Low Rank Quantization-Aware Training for LLMs

Yelysei Bondarenko [1] Riccardo Del Chiaro [1] Markus Nagel [1]

Abstract

In this paper we propose **LR-QAT** – a lightweight and memory-efficient QAT algorithm for LLMs. LR-QAT employs several components to save memory without sacrificing performance: (a) low-rank quantization-aware reparameterization; (b) downcasting operation using fixed-point or double-packing and (c) checkpointing. Unlike most related work, our method (i) is inference-efficient, leading to no additional overhead compared to traditional PTQ; (ii) can be seen as a general extended pre-training framework, meaning that the resulting model can still be utilized for any downstream task afterwards; (iii) is orthogonal to most of recent PTQ methods and thus can be seamlessly combined with them. We apply LR-QAT to the LLaMA-1/2/3 and Mistral model families and validate its effectiveness on several downstream tasks. Our method outperforms most of recent LLM quantization approaches and reaches the same model performance as full-model QAT at the fraction of its memory usage. Specifically, we can train a 7B LLM on a single consumer grade GPU with 24GB memory. Our source code is available at https://github.com/qualcomm-ai-research/LR-QAT.

1. Introduction

In recent years, large language models (LLMs) have emerged as a powerful tool for a plethora of natural language processing tasks. As these models continue to grow in size and capability, addressing their ever increasing computational and memory demands becomes crucial for practical deployment, especially when considering resource-constrained edge devices.

One of the most effective methods to tackle this problem is neural network quantization, which uses low-bit precision for weight and activation tensors. While recent post-training quantization (PTQ) methods can help with decreasing the model size and improving the computational efficiency of LLMs, they typically lead to subpar performance, especially in the case of low-bit (≤ 4) quantization. Quantization-aware training (QAT), conversely, yields significantly better model performance compared to PTQ. However, due to extreme model sizes of modern LLMs, using traditional QAT is very computationally expensive and requires a prohibitively high GPU memory usage, making it impractical.

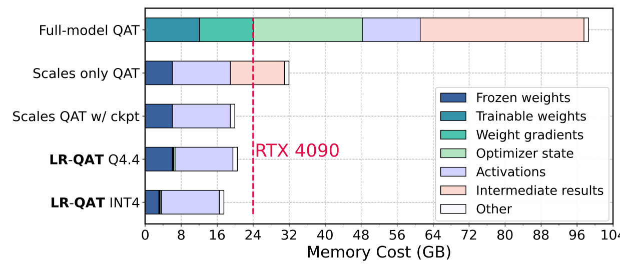

Figure 1: Memory requirements for training with various QAT techniques on LLaMA-2 7B, assuming batch size 1, sequence length 1024, $r = 32$, and BF16 compute data type. 'Intermediate results' refer to the results of some intermediate computations, *e.g.* after rounding/clipping in (1), which are saved in memory for the backward pass.

Inspired by parameter-efficient fine-tuning (PEFT) and low-rank adaptation (LoRA) literature, we propose **Low-Rank Quantization-Aware Training** (**LR-QAT**) – a lightweight memory-efficient and inference-efficient QAT algorithm for LLMs. LR-QAT reduces the memory requirements of training a 7B LLM from >98GB of GPU memory to <21GB (cf. Figure 1) without degrading the predictive performance compared to traditional full-model QAT, making it possible to train such models on a single consumer grade GPU. Unlike most related work that combines low-rank adaptation with quantization, our method is also inference-efficient: after the training is complete, the auxiliary matrices are naturally absorbed into the quantized weight tensor without loss of accuracy and no extra overhead at inference time. Additionally, LR-QAT does not relax the quantization constraints and is therefore applicable for any weight quantization granularity. LR-QAT is positioned as a general *extended pre-training* method, as opposed to being strictly a fine-tuning method – the resulting model is a low-bit general pre-trained LLM,

[1] Qualcomm AI Research, Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc. Correspondence to: {ybond, rdelchia, markusn}@qti.qualcomm.com.

Published at ICML 2024 Workshop on Efficient Systems for Foundation Models, Vienna, Austria.

that can be utilized for any task afterwards, including fine-tuning on specific downstream tasks and combining with LoRA adapters.

LR-QAT introduces and combines several innovations designed to reduce memory use without sacrificing performance: (1) A form of **QAT with low-rank reparameterization**, in which we put the low-rank weights in the integer domain such that they are aware of the quantization grid of the pre-trained weights and can be seamlessly fused at inference into a single low-bit integer matrix. (2) A **downcasting operator** that represents the frozen pre-trained weights as low-bit INT-b ($b \leq 4$) double-packed into INT8 or as fixed-point values stored in INT8. (3) Finally, we combine the proposed quantization formulation with gradient checkpointing to avoid aggressive memory spikes from storing some of the intermediate results in memory for the backward pass.

We apply LR-QAT to the LLaMA-2/3 and Mistral model families and demonstrate its effectiveness on several general language modeling datasets and zero-shot evaluation on some of the common reasoning downstream tasks. Our method outperforms common PTQ approaches and reaches the same model performance as full model QAT at the fraction of its memory usage.

2. Method

In this section discuss the components of LR-QAT followed by a formal definition of LR-QAT. A detailed background and related work discussion can be found in Appendix A.

QAT with low-rank adapters Let's recall how traditional QAT works. Given a linear layer with a weight matrix $\boldsymbol{W} \in \mathbb{R}^{m \times k}$ and assuming b-bit symmetric uniform affine quantization, the quantization is simulated as follows:

$$\widehat{\boldsymbol{W}} := \mathbf{s} \cdot \operatorname{clip}\left(\left\lfloor \frac{\boldsymbol{W}}{\mathbf{s}} \right\rceil,\ n,\ p\right), \tag{1}$$

where $n = -2^{b-1}$, $p = 2^{b-1} - 1$, weights $\boldsymbol{W}$ are trainable parameters and the quantization scale $\mathbf{s}$ can be either fixed or also learned. To be able to backpropagate through round-to-nearest operation in (1), it is common to use *straight-through estimator* (STE, Bengio et al. 2013), where it is assumed that $\frac{\partial \lfloor t \rceil}{\partial t} = 1$. When applied to LLMs, it's easy to see that this procedure is very expensive: we have to learn a comparable number of parameters to those used for pre-training, leading to excessive memory usage.

To make our approach more practical we *freeze* the pre-training weights $\boldsymbol{W}$ (denote $\boldsymbol{W_0}$) and introduce low-rank adapters $\boldsymbol{A} \in \mathbb{R}^{m \times r}$, $\boldsymbol{B} \in \mathbb{R}^{r \times k}$, $r \ll \min\{m, k\}$. We have to be careful where exactly those adapters are placed. After the training is complete, we want $\boldsymbol{A}$ and $\boldsymbol{B}$ to be seamlessly integrated into a single b-bit integer matrix $\boldsymbol{W}_{\mathbb{Z}}$ without loss of accuracy to facilitate the efficient inference. To accommodate that, we put the auxiliary matrices inside the rounding operator as follows

$$\widehat{\boldsymbol{W}} := \mathbf{s} \cdot \operatorname{clip}\left(\left\lfloor \frac{\boldsymbol{W_0}}{\mathbf{s}} + \frac{\alpha}{r}\boldsymbol{A}\boldsymbol{B} \right\rceil,\ n,\ p\right), \tag{2}$$

where we are using STE assumption for the rounding operation to compute the gradients of the loss w.r.t. $\boldsymbol{A}$, $\boldsymbol{B}$ and $\mathbf{s}$. We further employ a scaling factor α/r used in LoRA (Hu et al., 2021) to reduce the need to retune the hyper-parameters as we vary the rank r. After training is complete, (2) can be represented as regular fixed point tensor, $\widehat{\boldsymbol{W}} = \mathbf{s} \cdot \boldsymbol{W}_{\mathbb{Z}}$, without any extra effort or loss of accuracy and therefore enabling efficient inference without any extra overhead. Note that this is different to most of the literature, such as QLoRA (Dettmers et al., 2024), where adapters are placed outside of the quantization function (such as $\mathbf{y} = \widehat{\boldsymbol{W}}\mathbf{x} + \boldsymbol{A}\boldsymbol{B}\mathbf{x}$) and typically stored in higher precision formats such as BF16.

Downcasting operator The formulation (2) is already significantly more memory efficient compared to standard full model QAT (cf. (1)). We don't need to compute neither gradients w.r.t. weights $\boldsymbol{W}$ nor the respective first or second-order momentum terms for Adam-based optimizers, and only need to do so for the auxiliary matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, which is noticeably more affordable provided $r \ll \min\{m, k\}$.

Given that the weight matrix $\boldsymbol{W_0}$ is frozen, the next natural step to further reduce the memory utilization is to store it in a lower-precision format. One could apply downcasting of $\boldsymbol{W_0}$ directly in (2), however notice how those weights are divided by the scale $\mathbf{s}$ at every forward pass, which generally has to be stored in a high-precision format to guarantee stable training. To simplify further, we propose the following variant of low-rank QAT:

$$\widehat{\boldsymbol{W}} := \mathbf{s} \cdot \operatorname{clip}\left(\left\lfloor \frac{\boldsymbol{W_0}}{\mathbf{s_0}} + \frac{\alpha}{r}\boldsymbol{A}\boldsymbol{B} \right\rceil,\ n,\ p\right), \tag{3}$$

where we use the initial scale $\mathbf{s_0}$ instead of learned scale $\mathbf{s}$ inside the rounding operator. Now the entire fraction $\boldsymbol{W_0}/\mathbf{s_0}$ is fixed and we can store it in a lower-precision format. Note that the scale $\mathbf{s}$ outside of the clipping operator can still be learned. Empirically, we found that (3) performs consistently on par or slightly better compared to (2).

During training the pre-trained weights are represented and stored as follows

$$\boldsymbol{\Phi}_0 := \varphi\left(\frac{\boldsymbol{W_0}}{\mathbf{s_0}}\right), \tag{4}$$

where $\varphi(\cdot)$ – a *downcasting operator* that encapsulates a choice of different numeric formats or other pre-processing computations. In the simplest form, $\varphi(\cdot)$ would cast the input to one of pre-existing floating-point formats, such as FP16, BF16, FP8 etc.

Inspired by traditional fixed point quantization, we also explore integer representations for $\varphi(\cdot)$. Specifically, $\varphi(x) = \text{clip}(\lfloor x \rceil,\ n,\ p)$ corresponds to a standard b-bit integer quantization and can be stored as INT-b number. We denote this approach $\varphi = \text{INT-}b$ for brevity. In addition to that, in case of low-bit quantization ($b \leq 4$), which is a primary focus, two INT-b numbers can be *double-packed* into a single INT8 number, leading to further memory savings. This is helpful as many common deep learning frameworks like PyTorch, at the time of writing this paper, do not natively support low-bit formats such as INT4.

Using $\varphi = \text{INT-}b$ naturally leads to aggressive memory reduction by only keeping the integer part of (clipped) $\boldsymbol{W_0}/\mathbf{s_0}$. In our preliminary experiments, we found that this setting, combined with the standard initialization for $\boldsymbol{A}$ and $\boldsymbol{B}$ used in (Hu et al., 2021), did not work as well compared to $\varphi = \text{BF16}$. This indicates the importance of keeping some information of the fractional part of $\boldsymbol{W_0}/\mathbf{s_0}$ and potentially the need for better initialization of auxiliary matrices.

We address this problem in two distinct ways: We adapt and experiment with a variant of SVD-based initialization for low-rank matrices $\boldsymbol{A}$, $\boldsymbol{B}$ proposed in (Li et al., 2023) before we apply a downcasting operator to $\boldsymbol{W_0}/\mathbf{s_0}$, to capture some of the information of its fractional part.

Another way is to use INT8 storage type, use b bits to represent the integer part as before, but use the remaining $8 - b$ bits for storing the approximate fractional part ($2 \leq b \leq 7$). In other words, we represent $\boldsymbol{\Phi}_0$ using fixed-point numbers. Specifically, assuming the the rest of the computation is performed in BF16, we define the downcasting and the corresponding upcasting operators as follows:

$$\begin{aligned} \varphi(x) &= \text{INT8}\left(\left\lfloor 2^{8-b} \cdot \text{clip}\left(x,\ n,\ p\right)\right\rceil\right), \\ \varphi^{-1}(y) &= \text{BF16}(y)/2^{8-b}. \end{aligned} \tag{5}$$

Following common fixed-point notation (Oberstar, 2007), which uses Qn.m for n integer bits and m fractional bits, we will refer to (5) as $\varphi = \text{Q}b.(8-b)$. Specifically, in this work we will be mainly focusing on $b \in \{3, 4\}$, which corresponds to Q4.4 and Q3.5, respectively.

Gradient checkpointing Note that both in the original LoRA paper (Hu et al., 2021) and in the related work like QLoRA (Dettmers et al., 2024), there is no need to compute the product $\boldsymbol{AB}$. Instead, those matrices are multiplied with the activations $\mathbf{x}$ as $\boldsymbol{A}(\boldsymbol{B}\mathbf{x})$. However, we do compute a product $\boldsymbol{AB}$ in (3), and in an naive implementation of our method, this product as well as the results of some of the intermediate computations (e.g., after rounding and clipping) will be automatically kept in memory for the backward pass, leading to increased memory usage. To prevent this, we employ gradient checkpointing (Chen et al., 2016) on (3). In other words, we recompute the quantizer function in the backward pass, leading to a slight runtime overhead but avoiding significantly increased memory usage.

LR-QAT Using the components described above, we define LR-QAT for a single layer with a (pre-trained) weight matrix $\boldsymbol{W_0}$ as follows

$$\widehat{\boldsymbol{W}} := \mathbf{s} \cdot \text{clip}\left(\left\lfloor \boldsymbol{\Phi}_0 + \frac{\alpha}{r}\boldsymbol{AB} \right\rceil,\ n,\ p\right), \tag{6}$$

where $\mathbf{s}$ – trainable or frozen quantization scale with the initial value of $\mathbf{s_0}$, $\boldsymbol{A}$, $\boldsymbol{B}$ – trainable rank r auxiliary matrices, $\boldsymbol{\Phi}_0 := \varphi(\boldsymbol{W_0}/\mathbf{s_0})$ – frozen representation of the original pre-trained weights, φ is the downcasting operation. To avoid excessive memory allocation for the results of intermediate computations in (6) involving the product $\boldsymbol{AB}$, we apply checkpointing on $\widehat{\boldsymbol{W}}$. After the training is complete, low-rank adapters are naturally integrated into a single integer matrix $\boldsymbol{W}_{\mathbb{Z}} = \text{clip}(\cdots)$ without loss of accuracy. Note, while we presented our method for symmetric quantization which is commonly used for weights (Nagel et al., 2021), it can equally be applied for asymmetric quantization by adding a zero offset z outside the rounding operation as shown in (7).

3. Experiments

We assess the effectiveness of LR-QAT by conducting experiments on LLaMA 7B (Touvron et al., 2023a), LLaMA-2 7B/13B (Touvron et al., 2023b), LLaMA-3 8B (AI@Meta, 2024), and Mistral-0.1 7B (Jiang et al., 2023). We focus here on comparing with related literature for weight-only quantization and comparing to full-model QAT. Ablation studies on the impact of rank r, the downcasting operator $\varphi(\cdot)$, the initialization of auxiliary matrices $\boldsymbol{A}$, $\boldsymbol{B}$ as well as weight-activation quantization are discussed in Appendix B.

Experimental setup We experiment with weight-only and weight-activation quantization.Our default settings are INT4 / INT3 per-channel (denoted 'pc') and group-wise weight quantization with a group size of 128 (denoted 'g128'). We quantize all linear layers, except the classification head. Following the previous work (Frantar et al., 2022; Xiao et al., 2023; Shao et al., 2023; Liu et al., 2023b), we evaluate quantized models by reporting the WikiText-2 (Merity et al., 2016) perplexity and zero-shot accuracy on a set of common sense reasoning tasks including BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), Winogrande (Sakaguchi et al., 2021), ARC (Clark et al., 2018), and HellaSwag (Zellers et al., 2019).

We train LR-QAT on a small subset of SlimPajama (Soboleva et al., 2023) and initialize it with round-to-nearest quan-

Table 1: **Weight-only quantization results for LLaMA and Mistral models**. We report WikiText-2 test set perplexity (lower is better) and average zero-shot accuracy (higher is better). Models marked 'L1'/'L2'/'L3', and 'M' denote LLaMA-1/2/3 and Mistral, respectively. Numbers marked in bold are SOTA or on par (within 0.05). [§]Uses asymmetric weight quantization. [*]Uses a maximum sequence length of 1024 for evaluation.

# Bits	Method	WikiText-2 Perplexity ↓					Avg. zero-shot accuracy ↑				
		L1-7B	L2-7B	L2-13B	L3-8B	M-7B	L1-7B	L2-7B	L2-13B	L3-8B	M-7B
FP16		5.68	5.47	4.88	6.14	5.25	69.68	70.47	73.18	74.22	75.69
W4 pc	RTN	6.33	6.14	5.21	7.53	5.91	68.51	68.88	71.73	72.19	73.44
	GPTQ[§]	6.13	5.83	5.13	-	-	64.95	-	-	-	-
	AWQ	6.08	6.15	5.12	-	-	-	-	-	-	-
	OmniQuant[§]	**5.86**	5.74	**5.02**	-	-	-	-	-	-	-
	LLM-QAT	10.9[*]	-	-	-	-	**68.63**	-	-	-	-
	PEQA (our impl.)	**5.86**	**5.71**	**5.03**	7.51	5.56	68.49	69.23	72.51	72.79	73.73
	LR-QAT (ours)	**5.84**	**5.66**	**5.03**	**6.78**	**5.46**	68.54	**69.95**	**73.19**	**73.84**	**74.44**
W3 pc	RTN	12.88	26.73	8.71	34.10	9.49	54.66	43.87	55.01	47.46	64.58
	GPTQ[§]	8.06	8.37	6.44	-	-	-	-	-	-	-
	AWQ	11.88	24.00	10.45	-	-	-	-	-	-	-
	OmniQuant[§]	6.49	6.58	**5.58**	-	-	-	-	-	-	-
	PEQA (our impl.)	6.56	6.45	5.73	26.20	6.51	65.75	65.44	69.81	51.05	71.02
	LR-QAT (ours)	**6.27**	**6.13**	**5.54**	**8.12**	**6.03**	**66.60**	**67.66**	**71.22**	**70.46**	**71.87**
W4 g128	RTN	6.05	5.78	5.04	6.96	5.49	68.93	69.75	72.94	72.30	75.07
	GPTQ[§]	5.85	**5.61**	**4.98**	-	-	-	-	-	-	-
	AWQ	5.81	**5.62**	**4.97**	-	-	-	-	-	-	-
	OmniQuant[§]	**5.77**	**5.58**	**4.95**	-	-	-	-	-	-	-
	PEQA (our impl.)	**5.75**	5.67	5.02	6.89	5.48	**69.19**	69.64	72.80	72.99	73.34
	LR-QAT (ours)	**5.75**	**5.59**	**4.97**	**6.57**	**5.37**	**69.15**	**69.88**	**72.91**	**73.66**	**75.28**
W3 g128	RTN	7.96	7.61	6.20	15.11	6.77	63.50	63.20	67.60	57.74	69.35
	GPTQ[§]	6.55	6.29	5.42	-	-	-	-	-	-	-
	AWQ	6.46	6.24	**5.32**	-	-	-	-	-	-	-
	OmniQuant[§]	**6.15**	**6.03**	**5.28**	-	-	-	-	-	-	-
	PEQA (our impl.)	6.22	6.05	5.58	9.64	**5.85**	66.66	**68.10**	70.29	67.19	72.21
	LR-QAT (ours)	**6.17**	**5.99**	**5.32**	**7.74**	**5.80**	**66.81**	67.98	**71.51**	**70.48**	**72.41**

tization (RTN) for which we set the initial scale $\mathbf{s}_0$ based on minimizing the L^p-norms between quantized and unquantized weights. All detailed hyperparameters of our experiments are in Appendix C.

3.1. Weight-only quantization

For weight-only quantization, we compare LR-QAT with GPTQ (Frantar et al., 2022), AWQ (Lin et al., 2023), OmniQuant (Shao et al., 2023), and our implementation of PEQA (Kim et al., 2024), where we use symmetric weight quantization and following the same experimental setup and RTN initialization for a fair comparison. We summarize our weight-only quantization results in Table 1. As we can see, in almost all cases LR-QAT outperforms or is on par with prior weight-only quantization methods across various LLM families and quantization settings, including both per-channel and group-wise quantization. In a few cases, especially in case of group-wise quantization, our method did not outperform OmniQuant. However, OmniQuant uses asymmetric quantization which provides an extra degrees of freedom compared to symmetric quantization. While this can improves accuracy it also leads to additional inference overhead (Nagel et al., 2021). Additionally, PTQ techniques like OmniQuant are orthogonal to our method and can be used as an initialization of LR-QAT.

Table 2: A comparison of the proposed method ($\varphi = \mathsf{Q4.4}$) with the full-model QAT on LLaMA-2 7B with W4 and W3 per-channel quantization.

Method	GPU mem.	WikiText-2 ↓		Zero-shot acc. ↑	
		W4pc	W3pc	W4pc	W3pc
Full-model QAT	98.5 GB	5.76	6.14	68.71	66.91
LR-QAT	**20.5 GB**	**5.66**	**6.13**	**69.95**	**67.66**

3.2. Comparison with full model QAT

We compare our method with a vanilla full model QAT (Esser et al., 2020). For full model QAT, we follow the same training setup and hyper-parameter tuning as for our method. As we can see in Table 2, training with LR-QAT leads to a slightly better predictive performance at a significantly lower memory usage compared to vanilla QAT.

4. Conclusions

In this paper we propose LR-QAT, a lightweight and memory-efficient QAT algorithm for LLMs which enables training a 7B LLM on a single consumer grade GPU with 24GB memory. Inspired by PEFT methods, we introduce a low-rank reparameterization that is aware of the quantization grid. We further reduce the memory requirements by introducing a downcasting operator to fixed-point and applying checkpointing. In almost all cases our method outperforms common PTQ approaches and reaches the same model performance as full model QAT at the fraction of its memory usage.

References

AI@Meta. Llama 3 model card, 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. Post-training 4-bit quantization of convolution networks for rapid-deployment. *arXiv preprint arXiv:1810.05723*, 2018.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2020.

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.

Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. Understanding and overcoming the challenges of efficient transformer quantization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7947–7969, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.627. URL https://aclanthology.org/2021.emnlp-main.627.

Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. Quantizable transformers: Removing outliers by helping attention heads do nothing. *Advances in Neural Information Processing Systems*, 36, 2024.

Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Zeroq: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13169–13178, 2020.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. Quip: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems*, 36, 2024.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. In *ICCV Workshops*, pages 3009–3018, 2019.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 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 preprint arXiv:1803.05457*, 2018.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*, 2022.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*, 2023.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *International Conference on Learning Representations (ICLR)*, 2020.

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, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept*, page 8, 2021.

Sylvain Gugger, Lysandre Debu, Thomas Wolf, Philipp Schmid, Zachary Mueller, and Sourab Mangrulkar. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate, 2022.

Han Guo, Philip Greengard, Eric P Xing, and Yoon Kim. Lq-lora: Low-rank plus quantized matrix decomposition for efficient language model finetuning. *arXiv preprint arXiv:2311.12023*, 2023.

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International conference on machine learning*, pages 1737–1746. PMLR, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

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.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. Improving post training neural quantization: Layer-wise calibration and integer programming. *arXiv preprint arXiv:2006.10518*, 2020.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

Hyesung Jeon, Yulhwa Kim, and Jae-joon Kim. L4q: Parameter efficient quantization-aware training on large language models via lora-wise lsq. *arXiv preprint arXiv:2402.04902*, 2024.

Yongkweon Jeon, Chungman Lee, Kyungphil Park, and Ho-young Kim. A frustratingly easy post-training quantization scheme for llms. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14446–14461, 2023.

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.

Jeonghoon Kim, Jung Hyun Lee, Sungdong Kim, Joonsuk Park, Kang Min Yoo, Se Jung Kwon, and Dongsoo Lee. Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization. *Advances in Neural Information Processing Systems*, 36, 2024.

Olga Kovaleva, Saurabh Kulshreshtha, Anna Rogers, and Anna Rumshisky. Bert busters: Outlier dimensions that disrupt transformers. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3392–3405, 2021.

Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. Owq: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 13355–13364, 2024.

Jangwhan Lee, Minsoo Kim, Seungcheol Baek, Seok Joong Hwang, Wonyong Sung, and Jungwook Choi. Enhancing computation efficiency in large language models through weight and activation quantization. *arXiv preprint arXiv:2311.05161*, 2023a.

Jung Hyun Lee, Jeonghoon Kim, Se Jung Kwon, and Dongsoo Lee. Flexround: Learnable rounding based on element-wise division for post-training quantization. In *International Conference on Machine Learning*, pages 18913–18939. PMLR, 2023b.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. URL `https://aclanthology.org/2021.emnlp-demo.21`.

Yixiao Li, Yifan Yu, Chen Liang, Pengcheng He, Nikos Karampatziakis, Weizhu Chen, and Tuo Zhao. Loftq: Lora-fine-tuning-aware quantization for large language models. *arXiv preprint arXiv:2310.08659*, 2023.

Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. Brecq: Pushing the limit of post-training quantization by block reconstruction. *arXiv preprint arXiv:2102.05426*, 2021.

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.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. *arXiv preprint arXiv:2405.04532*, 2024.

Jing Liu, Ruihao Gong, Xiuying Wei, Zhiwei Dong, Jianfei Cai, and Bohan Zhuang. Qllm: Accurate and efficient low-bitwidth quantization for large language models. *arXiv preprint arXiv:2310.08041*, 2023a.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. Llm-qat: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888*, 2023b.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Yan Luo, Yangcheng Gao, Zhao Zhang, Jicong Fan, Haijun Zhang, and Mingliang Xu. Long-range zero-shot generative deep network quantization. *Neural Networks*, 166:683–691, 2023.

Eldad Meller, Alexander Finkelstein, Uri Almog, and Mark Grobman. Same, same but different: Recovering neural network quantization error through weight factorization. In *International Conference on Machine Learning*, pages 4486–4495. PMLR, 2019.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1325–1334, 2019.

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 (ICML)*, 2020.

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

Erick L Oberstar. Fixed-point representation & fractional math. *Oberstar Consulting*, 9:19, 2007.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems (NeuRIPS)*. 2019.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

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.

Xuan Shen, Zhenglun Kong, Changdi Yang, Zhaoyang Han, Lei Lu, Peiyan Dong, Cheng Lyu, Chih-hsiang Li, Xuehang Guo, Zhihao Shu, et al. Edgeqat: Entropy and distribution guided quantization-aware training for the acceleration of lightweight llms on the edge. *arXiv preprint arXiv:2402.10787*, 2024.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama, June 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

Mingjie Sun, Xinlei Chen, J Zico Kolter, and Zhuang Liu. Massive activations in large language models. *arXiv preprint arXiv:2402.17762*, 2024.

Hanlin Tang, Yifu Sun, Decheng Wu, Kai Liu, Jianchen Zhu, and Zhanhui Kang. Easyquant: An efficient data-free quantization algorithm for llms. *arXiv preprint arXiv:2403.02775*, 2024.

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.

Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, Ruihao Gong, Shanghang Zhang, Qi Zhang, Fengwei Yu, and Xianglong Liu. Outlier suppression: Pushing the limit of low-bit transformer language models. *arXiv preprint arXiv:2209.13325*, 2022.

Xiuying Wei, Yunchen Zhang, Yuhang Li, Xiangguo Zhang, Ruihao Gong, Jinyang Guo, and Xianglong Liu. Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling. *arXiv preprint arXiv:2304.09145*, 2023.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.

Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhensu Chen, Xiaopeng Zhang, and Qi Tian. Qa-lora: Quantization-aware low-rank adaptation of large language models. *arXiv preprint arXiv:2309.14717*, 2023.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 27168–27183. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/adf7fa39d65e2983d724ff7da57f00ac-Paper-Conference.pdf.

Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. Rptq: Reorder-based post-training quantization for large language models. *arXiv preprint arXiv:2304.01089*, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Cheng Zhang, Jianyi Cheng, George A Constantinides, and Yiren Zhao. Lqer: Low-rank quantization error reconstruction for llms. *arXiv preprint arXiv:2402.02446*, 2024.

Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. Improving neural network quantization without retraining using outlier channel splitting. In *International conference on machine learning*, pages 7543–7552. PMLR, 2019.

Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

A. Background and related work

Neural network quantization is one of the most powerful ways to reduce model footprint, data transfer and compute requirements. By quantizing a model, high bit-width floating point weights and activations can be represented using low-bit numbers. On top of that, using low-bit fixed-point representations, such as INT8, one can further reduce energy consumption since the fixed-point operations are more efficient than their floating-point counterparts. Quantizing to 8 bits or lower, however, typically introduces quantization noise in the model, resulting in a potential drop in accuracy.

In this section we provide a brief overview of uniform affine quantization and a summary of recent methods for LLM quantization. We will discuss some of the trade-offs of those techniques. Finally, we touch upon the challenges of LLM quantization and some of the limitations of current approaches.

Uniform affine quantization We use the following definition of the quantization function:

$$\widehat{\mathbf{x}} := q\left(\mathbf{x};\, s, z, b\right) = s \cdot \Big(\underbrace{\mathrm{clip}\Big(\Big\lfloor \frac{\mathbf{x}}{s} \Big\rceil + z; n, p\Big)}_{=:\,\mathbf{x}_{\mathbb{Z}}} - z\Big), \quad (7)$$

where $n = -2^{b-1}$, $p = 2^{b-1} - 1$, $\mathbf{x}$ denotes the quantizer input (i.e., network weights or activations), s the higher precision quantization scale, z the integer zero point, and b the bitwidth. $\lfloor\cdot\rceil$ denotes the round-to-nearest-integer operator. Quantization parameters s, z can be shared across the components of $\mathbf{x}$. One can see that such a quantizer approximates an original floating point vector as $\mathbf{x} \approx s \cdot (\mathbf{x}_{\mathbb{Z}} - z)$, where each element in $\mathbf{x}_{\mathbb{Z}}$ is a b-bit integer value. This quantization scheme is called *uniform affine* or *asymmetric* quantization (Hubara et al., 2017; Krishnamoorthi, 2018; Zhou et al., 2016) and it is one of the most commonly used quantization schemes because it allows for efficient implementation of fixed-point arithmetic. In the case of *symmetric* quantization, we restrict the quantization grid to be symmetric around $z = 0$.

Post-training quantization methods Post-training quantization (PTQ) algorithms take a pre-trained high precision (FP32 / FP16 / BF16) network and convert it directly into a fixed-point network without the need for the original training pipeline (Banner et al., 2018; Cai et al., 2020; Choukroun et al., 2019; Hubara et al., 2020; Krishnamoorthi, 2018; Li et al., 2021; Meller et al., 2019; Nagel et al., 2019; 2020; Zhao et al., 2019). These methods are either data-free or only require a small calibration dataset and are generally quite easy to use. Having almost no hyperparameter tuning makes them usable via a single API call as a black-box method to quantize a pre-trained neural network in a computationally efficient manner.

Post-training quantization of LLMs is a challenging task due to presence of numerical outliers in weights and activations (Bondarenko et al., 2021; 2024; Kovaleva et al., 2021; Dettmers et al., 2022; Sun et al., 2024). Existing LLM PTQ methods can be broadly categorized into *weights-only* quantization and *weight-activation* quantization.

Weights-only quantization focuses on converting weights to low-bit values. For instance, GPTQ (Frantar et al., 2022) employs second-order information to iteratively round grouped weights and correct the quantization error in the remaining groups. SpQR (Dettmers et al., 2023), AWQ (Lin et al., 2023) and OWQ (Lee et al., 2024) emphasize the importance of (so-called "salient") weights that correspond to high-magnitude activations. Other recent W-only methods include (Jeon et al., 2023; Lee et al., 2023b; Luo et al., 2023; Chee et al., 2024).

Weight-activation quantization compresses both weights and activations. SmoothQuant (Xiao et al., 2023), `LLM.int8()` (Dettmers et al., 2022) and Outlier Suppression (Wei et al., 2022) achieve W8A8 quantization by managing activation outliers. `LLM.int8()` uses mixed-precision decomposition, while the other two employ channel-wise scaling. OmniQuant (Shao et al., 2023) modulates the extreme values of weights by optimizing the clipping threshold and shifts the challenge of quantization from activations to weights by employing the learnable equivalent transformation. Some of the other recent W&A PTQ methods are (Lee et al., 2023a; Liu et al., 2023a; Wei et al., 2023; Yuan et al., 2023; Tang et al., 2024; Yao et al., 2022; Lin et al., 2024).

Quantization-aware training methods Quantization-aware training (QAT) methods (Bhalgat et al., 2020; Esser et al., 2020; Gupta et al., 2015; Jacob et al., 2018; Krishnamoorthi, 2018) simulate quantization during training, allowing the model to find more optimal solutions compared to post-training quantization. However, better accuracy/perplexity comes at the cost of neural network training, i.e., longer training times, need for labeled data and hyperparameter search.

The excessive training cost and memory usage of traditional QAT methods make them unsuitable for quantizing modern LLMs. A few works that apply QAT to LLMs include LLM-QAT (Liu et al., 2023b) that combine QAT with data-free knowledge distillation, and EdgeQAT (Shen et al., 2024) that only considers tiny (sub 100M parameter) language models.

Low-rank adapters for fine-tuning Low-rank adaptation (LoRA) (Hu et al., 2021) is a parameter efficient fine-tuning (PEFT) method that reduces memory requirements. LoRA freezes the pretrained weight $\boldsymbol{W} = \boldsymbol{W_0}$, and only trains a small set of low-rank trainable parameters, often termed adapters. Given a linear projection $\mathbf{y} = \boldsymbol{W}\mathbf{x}$ with $\boldsymbol{W} \in \mathbb{R}^{m \times k}$, LoRA computes

$$\mathbf{y} = \boldsymbol{W}\mathbf{x} + \frac{\alpha}{r}\boldsymbol{A}\boldsymbol{B}\mathbf{x}, \tag{8}$$

where $\boldsymbol{A} \in \mathbb{R}^{m \times r}$, $\boldsymbol{B} \in \mathbb{R}^{r \times k}$, $r < \min\{m, k\}$ – rank, and α is a scalar that is constant in r. The benefits of LoRA are that it's much cheaper and often performs on par or better than full fine-tuning and also that the fine-tuned (floating-point) model can be deployed without extra cost as the low-rank matrices can be fused into the pretrained weights after fine-tuning ($\boldsymbol{W} := \boldsymbol{W_0} + \frac{\alpha}{r}\boldsymbol{A}\boldsymbol{B}$).

Naturally, there have been several works that explored the combination of LoRA and quantization. QLoRA (Dettmers et al., 2024) quantizes the pretrained weights to 4 bit using NF4 format and dequantizes them in the forward pass to further reduce fine-tuning memory footprint. QA-LoRA (Xu et al., 2023), uses INT4 quantization and introduces group-wise operators to enable quantization during inference stage. LoftQ (Li et al., 2023) proposed an iterative SVD-based procedure for initializing $\boldsymbol{A}$, $\boldsymbol{B}$ that yields faster fine-tuning convergence when used together with low-bit quantization. LQ-LoRA (Guo et al., 2023) further extends initialization technique from LoftQ to mixed precision and data aware cases. Other recent works include (Jeon et al., 2024; Zhang et al., 2024).

Finally, the closest work to ours is PEQA (Kim et al., 2024), that attempts to combine the benefits of inference-efficiency of QAT together with memory-efficiency of PEFT methods. However, their approach is different since they focus on a task-specific fine-tuning as opposed to being a general extended pre-trinaing method. In addition to that, PEQA has significantly less degrees of freedom compared to our method, leading to a subpar performance.

Motivation PTQ, while generally fast and simple, suffers from limited performance in low-bit scenarios. Although QAT methods still perform well in low-bit regimes, their high training costs and memory usage make them impractical for LLMs.

LoRA-based methods address memory issues for efficient fine-tuning, however, in most cases they don't consider efficient inference. The adapters $\boldsymbol{A}$, $\boldsymbol{B}$ are typically stored in higher precision format such as BF16 and at inference they dequantize the low-bit integer matrix $\boldsymbol{W}_{\mathbb{Z}}$ to the same data format, resulting in runtime overhead.

Simply quantizing adapters after training will lead to a different quantization grid compared to $\boldsymbol{W}$, and quantizing them specifically using the same quantization grid as $\boldsymbol{W}$ will lead to high error. QA-LoRA is the only work we are aware of that attempts to fuse auxiliary LoRA weights back

Method	Accuracy	Memory efficiency	Inference efficiency
PTQ	✗	✓	✓
(Full-model) QAT	✓	✗	✓
LoRA / PEFT	✓	✓	✗
LR-QAT (ours)	✓	✓	✓

Table 3: A comparison between existing approaches and the proposed method.

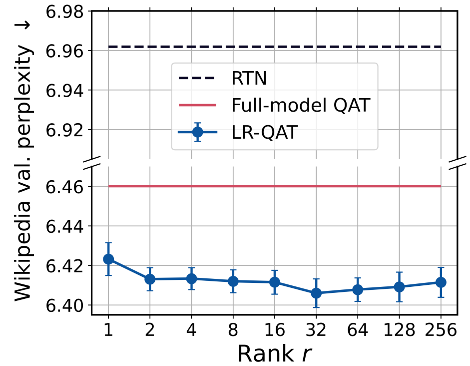

Figure 2: The performance of LR-QAT (φ = Q4.4) depending on the rank r of auxiliary matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ on LLaMA-2 7B with W4 per-channel quantization. We report mean and standard deviation over 5 runs with different random seeds.

into the frozen $\boldsymbol{W}_{\mathbb{Z}}$. However, their method is designed to only work with group-wise quantization with high number of groups (a small group size of 32). In addition to that, QA-LoRA and most of LoRA-based methods combine their proposed technique with the task-specific fine-tuning whereas we propose LR-QAT as an *extended pre-training* method.

We are inspired by LoRA-based methods to make QAT more memory and runtime efficient. In addition to that, our goal is to design a method that is *inference efficient*, *i.e.* where the low-rank adapters can be fused back into a low-bit integer matrix $\boldsymbol{W}_{\mathbb{Z}}$ without any loss of accuracy/perplexity, yielding PTQ level of inference efficiency. Contrary to QA-LoRA (Xu et al., 2023), we are not relaxing the quantization constraints – our method is applicable at any weight quantization granularity. Finally, we see our method as a general extended pre-training framework. The resulting model can afterwards still be used on any task. We summarize different trade-offs for the discussed techniques in Table 3.

B. Additional results

B.1. The impact of rank r

We investigate the effect of different values of rank r of the auxiliary matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ and present results in Figure 2. Increasing the rank from 1 to 32 leads to progressively

Table 4: The performance of LR-QAT applied to LLaMA-2 7B depending on the choice of downcasting operator $\varphi(\cdot)$, compute data type, and initialization method for low-rank auxiliary matrices. We report WikiText-2 test set perplexity, lower is better, and average zero-shot accuracy of 6 tasks, higher is better. Numbers marked in bold are the best results.

$\varphi(\cdot)$	dtype	$\boldsymbol{A}$, $\boldsymbol{B}$ init.	WikiText-2 ↓		Zero-shot acc. ↑	
			W4 pc	W3 pc	W4 pc	W3 pc
FP32	FP32	LoRA	5.69	6.21	69.28	66.62
FP16	FP32	LoRA	+0.00	**+0.01**	−0.13	−0.01
BF16	FP32	LoRA	**−0.01**	**+0.01**	+0.11	**+0.45**
Q4.4 / Q3.5	FP32	LoRA	**−0.01**	**+0.01**	**+0.16**	+0.31
Q4.4 / Q3.5	**BF16**	**LoRA**	**−0.01**	**+0.01**	+0.15	+0.31
INT-4 / INT-3	FP32	LoRA	+0.02	+20.5	−0.04	−22.8
INT-4 / INT-3	FP32	LoftQ ($T = 1$)	+0.28	+0.18	−0.67	+0.26
INT-4 / INT-3	FP32	LoftQ ($T = 64$)	+0.40	+1.37	−1.40	−2.01

slightly better performance, excluding one outlier. The fact that using $r > 32$ doesn't lead to further improvement in perplexity is likely because of the limited number of training steps we used for this experiment (10^3), and more steps needed for the procedure to fully converge. Interestingly, a rank r as small as 1 already performs really well. We hypothesize that this is the case because of the following. Even though $\mathrm{rank}\,(\boldsymbol{AB}) = 1$, due to applying a low-rank approximation inside the rounding and clipping operators in (6), this can overall leads to a high-rank perturbation to the original weights $\boldsymbol{\Phi}_0$ (in the integer domain). Going forward, we use $r = 32$ in all our experiments[1].

B.2. The impact of downcasting operator $\varphi(\cdot)$ and initialization

We study the effect of several choices of the downcasting operator and summarize results in Table 4. We can see that by going from FP32 to BF16, and finally to an 8-bit fixed-point representation of $\boldsymbol{\Phi}_0$, aside from memory savings we also maintain the same WikiText-2 perplexity and even slightly improve in terms of zero-shot accuracy. The latter is likely due to a slight regularization effect caused by the fact that we discard some of the information in the fractional part in $\boldsymbol{W_0}/\mathbf{s_0}$, some of which might be noise. One step further, however, while $\varphi =$ INT-b still leads to a good model performance in the case of 4-bit weight quantization, it completely breaks for W3.

So far, we initialized matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ following the procedure proposed in LoRA (Hu et al., 2021) where $\boldsymbol{B}$ is initialized to zero, and $\boldsymbol{A}$ is initialized randomly as in (He et al., 2015). We refer to this initialization scheme as 'LoRA'. We hypothesize that a poor performance of $\varphi =$ INT3 can be explained by the fact that we lose all the information in the fractional part of $\boldsymbol{W_0}/\mathbf{s_0}$ and that without that information it is difficult for low rank approximation to learn. To address this, we adapt and experiment with a variant of SVD-based initialization proposed in LoftQ (Li et al., 2023). We see that using LoftQ initialization with $T = 1$ step recovers almost all the predictive performance compared to a fixed-point representation. Increasing number of LoftQ steps, or applying it to a 4-bit case did not help, however.

Finally, when using the fixed point representation for $\boldsymbol{\Phi}_0$, we still maintain the same model performance by switching the compute data type[2] from FP32 to BF16, where the latter is what is commonly used for LLMs.

B.3. Weight-activation quantization

In weight-activation quantization, defaults are INT4 per-channel weight and per-token activation quantization (Dettmers et al., 2022). Following OmniQuant (Shao et al., 2023) we quantize all inputs to matmuls with exception of the softmax output and additionally quantize the KV-cache as in LLM-QAT (Liu et al., 2023b). For weight-activation quantization, we compare to RTN, SmoothQuant (Xiao et al., 2023) and LLM-QAT (Liu et al., 2023b). Following (Liu et al., 2023b), we compare to them in several different settings, where the weights, activations and KV cache values are quantized to different levels (denoted as W-A-KV). To compare to the above literature, we apply LR-QAT to LLaMa-1 7B (Touvron et al., 2023a).

We present our results for weight-activation quantization applied to LLaMA-1 7B in Table 5. LR-QAT consistently outperforms the PTQ baselines and is on par or better compared to LLM-QAT. This demonstrates that the proposed method is readily applicable not only to weight-only quantization but also weight-activation quantization, a setting that allows for a very efficient inference using fixed-point arithmetic. Finally, our method can still be combined with

[1] This amounts to only 1.2% of the total number of parameters for 7B LLaMA model.

[2] A data type used for activations, gradients, and frozen parameters.

Table 5: **Weight and activation quantization results for LLaMA-1 7B**. We report WikiText-2 test set perplexity (lower is better) and zero-shot accuracy of 6 tasks (higher is better). Numbers marked in bold are SOTA. §Uses asymmetric weight quantization. *Uses a maximum sequence length of 1024 for evaluation.

# Bits (W-A-KV)	Method	WikiText-2 ↓	Zero-shot accuracy ↑						
			BoolQ	PIQA	Winogrande	ARC-e	ARC-c	HellaSwag	**Avg.**
FP16		5.68	75.05	79.16	70.01	72.85	44.80	76.21	69.68
4-8-8	RTN	6.88	71.35	76.66	66.46	66.84	41.55	72.10	65.83
	SmoothQuant	13.7*	71.00	76.00	66.00	67.40	42.80	67.80	65.17
	LLM-QAT	11.2*	74.60	77.50	67.70	70.20	**45.60**	73.50	68.18
	PEQA (our impl.)	5.89	**74.86**	78.24	70.01	70.12	42.83	**75.14**	68.53
	LR-QAT (ours)	**5.85**	73.76	**78.51**	**71.19**	**71.09**	41.81	75.10	**68.58**
4-8-4	RTN	7.66	68.81	75.46	62.12	62.46	39.51	68.33	62.78
	SmoothQuant	163.6*	54.70	55.40	51.50	43.90	27.70	38.90	45.35
	LLM-QAT	11.6*	69.50	75.40	64.60	66.00	**43.80**	69.20	64.75
	PEQA (our impl.)	6.15	72.97	77.80	**67.72**	67.13	40.27	**73.35**	66.54
	LR-QAT (ours)	**6.07**	**73.64**	**77.91**	67.56	**69.28**	41.30	73.25	**67.16**
4-4-4	RTN	17.75	50.49	64.25	52.41	48.27	30.12	52.04	49.60
	SmoothQuant	25.25	49.10	49.80	48.00	30.40	25.80	27.40	38.42
	LLM-QAT	-	61.30	51.50	51.90	27.90	23.90	31.10	41.27
	LLM-QAT + SQ	-	62.40	55.90	50.60	35.50	26.40	47.80	46.43
	OS+	-	60.21	62.73	52.96	39.98	30.29	44.39	48.43
	OmniQuant§	11.26	63.51	66.15	53.43	45.20	31.14	56.44	52.65
	PEQA (our impl.)	8.60	65.69	**72.31**	**59.83**	56.52	34.22	61.79	58.39
	LR-QAT (ours)	**8.47**	**67.16**	71.76	59.59	**58.42**	**34.73**	**62.34**	**59.00**

most of the related PTQ methods including OmniQuant that shift the difficulty of activation quantization to weight quantization, and lead to even better results.

B.4. Extended results

In this section, we provide additional detailed results.

Table 6: **A comparison between min-max and the best range setting used for round-to-nearest (RTN) initialization for LLaMA and Mistral models**. We report WikiText-2 test set perplexity (lower is better) and average zero-shot accuracy (higher is better). Numbers marked in bold are the best results.

Model	# Bits	RTN init.	WikiText-2 ↓	Zero-shot accuracy ↑
LLaMA-1 7B	FP16		5.68	69.68
	W4 pc	min-max	6.85	66.23
		best (L^4)	**6.33**	**68.51**
	W3 pc	min-max	2.4e4	36.02
		best ($L^{3.5}$)	**12.88**	**54.66**
	W4 g128	min-max	6.08	**68.96**
		best (L^5)	**6.05**	68.93
	W3 g128	min-max	8.10	62.69
		best (L^5)	**7.95**	**63.50**
LLaMA-2 7B	FP16		5.47	70.47
	W4 pc	min-max	7.14	66.41
		best ($L^{3.5}$)	**6.14**	**68.88**
	W3 pc	min-max	1.9e4	35.71
		best ($L^{3.5}$)	**26.73**	**43.87**
	W4 g128	**best = min-max**	**5.78**	**69.75**
	W3 g128	min-max	8.22	**64.07**
		best (L^4)	**7.61**	63.20
LLaMA-2 13B	FP16		4.88	73.18
	W4 pc	min-max	5.40	**72.19**
		best ($L^{3.5}$)	**5.21**	71.73
	W3 pc	min-max	2.3e3	37.85
		best (L^5)	**8.71**	**55.01**
	W4 g128	**best = min-max**	**5.04**	**72.94**
	W3 g128	min-max	**6.14**	66.81
		best (L^5)	6.20	**67.60**
LLaMA-3 8B	FP16		6.14	74.22
	W4 pc	min-max	10.53	67.44
		best ($L^{3.5}$)	**7.53**	**72.19**
	W3 pc	min-max	1.6e5	35.78
		best ($L^{3.5}$)	**34.10**	**47.46**
	W4 g128	min-max	6.99	**72.95**
		best (L^4)	**6.96**	72.30
	W3 g128	min-max	29.38	54.54
		best (L^5)	**15.11**	**57.74**
Mistral 7B	FP16		5.25	75.69
	W4 pc	min-max	6.33	71.84
		best (L^4)	**5.91**	**73.44**
	W3 pc	min-max	3.2e3	36.78
		best (L^4)	**9.49**	**64.58**
	W4 g128	min-max	5.51	74.98
		best (L^5)	**5.49**	**75.07**
	W3 g128	min-max	7.22	68.35
		best (L^5)	**6.77**	**69.35**

Table 7: **LM-eval weight-only quantization results for LLaMA and Mistral models**. We report zero-shot accuracy of 6 tasks (higher is better).

Model	# Bits	Method	BoolQ	PIQA	Winogrande	ARC-e	ARC-c	HellaSwag	Avg.
LLaMA-1 7B	FP16		75.05	79.16	70.01	72.85	44.80	76.21	69.68
	W4 pc	RTN	73.18	78.78	69.14	71.38	44.37	74.22	68.51
		GPTQ	67.70	76.00	66.70	66.90	43.00	69.40	64.95
		LLM-QAT	75.50	78.30	69.00	70.00	45.00	74.00	68.63
		PEQA (our impl.)	74.71	78.29	70.09	70.33	42.24	75.27	68.49
		LR-QAT (ours)	**74.13**	**78.29**	**70.01**	**71.21**	**42.41**	**75.16**	**68.54**
	W3 pc	RTN	58.93	70.40	55.72	55.01	32.17	55.75	54.66
		PEQA (our impl.)	72.69	77.15	65.90	68.27	38.91	71.60	65.75
		LR-QAT (ours)	**73.24**	**78.18**	**67.40**	**67.47**	**40.53**	**72.77**	**66.60**
	W4 g128	RTN	74.77	78.51	70.64	71.30	43.60	74.74	68.93
		PEQA (our impl.)	75.75	79.17	70.17	70.75	43.60	75.71	69.19
		LR-QAT (ours)	**75.29**	**78.62**	**69.61**	**71.59**	**44.11**	**75.67**	**69.15**
	W3 g128	RTN	69.48	76.33	64.40	64.44	38.65	67.67	63.50
		PEQA (our impl.)	71.65	78.24	68.51	68.18	40.10	73.30	66.66
		LR-QAT (ours)	**72.84**	**78.02**	**67.40**	**68.52**	**41.04**	**73.04**	**66.81**
LLaMA-2 7B	FP16		77.74	79.11	69.14	74.58	46.25	75.98	70.47
	W4 pc	RTN	76.36	78.07	68.19	71.21	44.80	74.65	68.88
		PEQA (our impl.)	77.49	78.24	69.61	70.96	43.52	75.54	69.23
		LR-QAT (ours)	**77.43**	**78.45**	**69.69**	**73.15**	**45.48**	**75.51**	**69.95**
	W3 pc	RTN	46.27	60.28	54.85	38.05	23.29	40.47	43.87
		PEQA (our impl.)	71.62	76.82	66.14	65.66	39.76	72.63	65.44
		LR-QAT (ours)	**74.43**	**77.15**	**68.03**	**69.95**	**43.09**	**73.29**	**67.66**
	W4 g128	RTN	76.76	78.18	69.77	72.60	45.73	75.43	69.75
		PEQA (our impl.)	76.88	78.89	69.85	72.18	44.11	75.95	69.64
		LR-QAT (ours)	**76.73**	**78.62**	**70.48**	**72.85**	**44.97**	**75.62**	**69.88**
	W3 g128	RTN	66.42	75.57	65.19	64.90	38.14	68.96	63.20
		PEQA (our impl.)	75.38	77.97	68.59	70.62	42.32	73.74	68.10
		LR-QAT (ours)	**73.30**	**78.07**	**67.72**	**71.46**	**43.77**	**73.53**	**67.98**
LLaMA-2 13B	FP16		80.55	80.52	72.22	77.44	48.98	79.38	73.18
	W4 pc	RTN	79.30	79.71	70.01	75.51	48.89	76.96	71.73
		PEQA (our impl.)	78.99	80.14	71.27	76.43	48.98	79.24	72.51
		LR-QAT (ours)	**80.15**	**80.09**	**72.06**	**77.65**	**49.91**	**79.28**	**73.19**
	W3 pc	RTN	55.05	71.06	54.22	56.19	32.25	61.27	55.01
		PEQA (our impl.)	74.28	78.67	69.06	74.87	45.99	76.00	69.81
		LR-QAT (ours)	**78.62**	**79.49**	**72.61**	**73.99**	**45.56**	**77.05**	**71.22**
	W4 g128	RTN	81.10	79.82	72.38	76.73	49.06	78.52	72.94
		PEQA (our impl.)	80.28	80.63	71.74	76.14	48.38	79.62	72.80
		LR-QAT (ours)	**80.73**	**80.30**	**71.74**	**76.14**	**49.06**	**79.51**	**72.91**
	W3 g128	RTN	74.65	76.93	69.14	70.16	42.66	72.06	67.60
		PEQA (our impl.)	78.56	78.73	69.85	73.61	44.28	76.69	70.29
		LR-QAT (ours)	**79.79**	**79.60**	**70.64**	**74.24**	**46.76**	**78.00**	**71.51**
LLaMA-3 8B	FP16		81.44	80.79	72.85	77.74	53.33	79.16	74.22
	W4 pc	RTN	79.02	78.56	72.85	75.97	49.32	77.44	72.19
		PEQA (our impl.)	79.57	78.67	72.93	77.19	51.11	77.25	72.79
		LR-QAT (ours)	**81.62**	**79.98**	**72.85**	**78.32**	**52.05**	**78.19**	**73.84**
	W3 pc	RTN	58.65	61.75	56.04	39.60	23.81	44.91	47.46
		PEQA (our impl.)	63.18	64.74	57.62	43.39	26.88	50.48	51.05
		LR-QAT (ours)	**77.46**	**78.51**	**69.85**	**74.83**	**47.35**	**74.73**	**70.46**
	W4 g128	RTN	79.48	79.27	73.56	75.08	48.81	77.61	72.30
		PEQA (our impl.)	80.98	80.14	72.61	76.18	49.57	78.45	72.99
		LR-QAT (ours)	**80.40**	**80.90**	**73.48**	**77.44**	**51.11**	**78.60**	**73.66**
	W3 g128	RTN	65.47	68.39	65.19	54.00	33.45	59.96	57.74
		PEQA (our impl.)	72.26	76.06	67.80	69.02	46.08	71.89	67.19
		LR-QAT (ours)	**72.97**	**79.38**	**71.67**	**74.37**	**49.06**	**75.44**	**70.48**
Mistral 7B	FP16		83.58	82.10	73.88	79.59	53.92	81.07	75.69
	W4 pc	RTN	81.22	80.63	72.53	76.77	50.09	79.41	73.44
		PEQA (our impl.)	81.80	81.12	72.61	77.23	50.17	79.43	73.73
		LR-QAT (ours)	**81.99**	**81.28**	**73.56**	**78.20**	**51.02**	**80.57**	**74.44**
	W3 pc	RTN	68.13	77.64	63.93	63.93	41.13	72.73	64.58
		PEQA (our impl.)	80.03	80.09	69.93	72.90	45.82	77.32	71.02
		LR-QAT (ours)	**81.62**	**80.09**	**70.96**	**74.75**	**46.08**	**77.71**	**71.87**
	W4 g128	RTN	84.16	81.77	74.43	77.95	51.71	80.42	75.07
		PEQA (our impl.)	80.89	81.72	73.80	75.42	48.46	79.76	73.34
		LR-QAT (ours)	**83.55**	**81.61**	**74.51**	**78.28**	**52.90**	**80.84**	**75.28**
	W3 g128	RTN	78.44	79.60	69.14	71.17	43.00	74.75	69.35
		PEQA (our impl.)	81.99	81.18	69.61	74.92	47.18	78.37	72.21
		LR-QAT (ours)	**81.71**	**80.90**	**70.48**	**75.08**	**47.78**	**78.50**	**72.41**

C. Experimental details

In this section, we list the details related to hyperparameters and other settings used in our experiments. If not stated otherwise, the standard hyperparameters that we use are the one shown in Table 8.

Hyperparameter	Value / Search space
Optimizer	AdamW
Learning rate for $\boldsymbol{A}$, $\boldsymbol{B}$	$\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$
Learning rate for quantization scale ($\mathbf{s}$)	$\{0^{*}, 10^{-5}\}$
Learning rate schedule for $\boldsymbol{A}$, $\boldsymbol{B}$	`linear` (with warmup)
Learning rate schedule for quantization scale ($\mathbf{s}$)	`linear` (with warmup)
Weight decay for $\boldsymbol{A}$, $\boldsymbol{B}$	0
Weight decay for quantization scale ($\mathbf{s}$)	0
Adam (β_1, β_2)	$(0.9, 0.95)$
Training steps	10^4
Warmup steps	10% of Training steps
Batch size	32
Maximum sequence length (during training)	1024
L^2-norm gradient clipping (maximum norm)	1.0
α in (6)	1.0

Table 8: Common hyperparameters used for experiments. *Is equivalent to freezing the quantization scale obtained after initial range estimation ($\mathbf{s} = \mathbf{s_0}$).

Quantization We experiment with both weight-only and weight-activation quantization. The default settings are INT4 / INT3 per-channel (denoted 'pc') and group-wise weight quantization with a group size of 128 (denoted 'g128'). We always use symmetric uniform affine quantization. We quantize all linear layers, except the classification head. RMSNorm and embedding layers are always kept at full precision. In weight-activation quantization, defaults are INT4 per-channel weight and per-token activation quantization (Dettmers et al., 2022). Following OmniQuant (Shao et al., 2023) we quantize all inputs to matmuls with exception of the softmax output and additionally quantize the KV-cache as in LLM-QAT (Liu et al., 2023b).

Libraries We implement our method in PyTorch (Paszke et al., 2019) and use training and evaluation pipelines from HuggingFace libraries (Gugger et al., 2022; Lhoest et al., 2021; Wolf et al., 2020). For zero-shot evaluation, we use the LM Evaluation Harness framework (Gao et al., 2021).

Datasets and training To optimize the learnable parameters, we use AdamW optimizer (Loshchilov and Hutter, 2017) with weight decay set to zero, $(\beta_1, \beta_2) = (0.9, 0.95)$ and linear learning rate warm up over the first 10% steps, following by a linear decay to zero by the end of training. We use a separate maximum learning rate for quantization scales and for low-rank adapters, which are tuned depending on the experiment.

We apply our methods to all linear layers in the attention blocks (both in self-attention and in the feed-forward network). We only train low-rank auxiliary matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and the quantization parameters $\mathbf{s}$. Specifically, we freeze embedding layers, the final classification heads and RMSNorm parameters.

We train on a small subset of SlimPajama (Soboleva et al., 2023), which is a close open-source replica of the dataset used for pre-training LLaMA models. We select hyperparameters based on the perplexity of a small subset of Wikipedia validation set[3] (512 sequences). For weight-only and weight-activation quantization results, including the comparison with full-model QAT in Section 3.2, we train for 10^4 steps. For ablation studies in Sections B.1 and B.2, we use shorter training of 10^3 steps. Since the full-model QAT experiment requires more than 80GB of GPU memory, we used CPU optimizer state offloading to be able to run the experiment on an NVidia A100 GPU with 80GB VRAM. The rest of the hyperparameters and their search spaces are listed in Table 8.

PTQ initialization We compare with vanilla round-to-nearest quantization (RTN), where we explore several choices of range setting and report the best one based on Wikipedia validation set perplexity, and also use that as initialization for our method. Specifically, we experimented with min-max range estimator and with L^p-norm range estimator with the following values for p: $\{2.0, 2.4, 3.0, 3.5, 4.0, 5.0\}$.

Computational Resources Each experiment for which we reported results, was executed on a single NVidia A100 GPU equipped with 80GB of VRAM. LLaMA-2 7B and 13B experiments needed respectively around 3 and 5 days for 10k training steps experiments. Mistral 7B and LLaMA-3 8B needed around 1.6 days for 5k training steps experiments. For obtaining all the results in the paper, including the ablations, we needed 107 GPU days (A100). Including preliminary experiments that did not make it in the final paper and hyperparameter turning we estimate the total compute costs of this research to approximately 500 GPU days.

[3] Specifically, we use the English subset of Wiki40b, `https://huggingface.co/datasets/wiki40b`, that contains cleaned-up text of English Wikipedia and training/validation splits.