; Ashkboos et al., 2024). Concurrently to our work, Trukhanov and Soloveychik (2024) propose a quantization method based on permutations of rows from K and V matrices. We are currently studying how to adapt ReALLM to KV-cache quantization, and how to combine it with activation quantization.

REFERENCES

Almazrouei, E., Alobeidli, H., Alshamsi, A., Cappelli, A., Cojocaru, R., et al. (2023). The falcon series of open language models. *arXiv preprint*.

Ashkboos, S., Mohtashami, A., Croci, M. L., Li, B., Jaggi, M., et al. (2024). Quarot: Outlier-free 4-bit inference in rotated llms. *arXiv preprint*.

Bengio, Y. (2013). Estimating or propagating gradients through stochastic neurons. *arXiv preprint*.

Chen, H., Gwilliam, M., Lim, S.-N., and Shrivastava, A. (2023). Hnerv: A hybrid neural representation for videos. In *Conf. on Computer Vision and Pattern Recognition*.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., et al. (2018). Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint*.

Courbariaux, M., Bengio, Y., and David, J.-P. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*.

Dao, T., Gu, A., Eichhorn, M., Rudra, A., and Ré, C. (2019). Learning fast algorithms for linear transforms using butterfly factorizations. In *International Conf. on machine learning*.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. (2022). Gpt3. int8: 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2023a). Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*.

Dettmers, T., Svirschevski, R. A., Egiazarian, V., Kuznedelev, D., Frantar, E., et al. (2023b). Spqr: A sparse-quantized representation for near-lossless llm weight compression. In *International Conf. on Learning Representations*.

Egiazarian, V., Panferov, A., Kuznedelev, D., Frantar, E., Babenko, A., et al. (2024). Extreme compression of large language models via additive quantization. *arXiv preprint*.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. (2022). Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint*.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., et al. (2021). A framework for few-shot language model evaluation.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., et al. (2022). A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*. Chapman and Hall/CRC.

Guo, D., Rush, A. M., and Kim, Y. (2021). Parameter-efficient transfer learning with diff pruning. In *Annual Meeting of the Association for Computational Linguistics and the International Joint Conf. on Natural Language Processing*.

Guo, H., Greengard, P., Xing, E., and Kim, Y. (2023). Lq-lora: Low-rank plus quantized matrix decomposition for efficient language model finetuning. In *International Conf. on Learning Representations*.

Heo, J. H., Kim, J., Kwon, B., Kim, B., Kwon, S. J., et al. (2024). Rethinking channel dimensions to isolate outliers for low-bit weight quantization of large language models.

Hooper, C., Kim, S., Mohammadzadeh, H., Mahoney, M. W., Shao, Y. S., et al. (2024). Kvquant: Towards 10 million context length llm inference with kv cache quantization. *arXiv preprint*.

Hu, E. J., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., et al. (2021). Lora: Low-rank adaptation of large language models. In *International Conf. on Learning Representations*.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., et al. (2023). Mistral 7b. *arXiv preprint*.

Kim, S., Hooper, C., Gholami, A., Dong, Z., Li, X., et al. (2023). Squeezellm: Dense-and-sparse quantization. *arXiv preprint*.

Li, X. L. and Liang, P. (2021). Prefix-tuning: Optimizing continuous prompts for generation. In *Annual Meeting of the Association for Computational Linguistics and the International Joint Conf. on Natural Language Processing*.

Li, Y., Yu, Y., Liang, C., Karampatziakis, N., He, P., et al. (2023). Loftq: Lora-fine-tuning-aware quantization for large language models. In *International Conf. on Learning Representations*.

Liao, B. and Monz, C. (2024). Apiq: Finetuning of 2-bit quantized large language model. *arXiv preprint*.

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., et al. (2023). Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint*.

Liu, J., Gong, R., Wei, X., Dong, Z., Cai, J., et al. (2023). Qllm: Accurate and efficient low-bitwidth quantization for large language models. In *International Conf. on Learning Representations*.

Liu, S.-Y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., et al. (2024). Dora: Weight-decomposed low-rank adaptation. *arXiv preprint*.

Liu, Z., Shen, Z., Savvides, M., and Cheng, K.-T. (2020). Reactnet: Towards precise binary neural network with generalized activation functions. In *European Conf. in Computer Vision*.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2016). Pointer sentinel mixture models. In *International Conf. on Learning Representations*.

Nrusimha, A., Mishra, M., Wang, N., Alistarh, D., Panda, R., et al. (2024). Mitigating the impact of outlier channels for language model quantization with activation regularization. *arXiv preprint*.

Park, G., Park, B., Kim, M., Lee, S., Kim, J., et al. (2022). Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models. *arXiv preprint*.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., et al. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. (2021). Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*.

Shao, W., Chen, M., Zhang, Z., Xu, P., Zhao, L., et al. (2023). Omniquant: Omnidirectionally calibrated quantization for large language models. In *International Conf. on Learning Representations*.

Tata, S. and Patel, J. M. (2003). Piqa: An algebra for querying protein data sets. In *Conf. on Scientific and Statistical Database Management*.

Team, G., Mesnard, T., Hardin, C., Dadashi, R., Bhupatiraju, S., et al. (2024). Gemma: Open models based on gemini research and technology. *arXiv preprint*.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., et al. (2023). Llama: Open and efficient foundation language models. *arXiv preprint*.

Trukhanov, N. and Soloveychik, I. (2024). Accurate block quantization in llms with outliers. *arXiv preprint*.

Tseng, A., Chee, J., Sun, Q., Kuleshov, V., and De Sa, C. (2024). Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. *arXiv preprint*.

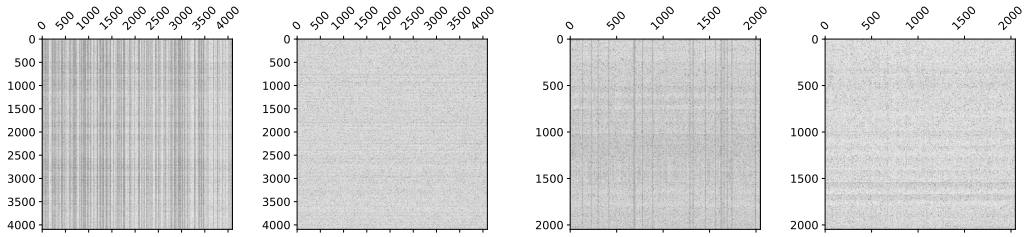
Van Den Oord, A., Vinyals, O., et al. (2017). Neural discrete representation learning. *Advances in neural information processing systems*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., et al. (2017). Attention is all you need. *Advances in neural information processing systems*.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., et al. (2023). Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conf. on Machine Learning*.

Yao, Z., Yazdani Aminabadi, R., Zhang, M., Wu, X., Li, C., et al. (2022). Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*.

A METHODOLOGY DETAILS



(a) Mistral-7B (Jiang et al., 2023) (b) Gemma-2B (Team et al., 2024)

Figure 2: Pre-trained matrix from the first layer 'q' (left; with "structures"), and from the last layer 'q' (right) for Mistral-7B (Jiang et al., 2023). Stronger vertical patterns appear in the first blocks.

Table 5: Quantization and fine-tuning approaches as particular case of ReALLM (with a rank r, and a budget of b bits for VQ in dimension d) for a matrix of size $p \times q$.

Method	rank r	AE	Latent	VQ dim. (d)	#bits
LoRA (Hu et al., 2021)	64	-	$(p, q, 1)$	1	16
GPTQ (Frantar et al., 2022)	0	-	$(p, q, 1)$	1	4
QLoRA (Dettmers et al., 2023a)	64	-	$(p, q, 1)$	1	4
LQ-LoRA (Guo et al., 2023)	64	-	$(p, q, 1)$	1	3
QuIP# (Tseng et al., 2024)	0	Rotation	$(p, q, 1)$	8	2
AQLM (Egiazarian et al., 2024)	0	-	$(p, q, 1)$	8	2
ReALLM	64	Trainable	(e_0, e_1, e_2)	4	2

Table 6: Comparison of several LLM format for m matrices of size $p \times q$, and a budget of b bits per coordinate. ReALLM uses a decoder model with c parameters trained on b_ϕ bits, and a rank r.

Method	LoRA	VQ only (like AQLM)	ReALLM
Matrix representation	$(p \times q) \cdot 16$	$(p \times \frac{q}{d}) \cdot b \cdot d$	$(e_0 \times \frac{e_1}{d} \times e_2) \cdot b \cdot d$
Codebook	–	$2^{bd} \cdot d \cdot 16$	$2^{bd} \cdot d \cdot 16$
Decoder	–	–	cb_ϕ
Low-rank	$(2 \times r \times \min(p, q)) \cdot 16$	–	$(2 \times r \times \min(p, q)) \cdot 16$
Total bit cost	$16(pq + 2r\min(p, q)) \cdot m$	$(bpq + 2^{bd+4}d) \cdot m$	$cb_\phi + 32r\min(p, q) + m(16d2^{bd} + e_0e_1e_2b)$

(a) Mistral-7B (Jiang et al., 2023) (b) Llama2-7B (Touvron et al., 2023)

Figure 3: Reconstruction (Frobenius norm) error for layer of type "q" for all blocks for Mistral-7B (Jiang et al., 2023). QuIP# (Tseng et al., 2024) does not take advantage of the structures in the first blocks.

B ALGORITHM DESCRIPTION

Algorithm 1: Pseudo-code for ReALLM with block-wise and end-to-end fine-tuning

Input : Number of end-to-end fine-tuning steps T, Number of block-wise fine-tuning steps K, Number of blocks n, Shape of the latent space (e_0, e_1, e_2), Number of weights in the decoder c, Number of bits for the decoder weights b_ϕ, Number of VQ bits per dimension b, VQ dimension d, Rank r ;

Initialize

  Get pre-trained matrices $\{W^q, W^k, W^v, W^o, W^{gate}, W^{up}, W^{down}\}$ for all n blocks ;

end

`/* Block-wise fine-tuning */`

for $j = 0, \dots, n-1$ **do**

  $B_j = \{W^q, W^k, W^v, W^o, W^{gate}, W^{up}, W^{down}\}[block = j]$;

  $output_j = forward\_pass(B_j)$ `/* get non-quantized output */`

  for $l \in \{q, k, v, o, gate, up, down\}$ **do**

    $L1^l_j, L2^l_j = svd\_decomposition(W^l_j, rank = r)$;

    $W^l_j = W^l_j - L1^l_j (L2^l_j)^t$;

    $\mathcal{E}_\psi(W^l_j), \mathcal{D}_\phi{}^l_j = autoencoder(W^l_j, e_0, e_1, e_2, c, b_\phi)$ `/* latent representation and decoder */`

    $\mathcal{E}_\psi(W^l_j), inv\_permut^l_j = permute(\mathcal{E}_\psi(W^l_j))$ `/* with Algorithm 2 */`

    $\mathcal{E}_\psi(W^l_j) = normalize(\mathcal{E}_\psi(W^l_j))$ `/* with NF-normalization (Dettmers et al., 2023a; Guo et al., 2023) */`

    $codebook^l_j = \text{K-means}(\mathcal{E}_\psi(W^l_j), b, d)$;

    $codes^l_j = get\_index\_nn(\mathcal{E}_\psi(W^l_j), codebook^l_j)$ `/* get nearest neighbor index in` $codebook^l_j$ `*/`

    $W^l_j \leftarrow \{codes^l_j, codebook^l_j, \mathcal{D}_\phi{}^l_j, inv\_permut^l_j, L1^l_j, L2^l_j\}$;

    $dora^l_j = DoRA(W^l_j, L1^l_j, L2^l_j)$ `/* get DoRA scale */`

  end

  $dora\_quantized\_output_j = forward\_pass\_quantized(\{dora^l_j, L1^l_j, L2^l_j, W^l_j\}_{l \geq 0})$ `/* get output after quantization and DoRA */`

  $L_j = \|output_j - dora\_quantized\_output_j\|^2$;

  for $k = 0, \dots, K-1$ **do**

    Optimize $\{dora^l_j, L1^l_j, L2^l_j\}_{l \geq 0}$ with gradient descent to minimize L_j;

  end

end

`/* End-to-end fine-tuning */`

for $t = 0, \dots, T-1$ **do**

  Optimize $\{dora^l_j, L1^l_j, L2^l_j\}_{l,j \geq 0}$ with gradient descent ;

end

Algorithm 2: permutation function

Input : Matrix w of size $128 \times q$;

for $j = 0, \dots, q-1$ **do**

  $column_j = w[:, j]$;

  $indx_j = get\_index\_nn(column_j, w[:, j+1 : q])$ `/* get the nearest neighbor index of current` $column_j$`, among the rest of un-permuted columns` $w[:, j+1 : q]$ `*/`

  Permute $w[:, j+1]$ and $w[:, indx_j]$;

  Save the inverse of the permutation index in inv_permut ;

end

Output : w, inv_permut

C ABBLATION STUDIES

Decoder's Weight Ablation In Table 7, we present ablation experiment results on the type of decoder weight quantization. For a decoder parameter $c = 7.2 \times 10^6$, we performed a quantization-aware training approach, optimizing weights quantized to $b_\phi = 6$ bits using the straight-through estimator (Bengio, 2013). We also tested a post-training quantization method where the decoder weights are quantized with a round-to-nearest (RTN) approach at the end of the training steps. This experiment shows that under the same parameters, QAT gives better performance than the respective PTQ approach. Furthermore, for a reduced number of bits (2.82 vs 3), ReALLM yields a smaller quantization error compared to the scalar quantization NF3 (Dettmers et al., 2023a; Guo et al., 2023) on the layer "q0" of Mistral-7b.

Table 7: Reconstruction (Frobenius norm) error for layer of type "Q" inside the first block of Mistral-7b model, for patches of size 512×512 using a constant embedding size of $(e_0, e_1, e_2) = (16, 16, 16)$, and a varying quantization strategy (during the decoder training, i.e. QAT, or after the training, i.e. PTQ).

Method	# parameters c ($\times 10^6$)	b_ϕ	bit budget	Error
NF3(Guo et al., 2023)	–	–	3	0.84
PTQ	7.2	6	2.82	1.78
QAT	7.2	6	2.82	**0.69**

Table 8: Reconstruction (Frobenius norm) error and computational overheads for layers of type "q" and "o" inside blocks 30 and 20 of the Lamma-7b model, for patches of size 512×512 using a constant embedding size of $(e_0, e_1, e_2) = (16, 16, 16)$, and a varying encoder quantization b_ϕ and #parameters c for a total budget of #bits $= 2$.

Weights quantization b_ϕ	4	5	6	32
#params c, $\times 10^6$	7.3	6	5	0.95
Error for q30 (↓)	27.42	**24.82**	25.8	43.56
Error for o20 (↓)	20.04	**18.18**	18.93	31.81
Training time	1h15'	1h13'	1h11'	1h05
Decoding time (ms)	11 ± 12	11 ± 10	11 ± 8.7	8.9 ± 3.8

Permutation Ablation We experimentally observed that permutations are indispensable for both AE and VQ components of ReALLM. In Figure 4, we report the Frobenius error and the learning rate during the autoencoder's training for layer "q" of Llama2-7b. The figure demonstrates that without permutation, the training is unstable, even with smaller learning rates. In Figure 5, we report the Frobenius error between the quantized layer "up" with the VQ component, both with and without column permutations. Permutations show a clear improvement in the quantization in both Llama2-7B and Llama2-13B.

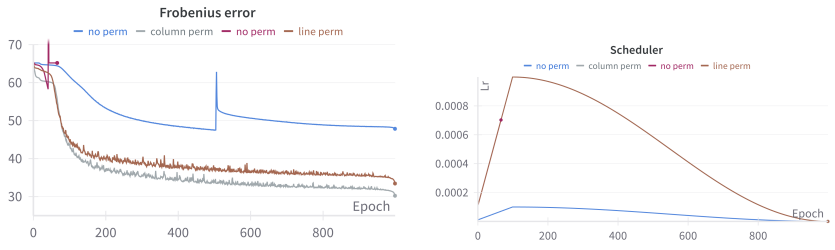


Figure 4: ReALLM's decoder training curves on Llama2-7B (Touvron et al., 2023) layer "q5" for different pre-processings and learning rates for a 2-bits budget. Blue / purple: no permutation, max-lr=0.0001 / max-lr=0.001. Gray / brown: column / line permutation, max-lr=0.001.

(a) Llama2-7B (b) Llama2-13B

Figure 5: Reconstruction (Frobenius norm) error for layer type "up" for all blocks, with ReALLM's VQ both with and without column permutation.

D Latency and memory overhead analysis

Our primary objective is to maximize accuracy for a given model size. Indeed, pushing the limits of LLM quantization is a crucial subject as it suggests that the architecture of LLMs could be further refined to be lighter while remaining efficient, providing an important guideline for research teams and startups with fewer computational resources.

That being said, we provide latency and memory overhead analysis. We report the memory used by the quantized model and the on-the-fly dequantization memory overhead, the prefilling and the generation acceleration at constant GPU memory (see Table 9)

We measure acceleration relative to a full-precision model for encoding and generating 2k tokens, we compare a model quantized using ReALLM with a model in FP16. To do this, we use the largest batch size that fits on the GPU for both the quantized model and the non-quantized model. This analysis is conducted using an A100 GPU.

Table 9: Latency and memory overhead analysis on A100 GPU.

	LLaMA-2-7b		LLaMA-2-13b	
#bits	2	3	2	3
Memory usage (GB)	1.8	2.4	3.1	4.7
Dequant. overhead (GB)	0.9	0.7	1.3	1.1
Prefiling speedup	1.42x	1.35x	1.42x	1.37x
Generation speedup	1.23x	1.15x	1.24x	1.15x

E Few-shots tasks

We include evaluations on 5-shots MMLU and ThruthfulQA to better demonstrate the performance of our compressed LLM on both Llama2-7b and Llama3-8b in Tables 3, 4 and 10. ReALLM outperforms all other tested quantization methods.

Table 10: 0-shot evaluation on TruthfulQA, multiple choices and generation (acc).

Base-model	ReALLM (2-bits) Llama2-7B	ReALLM (3-bits) Llama2-7B
mc1	24.2 ± 1.5	27.2 ± 1.6
mc2	38.3 ± 1.4	41.6 ± 1.4
BLEU	34.5 ± 1.7	35.0 ± 1.7
ROUGE-1	34.4 ± 1.7	35.3 ± 1.7
ROUGE-2	30.1 ± 1.6	31.1 ± 1.6
ROUGE-L	33.7 ± 1.7	35.0 ± 1.7