SiLQ: Simple Large Language Model Quantization-Aware Training

Steven K. Esser[1], Jeffrey L. McKinstry[1], Deepika Bablani[1], Rathinakumar Appuswamy[1], Dharmendra S. Modha[1]

[1]IBM Research, San Jose, CA, USA

Correspondence: sesser@us.ibm.com

Abstract

Large language models can be quantized to reduce inference time latency, model size, and energy consumption, thereby delivering a better user experience at lower cost. A challenge exists to deliver quantized models with minimal loss of accuracy in reasonable time, and in particular to do so without requiring mechanisms incompatible with specialized inference accelerators. Here, we demonstrate a simple, end-to-end quantization-aware training approach that, with an increase in total model training budget of less than 0.1%, outperforms the leading published quantization methods by large margins on several modern benchmarks, with both base and instruct model variants. The approach easily generalizes across different model architectures, can be applied to activations, cache, and weights, and requires the introduction of no additional operations to the model other than the quantization itself.

1 Introduction

Large language models (LLMs) have achieved remarkable capabilities across a wide range of AI tasks. However, there are two major challenges emerging in wide-scale deployment of LLMs: energy consumption and response latency (Appuswamy et al., 2024). It is estimated that inference accounts for 80% of the total energy cost of LLM solutions (World Economic Forum, 2024). In addition, current and emerging applications such as interactive dialog and agentic workflows require very low latencies. Both of these concerns are addressed by low precision neural inference processors, such as NorthPole (Modha et al., 2023). Lower precision compute reduces power consumption directly, and by reducing area, allows model memory to be placed next to compute for further energy savings and lower latency (Modha et al., 2023).

While nearly all LLMs are trained today using 16-bit precision, it is possible to quantize such models to run on low precision inference accelerators. To this end, a variety of techniques have emerged that seek to minimize quantization related accuracy loss, while providing fast time-to-solution, judged relative to total development time, simple implementation, and a quantized model that is fully compatible with the target deployment platform. Efforts have focused on post-training quantization (PTQ), where quantization is tuned using a small amount of calibration data, or quantization-aware training (QAT), where a model is fine-tuned with differentiable quantization operators. PTQ is argued to be preferable due to its low dataset and compute requirements, and recent work has shown it to outperform QAT based approaches (Liu et al., 2024).

Here, we provide a counterpoint to this perspective. We introduce a simple approach to QAT that requires increasing the total training budget by less 0.1%, measured in training tokens, and that can make use of publicly available datasets or the model's original fine-tuning data. The approach achieves accuracy several percentage points superior to the best published alternative quantization methods (Table 1, with details in Tables 5, 6, and 7 in the appendix) on zero-shot Common Sense Reasoning tasks (CSR)[1] and the Huggingface Open LLM leader-board versions 1 and 2 (OLLMv1 and OLLMv2)[2].

Our approach, Simple Large Language Model Quantization-Aware Training (SiLQ), is to: 1) add quantization to the model to match the target deployment configuration, using the straight through estimator (Bengio et al., 2013) for training time gradients, 2) set quantizer step size values initially through calibration and then refine further using LSQ (Esser et al., 2019), and then 3) train end-to-end with a standard workflow, allowing one to employ existing code frameworks, using knowl-

[1]As in (Liu et al., 2024), see references therein

[2]As in (Lee et al., 2024), see references therein

Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 411–422

Table 1: SiLQ results in more accurate base and instruction-tuned LLMs than leading PTQ techniques across common sense reasoning and Open LLM benchmarks. The Bits column indicates bits to quantize activations, with "s" or "d" denoting static or dynamic quantization, then KV cache, then weights. SmoothQuant and SpinQuant results computed by us, using code published by each paper's authors.

Model	Bits A-C-W	Method	CSR	OLLM v1	OLLM v2
Llama-3-8B	16-16-16	Baseline	67.09	62.65	13.70
	8d-8-4	SmoothQuant*	58.73	45.15	6.65
		SpinQuant	63.97	57.99	12.28
		SiLQ	67.20	61.14	12.66
Llama-3.1-Tulu-3.1-8B	16-16-16	Baseline	69.91	69.56	26.45
	8d-8-4	SmoothQuant*	64.20	58.12	20.49
		SpinQuant	67.47	66.91	24.03
		SiLQ	69.59	69.79	27.10
Granite-3.1-8B-Instruct	16-16-16	Baseline	68.46	72.11	29.91
	8d-8-4	SmoothQuant*	62.68	61.66	20.08
		SpinQuant	65.96	65.52	21.35
		SiLQ	68.03	71.48	29.14
	8s-8-4	SmoothQuant*	49.06	35.24	8.93
		SiLQ	67.41	70.86	29.03
	8d-4-4	SmoothQuant*	56.48	39.78	7.15
		SpinQuant	63.12	56.67	14.47
		SiLQ	67.92	70.93	29.13

*head not quantized

edge distillation (Hinton et al., 2015), and either the model's original training dataset or a high quality public dataset. We further introduce a weight calibration technique based on an approximation of mean squared error. We train on up to a few billion tokens, a small investment compared to the trillions of tokens used to train modern LLMs.

A benefit of SiLQ is solution scalability. To the limit of what is possible through deep learning, one can train longer to improve accuracy, which we demonstrate empirically (Figure 1). This allows one to balance up-front training costs with deployment time performance as needed. In addition, QAT methods in principle offer better generalizability. One simply adds precision constraints specific to a given accelerator, and lets the optimizer find the best solution tailored to those constraints. On the other hand, to reach acceptable accuracy, PTQ methods require identifying model-specific impediments to quantization, such as outliers appearing in particular layers, and then devising custom solutions for those issues (Frantar et al., 2022; Xiao et al., 2023; Liu et al., 2024; Ashkboos et al., 2024).

Surprisingly few demonstrations have been published using QAT for typical LLM deployment scenarios. Existing work either does not quantize all tensors necessary for full acceleration (Liu et al., 2025), targets extremely low precisions at the cost of accuracy degradation that would likely be unacceptable for most users (Du et al., 2024), or introduces time-consuming complexities such as data self-generation (Liu et al., 2023). Further, existing publications focus on quantizing the base version of various models, rather than the higher accuracy instruction-tuned version typically used in deployment.

We demonstrate our approach on models with an 8-bit activation, 4- or 8-bit cache, and 4-bit weight configuration. Remarkably, even when applied to instruction-tuned models that were originally created using a multistage process of pre-training, supervised fine-tuning, direct preference optimization, proximal policy optimization, and model averaging (Lambert et al., 2024), our single stage end-to-end approach preserves accuracy at nearly the same level as the original model, outperforming all leading hardware-friendly LLM quantization methods.

2 Related work

A typical LLM consists of an initial embedding layer and final linear layer, the head, sandwiching a series of alternating attention and multi-layer perceptron blocks, each in turn comprising normalization, element-wise, matrix multiplication, and linear operations. Input is provided as a series of integer valued tokens, and the model stores intermediate responses to earlier tokens of an input sequence in multiple caches. The model is parameterized by various weights and biases learned during training. The linear and matrix multiplication layers comprise the majority of compute costs, and the

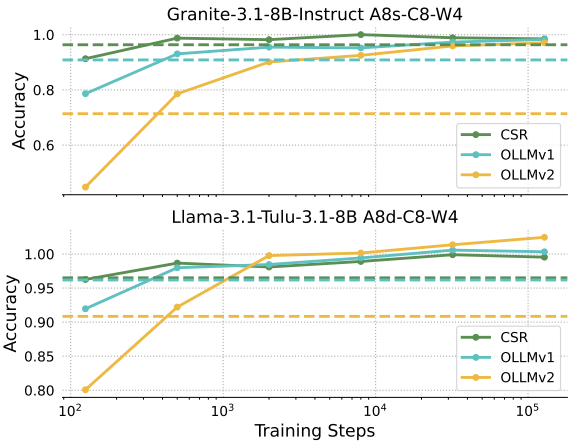

Figure 1: Accuracy improves with longer QAT. The y-axis represents accuracy relative to the original fp16 model. Horizontal dashed lines show PTQ method SpinQuant accuracy. Accuracy on the harder OLLMv1 and, in particular, OLLMv2 benchmarks improves the most with longer QAT, significantly outperforming PTQ.

cache and weights comprise the majority of memory costs. Both costs can be reduced by quantizing the cache and weights for storage, and all other inputs to linear and matrix multiplication layers, commonly referred to as activations, for compute.

In order to ground our work, we choose an inference accelerator, NorthPole (Modha et al., 2023), and ensure that our quantization scheme is fully compatible with this deployment platform. NorthPole employs a vector-matrix multiplication unit, used to compute linear and matrix multiplication layers, supporting any combination of 2-, 4-, 8-, or 16-bit integer activations as operands, can store the cache and weights at 2-, 4-, or 8-bit, and uses fp16 precision for other operations. We compare our approach to the three leading alternatives also compatible with these constraints, SmoothQuant (Xiao et al., 2023), SpinQuant (Liu et al., 2024), and LLM-QAT (Liu et al., 2023), but do not compare to approaches that would be unable to deploy to this platform.

SmoothQuant is a PTQ method that can effectively quantize LLMs with 8-bit weights, activations, and cache with no loss in accuracy. It increases the range of the weights in order to decrease the range of the activations, thereby reducing activation outliers, improving quantization accuracy. Unfortunately, it performs poorly with 4-bit weights (Table 1) (Liu et al., 2024).

SpinQuant is a state-of-the-art PTQ method that learns rotation matrices to reduce outliers, merges these rotations into the weights of the linear layers, then applies the GPTQ algorithm to quantize model weights. SpinQuant is shown to produce models with higher accuracy than methods with random weight rotations (Ashkboos et al., 2024). Surprisingly, it was also shown to be more accurate on a common-sense reasoning benchmark than the leading LLM QAT method discussed next.

LLM-QAT demonstrates superior results to the SmoothQuant PTQ method, and is compatible with our chosen constraints. LLM-QAT uses approximately 100,000 self-generated training samples to finetune LLMs. Data self generation removes dependency on external datasets, but requires a non trivial amount of compute time.

A number of other quantization methods exist that do not deliver a solution compatible with our chosen constraints, that is, that do not quantize all required tensors, or that do not reach acceptable accuracy in reasonable time, and so are not discussed beyond noting here. Weight-only PTQ methods like GPTQ (Frantar et al., 2022) are commonly employed to reduce model size, and at 4-bit precision come within 1 to 2% of the accuracy of full precision models (Lee et al., 2024). EfficientQAT focuses only on making QAT efficient for large models, and uses weight-only quantization at 2- and 3-bit precision (Chen et al., 2024b). PrefixQuant shows a greater than two percent improvement in accuracy by adding QAT after PTQ, however it adds unquantized values in the cache which is not hardware friendly (Chen et al., 2024a). QA-LoRA uses QAT but uses weight-only quantization (Xu et al., 2023). ParetoQ also applies QAT to weight-only quantization (Liu et al., 2025). BitDistiller is focused on 2 and 3 bit weight models, producing models with unacceptably low accuracy (Du et al., 2024). Finally, BitNet a4.8 uses 4-bit activations and 1-bit weights, but changes activation functions and retrains from scratch, impractical for quantizing existing LLMs (Wang et al., 2024).

3 Methods

3.1 Simple Large Language Model Quantization-Aware Training

Our approach is to apply QAT to train LLMs by following three simple practices: 1) add quantization to the model with the straight through estimator, 2) set step sizes through calibration, then refine using LSQ, and 3) train end-to-end using knowledge distillation. Details for each of these are provided below, including the specific configuration

we demonstrate here.

Add quantization to the model with the straight through estimator. Quantization can be added to any combination of activation, cache, and weight tensors, as dictated by the target deployment platform. Here, we quantize all three.

For training, we employ scaled quantization, as is common in QAT, to preserve data ranges, thereby making it easier to adapt existing training recipes. For symmetric quantization, which we employ here, this consists of quantizing (sub)tensor $\mathbf{x}$ with step size s according to

$$\hat{\mathbf{x}} = \lfloor \min(\max(\mathbf{x}/s, b_l), b_u) \rceil s, \qquad (1)$$

where $\lfloor . \rceil$ is the round to nearest operator, and b_l and b_u are the lower and upper bound (smallest and largest values) for the corresponding integer at the given precision. We pass the gradient through the round operation according to the straight through estimator (Bengio et al., 2013). For inference, weights are scaled to integers by dividing by their step size prior to deployment. Activations are similarly scaled during operation immediately prior to a linear or matrix multiplication layer, with the result then scaled back by multiplying by both associated scales. For our demonstration, we employ a scale per tensor for activations, and a scale per output channel for weights.

Set step sizes through calibration, then refine using LSQ. We use percentile initialization for activation step sizes. For our demonstration, we calibrate with 5 batches of 128 samples from the training set, setting the step size to the value at the 99.91, 99.99, and 99.995 percentile for 4-, 8-, and 16-bit activations, respectively.

We introduce a novel approach to set the weight step size to minimize a convex approximation of mean squared error. For a given set of weights $\mathbf{w}$, with precision p, we approximate the magnitude of the bound of the quantization range as $b = (2^{p-1} - 0.5)$. We assume a weight below this bound, that is $|w| < sb$, is equally likely to fall anywhere in its given quantization bin, treating its quantization error as a uniform random number bounded by $[-\frac{s}{2}, \frac{s}{2}]$, which has an expected squared value of $\frac{s^2}{12}$. This lets us approximate mean squared error as

$$\hat{\epsilon} = \Sigma_i \max(\tfrac{s^2}{12}, H(|w_i| - sb)(|w_i| - sb)^2) \qquad (2)$$

where H is the Heaviside step function. This function is convex with respect to s and so can easily be solved for the step size that approximately minimizes mean squared quantization error.

Following calibration, we employ the LSQ algorithm to adjust all quantization step sizes during training. We found that boosting the learning rate for the activation quantization step sizes by a factor of 50 was beneficial.

Train end-to-end using knowledge distillation. We employ standard end-to-end training using the original model training dataset when it is available, and high quality publicly available data when it is not. Training labels are provided using knowledge distillation, employing the original unquantized model as the teacher. Mixing a loss derived using teacher provided training labels with a loss using next-token-prediction may be beneficial in some instances, but we, along with others (Liu et al., 2023), have found this not to be necessary.

We test our method on four 7- or 8-billion parameter models available from Huggingface: meta-llama/Llama-2-7b, meta-llama/Meta-Llama-3-8B, allenai/Llama-3.1-Tulu-3.1-8B, and ibm-granite/granite-3.1-8b-instruct. The Tulu-3 instruction-tuned variant of Llama-3.1-8B was chosen because all instruction tuning datasets are publicly available. We train base models using the publicly available DCLM dataset (Li et al., 2024), and train instruct models using a mixture of the same Supervised Fine-Tuning (SFT) dataset used to finetune the original model (75%) and the DCLM dataset (25%). Further training details related to datasets and hyper-parameters are provided in Appendix B.

3.2 Precisions tested

For our demonstration, we use standard round-to-nearest, uniform, symmetric quantization, with 8-bit activations, 4- or 8-bit KV-Cache, and 4-bit weights, denoted A8-C8-W4 and A8-C4-W4. We use either token-wise dynamic quantization, or tensor-wise static quantization for all activations, indicated with 'd' and 's', respectively, e.g. A8d-C8-W4. Figure 2 shows the precisions used in the attention and multi-layer perceptron blocks used in this study. The input activations and weights of the final output linear layer are quantized to 8-bits, while the embedding layer remains fp16. To better match existing LLM quantization papers that do not quantize the matrix multiplication operations, we use INT16 precision for the two non-cache inputs to these layers, the query tensor and the softmax output tensor. Further, as the softmax output tensor

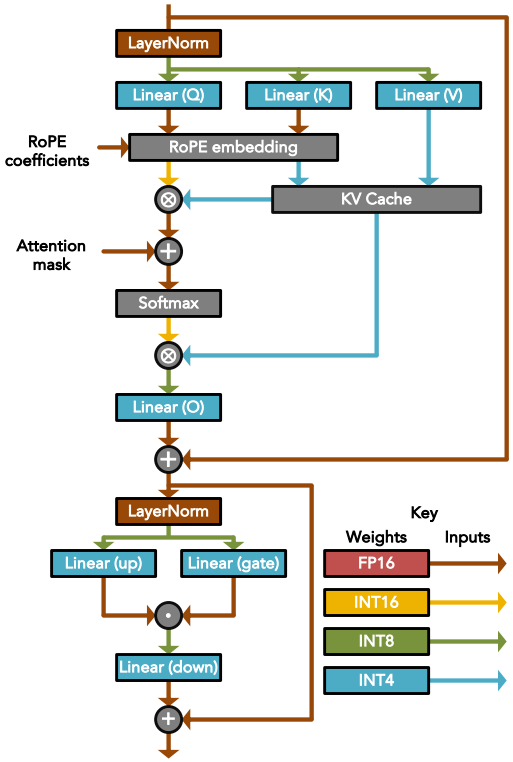

Figure 2: Transformer block with the A8-C4-W4 precision configuration tested here (A8-C8-W4 also tested) is fully compatible with the NorthPole processor.

is fully encapsulated by the flash attention CUDA kernel, which does not natively support QAT, we do not quantize this tensor during training. This allows us to take advantage of the significant memory and throughput advantages of flash attention (Dao et al., 2022), and empirically has no impact on task accuracy.

3.3 Model Evaluation

All evaluations are performed with the latest version of the lm-evaluation-harness (Gao et al., 2023). For reproducibility, see Appendix A for the exact flags used during evaluation.

3.4 Weight rotation analysis

Recent PTQ methods alleviate difficult to quantize outliers by rotating model weight matrices (Liu et al., 2024; Ashkboos et al., 2024). To understand if our method is learning such rotations, for each model weight matrix, we compare values prior to quantization with those after SpinQuant or QAT (in the case of SpinQuant, using as initial weights the values after folding in LayerNorm scale factors) as follows. First, we measure the orthogonal Procrustes distance (Schönemann, 1966), $d_p(A, B) = \min_R d_f(RA, B)$ for left-side rotations and $d_p(A, B) = \min_R d_f(AR, B)$ for right-side rotations, where R is constrained to the set of rotation matrices and d_f is Frobenius distance, and use whichever is smaller; this metric gives us the *non-rotational distance* between weights pairs, that is, how much of the weight change *can not* be accounted for by matrix rotation. Next we measure the total distance between weight pairs using Frobenius distance and subtract from this the orthogonal Procrustes distance; this metric provides the *rotational distance*, how much of the weight change *can* be accounted for by matrix rotation. To facilitate comparison across layers, we normalize all distances by the Frobenius norm of the corresponding original, unquantized weight. Two layers, v_proj and o_proj, are rotated twice by SpinQuant, once on the left side and once on the right side, which our measurement is insensitive too, and so they are omitted from our analysis.

4 Results

4.1 Comparison with PTQ approaches

We demonstrate that SiLQ leads to significantly higher accuracy than leading PTQ techniques from the literature on the CSR, OLLMv1, and OLLMv2 benchmarks, examining averages across each set (Table 1), as well as individual benchmark values (Tables 5, 6, and 7 in the appendix). We run our method for 128,000 steps for these comparisons.

Notably, our approach produces a quantized model that, on some metrics, matches the accuracy of the original full precision model, and across each benchmark average leads to an accuracy gap of no more than 2 absolute percentage points. Across the board, our method consistently outperforms leading PTQ methods[3], often by large margins. This performance advantage is maintained even when comparing PTQ methods employing dynamic quantization to our method employing static quantization, which is simpler to implement and computationally less expensive for both training and inference.

Looking at the impact of training duration, Figure 1 shows the accuracy of our technique begins to exceed that of SpinQuant on instruction tuned models after only 500 training steps, and continues to improve with longer training. On the more modern and challenging benchmarks, OLLMv1 and OLLMv2, accuracy continues to improve even at

[3]Sections C and D in the appendix provide implementation details of SpinQuant and SmoothQuant, respectively.

the longest training duration tested, suggesting further training may still improve accuracy. This curve nicely highlights the ability of QAT to trade-off time-to-solution for accuracy.

Across 128,000 steps of QAT, the model sees 0.1%, 0.07%, and 0.08% of the tokens needed to train the full precision base model for Llama-3-8B, Llama-3.1-Tulu-3.1-8B, Granite-3.1-8B-Instruct, requiring less than 2 weeks on a single, 8-H100 node for the 8B parameter models tested. Crucially, this demonstrates that our QAT method can quantize a model with near lossless accuracy at the cost of less than a tenth of a percent increase in total model training resources.

4.2 Comparison with alternative QAT method

LLM-QAT is the only published QAT method that meets our comparison criteria of quantizing activations, cache, and weights at precisions sufficient to approach full precision accuracy. While our main focus is the more useful instruction-tuned models, as LLM-QAT applies QAT to base models, we apply our method to the Llama-2-7B base model for a direct comparison[4]. Table 2 shows that our approach using the open-source DCLM dataset, on the same model with the same number of training samples as in the LLM-QAT paper, leads to higher accuracy than LLM-QAT in considerably less time. If we use the time spent by LLM-QAT for generating samples from the model to instead perform further QAT, the accuracy comes close to that of the original model (within 1% on all banchmarks, and within 0.2% on OLLMv2, Table 2, last row). Using an open-source dataset for training is both simpler and more scalable, and saves considerable effort and computational resources compared to using synthetic data generated from the model as in (Liu et al., 2023), and can lead to better results.

4.3 Generality of our approach

For simplicity, in the above experiments we use the original SFT dataset for QAT. However, if the original dataset is not available, will QAT with a publicly available substitute suffice? To test this, we repeat the 8,000 step run for the Granite-3.1-8B-Instruct model, replacing the original closed source SFT dataset with the open source Tulu3 SFT data. QAT with the Tulu3 dataset leads to better scores across all 3 benchmarks (Table 3). This is not surprising given the quality of the dataset, however, it shows the generality of our QAT approach, since the Tulu3 dataset was constructed to optimize the scores of the Llama models, and suggests that QAT can take advantage of potential accuracy gains afforded by newer datasets.

As a further test, we use the Tulu3 SFT dataset for QAT of the Meta-Llama-3-8B-Instruct model for 8,000 steps. Although we do not have access to the SFT dataset used to train the original model for comparison, our QAT technique with the open source Tulu3 SFT/DCLM dataset mixture performs remarkably well, exceeding the original model's accuracy on two of three benchmarks (Table 3). Based on these results, QAT remains a viable option even in the absence of access to the original closed source SFT datasets, given the availability of good open source alternatives.

Table 2: Using the same number of training samples, SiLQ achieves significantly better accuracy in considerably less time than LLM-QAT on Llama-2-7B with A8d-C8-W4 precision. Time is wall-clock time in hours for training plus data generation (only needed for LLM-QAT) on 8 H100 GPUs. With similar total time, our technique improves further, approaching the full precision baseline accuracy. LLM-QAT does not quantize the head.

Method	Hours	Samples (thousands)	CSR	OLLM v1	OLLM v2
Baseline	-	-	64.09	50.64	8.39
LLM-QAT	104	96	61.42	47.33	7.56
SiLQ	1.6	96	63.13	49.08	7.64
SiLQ	100	6,080	63.37	50.38	8.19

Table 3: High quality open source datasets provide excellent results when used in place of proprietary datasets for QAT. The open source Tulu3 SFT dataset led to better accuracy than using the model's original training data for Granite-3.1-8B-Instruct. For Llama-3-8b-Instruct, the original training data is not available for our purposes, but QAT using the Tulu3 SFT dataset is sufficient to approach or exceed the accuracy of the original fp16 model. For both comparisons, the quantized model was at A8d-C8-W4 precision and underwent 8,000 steps of QAT. Green or red indicate increase or decrease relative to the described baseline.

Model	SFT Dataset	CSR	OLLMv1	OLLMv2
Granite-3.1-8B -Instruct	Original	67.45	69.45	26.54
	Tulu3 SFT	67.78 (+0.33)	70.30 (+0.85)	28.52 (+1.98)
Llama-3-8B -Instruct fp16	Original	63.60	65.95	24.01
Llama-3-8B -Instruct	Tulu3 SFT	63.80 (+0.20)	66.13 (+0.18)	23.26 (-0.75)

[4]LLM-QAT source code does not support Llama 3 models (https://github.com/facebookresearch/LLM-QAT). Appendix E provides details about LLM-QAT implementation.

4.4 Ablation studies

To better understand the space around our chosen training and quantization configuration, we perform several ablation studies (Table 4). For simplicity, we report results for each ablation using a single 8,000 step training run on Granite-3.1-8B-Instruct at A8d-C8-W4 precision. Two factors of those we looked at stood out as having a major influence on accuracy: the use of knowledge distillation, and proper activation calibration. Specifically, we found that constructing our loss function using knowledge distillation alone lead to much better results than using a mix of knowledge distillation and conventional next token prediction, or next token prediction alone. This is slightly different from prior work (Liu et al., 2023) showing that a mix of teacher labels and true target labels gave similar results to just using the teacher labels, though the difference may be attributed to their use of a base model vs our use of an instruct model. Calibrating the step size for activation quantization using the quantile approach led to much better accuracy than using the maximum value from the calibration set.

4.5 Weight analysis

To understand if SiLQ is simply learning weight matrix rotations, and therefore if there is an obvious means to match its performance by improvements to existing rotation based PTQ mehods (Liu et al., 2024; Ashkboos et al., 2024), we measure how much of the weight change produced by SiLQ can be accounted for by matrix rotation, and perform that same analysis using SpinQuant as a baseline (as described in Section 3.4). On the Granite-3.1-8B-instruct A8d-C8-W4 model (Figure 3), we find that rotation accounts for 90% of all weight changes produced by SpinQuant, with the remainder attributable to the round and clamp operations present in the quantization itself. In contrast, only 43% of the weight changes produced by SiLQ can be accounted for by rotation, suggesting that a purely rotational PTQ based approach would be unable to find a similar solution.

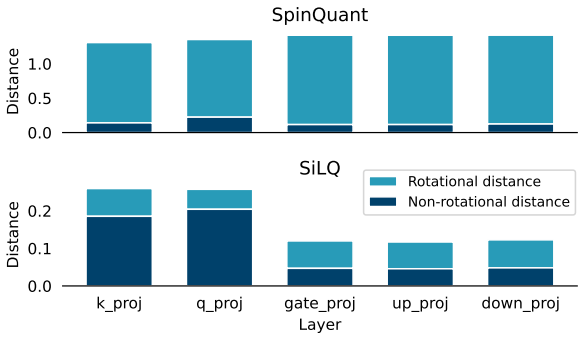

Figure 3: SiLQ solutions cannot be explained purely by weight matrix rotations; it finds solutions that SpinQuant cannot. Plotted are weight changes to the LLM linear layers as a result of QAT or SpinQuant factored into the amount explainable by matrix rotation (rotational distance), measured using orthogonal Procrustes distance, and the remainder (non-rotational distance), measured using Frobenius distance minus the orthogonal Procrustes distance. Changes are measured for each linear layer, normalized by the Frobenius norm of the starting weights, then averaged by layer type.

5 Conclusion

We demonstrate strong performance across three LLM benchmarks on base and instruct models in absolute terms, and relative to leading PTQ and QAT methods, in a commensurate comparison that equalizes quantization constraints. On Llama3-8B, our approach, quantizing activations, cache, and weights, is able to match the accuracy of the original full precision model on common sense reasoning tasks, while leading PTQ methods that quantize only weights are unable to reach this level of accuracy (for example, Table 3 in (Huang et al., 2024)) despite the more relaxed quantization requirements. Importantly, we are the first to focus on instruction-tuned models in this space, which are more likely to be deployed than base models.

We found that, even in a case with a relatively small dataset, the longer the QAT duration with knowledge distillation, the higher the accuracy of instruction-tuned models on three extensive benchmarks. This is somewhat surprising, given that further SFT training of the floating point Tulu3 model, for instance, leads to lower accuracy (see Figure 6 in (Lambert et al., 2024)), and our best QAT results are obtained with many additional epochs over this dataset with less than one million samples. Furthermore, contrary to a recent weight-only QAT LLM study which, based on much smaller LLMs (125 million parameters with 100 billion token training budget) found it best to use 10% of the training budget for QAT (Liu et al., 2025), we show that less than 0.1% of the budget for QAT can lead to accuracy close to the full-precision counterparts for models with 8-bit activation quantization. Finally, it is also surprising that, although the instruct models were originally created using a combination of

Table 4: Knowledge distillation and quantile activation calibration are critical components of SiLQ at A8d-C8-W4 precision. Results of ablation studies using Granite-3.1-8B-Instruct are shown. The first row is the baseline configuration. For ablations, *KD Ratio* is the ratio of knowledge distillation loss to next token prediction loss, *KD Temp* is the knowledge distillation temperature, *DCLM Ratio* is the ratio of DCLM data to instruct tuning data, *Act Lrx* is the multiplicative scaling factor on learning rate applied to activation quantizer scales, *Act Calib* is the activation quantizer calibration method, *Wgt Calib* is the weight quantizer calibration method, either the MSE based approach introduced here or the calibration method from the LSQ paper, *Online Rot* is whether online rotations were applied as in (Ashkboos et al., 2024). Green or red indicate increase or decrease relative to baseline.

KD Ratio	KD Temp	DCLM Ratio	Act Lrx	Act Calib	Wgt Calib	Online Rot	OLLM v1	OLLM v2
1.0	1.0	0.25	50	Quantile	MSE	No	68.65	27.67
0.0	1.0	0.25	50	Quantile	MSE	No	63.36 (-5.29)	23.35 (-4.32)
0.5	1.0	0.25	50	Quantile	MSE	No	67.74 (-0.91)	25.60 (-2.07)
1.0	0.5	0.25	50	Quantile	MSE	No	68.94 (+0.29)	26.33 (-1.34)
1.0	2.0	0.25	50	Quantile	MSE	No	69.72 (+1.07)	26.97 (-0.70)
1.0	1.0	0.0	50	Quantile	MSE	No	69.37 (+0.72)	26.23 (-1.44)
1.0	1.0	0.5	50	Quantile	MSE	No	68.99 (+0.34)	26.90 (-0.77)
1.0	1.0	0.25	1	Quantile	MSE	No	67.69 (-0.96)	27.13 (-0.54)
1.0	1.0	0.25	50	Max	MSE	No	63.98 (-4.67)	21.01 (-6.66)
1.0	1.0	0.25	50	Quantile	LSQ	No	68.18 (-0.47)	27.15 (-0.52)
1.0	1.0	0.25	50	Quantile	MSE	Yes	70.02 (+1.37)	27.27 (-0.4)

pre-training, SFT, proximal policy optimization, direct preference optimization, and model averaging, QAT with the knowledge distillation loss using only an SFT and pre-training dataset mixture restores accuracy to nearly the same level as the original model.

As the field scales up inference-time compute, low latency becomes critical, making these methods increasingly relevant for building performant LLMs capable of reasoning. A simple QAT procedure can, with a relatively short training run on a single 8-H100 node, produce efficient, quantized 8-billion parameter instruction-tuned models that can be deployed with low-power and low-latency on accelerators like NorthPole, enabling new (Yao et al., 2023) and more sustainable AI applications.

Limitations

Limitations of our work include training resources, dataset access, specific hardware constraints, specific model sizes, and limited number of trials due to resource constraints. As with QAT in general, training may take up to 2 weeks for the highest accuracy. We perform QAT on all model parameters, using 8, 80GB GPUs for the 8B models studied. We had access to the IBM internal SFT dataset for our study, however, all other datasets were open source available through HuggingFace, and we conducted experiments to verify that the internal dataset could be replaced by an open-source version. All models here were fully quantized to best match the hardware constraints of the NorthPole processor. Although activations, cache, and weights were fully integer, other operations remained at 16 bits as described in the Methods. Eight-billion parameter models were the focus of our study; although the method should generalize to larger and smaller models, 8B models are of special interest to us due to their reasonable accuracy and the fact that they can be implemented on the current NorthPole production system. Finally, due to resource constraints, we chose to use our training budget to study different model variations over different training durations with a single trial per experiment rather than to perform multiple trials on fewer models and durations. This is common with LLM training, and the clear trends across models justify the decision.

References

Rathinakumar Appuswamy, Michael V Debole, Brian Taba, Steven K Esser, Andrew S Cassidy, Arnon Amir, Alexander Andreopoulos, Deepika Bablani, Pallab Datta, Jeffrey A Kusnitz, and 1 others. 2024. Breakthrough low-latency, high-energy-efficiency llm inference performance using northpole. In *2024 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–8. IEEE.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. 2024. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240.

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

Mengzhao Chen, Yi Liu, Jiahao Wang, Yi Bin, Wenqi Shao, and Ping Luo. 2024a. Prefixquant: Static quan-

tization beats dynamic through prefixed outliers in llms. *arXiv preprint arXiv:2410.05265*.

Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, and Ping Luo. 2024b. Efficientqat: Efficient quantization-aware training for large language models. *arXiv preprint arXiv:2407.11062*.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Dayou Du, Yijia Zhang, Shijie Cao, Jiaqi Guo, Ting Cao, Xiaowen Chu, and Ningyi Xu. 2024. Bitdistiller: Unleashing the potential of sub-4-bit llms via self-distillation. *arXiv preprint arXiv:2402.10631*.

Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. 2019. Learned step size quantization. In *International Conference on Learning Representations*.

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

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2023. A framework for few-shot language model evaluation.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, and 1 others. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7).

Wei Huang, Xingyu Zheng, Xudong Ma, Haotong Qin, Chengtao Lv, Hong Chen, Jie Luo, Xiaojuan Qi, Xianglong Liu, and Michele Magno. 2024. An empirical study of llama3 quantization: From llms to mllms. *Visual Intelligence*, 2(1):36.

Hugging Face SFTTrainer. Sft trainer — trl documentation. https://huggingface.co/docs/trl/en/sft_trainer. Accessed: 2025-05-16.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, and 1 others. 2024. T\" ulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*.

Jemin Lee, Sihyeong Park, Jinse Kwon, Jihun Oh, and Yongin Kwon. 2024. A comprehensive evaluation of quantized instruction-tuned large language models: An experimental analysis up to 405b. *arXiv preprint arXiv:2409.11055*.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Yitzhak Gadre, Hritik Bansal, Etash Guha, Sedrick Scott Keh, Kushal Arora, and 1 others. 2024. Datacomp-lm: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems*, 37:14200–14282.

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

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. 2024. Spinquant: Llm quantization with learned rotations. *arXiv preprint arXiv:2405.16406*.

Zechun Liu, Changsheng Zhao, Hanxian Huang, Sijia Chen, Jing Zhang, Jiawei Zhao, Scott Roy, Lisa Jin, Yunyang Xiong, Yangyang Shi, and 1 others. 2025. Paretoq: Scaling laws in extremely low-bit llm quantization. *arXiv preprint arXiv:2502.02631*.

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

Dharmendra S Modha, Filipp Akopyan, Alexander Andreopoulos, Rathinakumar Appuswamy, John V Arthur, Andrew S Cassidy, Pallab Datta, Michael V DeBole, Steven K Esser, Carlos Ortega Otero, and 1 others. 2023. Neural inference at the frontier of energy, space, and time. *Science*, 382(6668):329–335.

Peter H Schönemann. 1966. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10.

Yikang Shen, Matthew Stallone, Mayank Mishra, Gaoyuan Zhang, Shawn Tan, Aditya Prasad, Adriana Meza Soria, David D Cox, and Rameswar Panda. 2024. Power scheduler: A batch size and token number agnostic learning rate scheduler. *arXiv preprint arXiv:2408.13359*.

https://github.com/facebookresearch/LLM-QAT. https://github.com/facebookresearch/LLM-QAT, Accessed: 2025-06-19.

Hongyu Wang, Shuming Ma, and Furu Wei. 2024. Bitnet a4. 8: 4-bit activations for 1-bit llms. *arXiv preprint arXiv:2411.04965*.

World Economic Forum. 2024. How to manage ai's energy demand—today, tomorrow and in the future. https://www.weforum.org/stories/2024/04/how-to-manage-ais-energy-demand-today-tomorrow-and-in-the-future/. Accessed: 2025-06-06.

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

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

A Evaluation Details

We use the following lm-evaluation-harness flags to run each task in the OLLMv1 leader-board and the 8 common-sense reasoning tasks on base models:

```
--model_args pretrained=<model_path>,
      dtype=bfloat16,max_length=4096 \
--tasks <task_name> \
--num_fewshot <num_shots> \
--model hf \
--batch_size auto \
--show_config \
--trust_remote_code
```

For the OLLMv2 leader-board, the flags are

```
--model_args pretrained=<model_path>,
      dtype=bfloat16 \
--tasks leaderboard \
--model $model_type \
--batch_size auto \
--show_config \
--trust_remote_code
```

For instruction-tuned models, we add the following flags to the above for all tasks:

```
--apply_chat_template --fewshot_as_multiturn
```

B Training Details

As described in section 3.1, during QAT, base models are trained using the publicly available DCLM dataset and instruct models are trained using a mixture of SFT data and DCLM. For Llama-3.1-Tulu-3.1-8B we use the tulu-3-sft-mixture (Lambert et al., 2024), and for Granite-3.1-8b-instruct[5], we use the same SFT dataset used to train the original model. For Llama models, we found it necessary during training to use the LlamaTokenizerFast tokenizer class which removes the bos_token following the SpinQuant source code.

To train, we use SFTTrainer (Hugging Face SFTTrainer) with the AdamW optimizer (beta1=0.9, beta2=0.95, epsilon=1e-10, bfloat16) (Loshchilov and Hutter, 2017), employ cross entropy loss with labels derived from the knowledge distillation teacher at a temperature of 1, disable dropout to avoid adverse affects on knowledge distillation, use a base learning rate of 5e-6 for 8,000 steps with a cosine learning rate schedule and a minimum learning rate set to 10% of the initial value, no warm-up period, and use a weight decay of 0.1. The learning rate was the best from {2e-6, 5e-6, 1e-5} at 8,000 training steps for both Llama-3-8B and granite-3.1-8b-instruct. The learning rate is reduced for longer training runs by the inverse square-root of the increase in training steps, e.g. when increasing training steps by a factor of 4, the learning rate is reduced to half (Shen et al., 2024). Similarly, for shorter runs, the learning rate is increased by the square-root of the decrease in steps. We train with a batch size of 128 with sequence length 1024, without packing on a single 8-H100 GPU node.

C SpinQuant Comparison

We compare our technique on all models with SpinQuant, the leading state-of-the-art PTQ method for 4-bit weights and 8-bit activations at the time of this publication. For a fair comparison, we run the SpinQuant author's code[6] on the original bfloat16 publicly available models. To avoid the additional inference overhead due to online rotations, we modify the code to support the no online Hadamard rotations option described in the paper. We use the default parameters from the paper with the following changes to match the hardware constraints of our target deployment platform. We use symmetric quantization for the activations with no groups for most activations in the network. We use group quantization for only the key cache and not the value cache, with group size 128, as the extra key cache group-wise scales can be integrated more readily into the matrix multiply operation on chip. The final head linear layer inputs and weights are also quantized to 8-bits.

D SmoothQuant Comparison

For an additional comparison, we also compare against SmoothQuant for all models. We use the authors' original code modified to use 4-bit weights instead of 8, and to control the precision of the KV-cache at either 4- or 8-bits while leaving the output of the query linear layer unconstrained to more closely match SpinQuant and our QAT precision

[5]https://huggingface.co/ibm-granite/granite-3.1-8b-instruct

[6]https://github.com/facebookresearch/SpinQuant

Table 5: Comparison of accuracy on zero-shot CSR tasks with leading PTQ techniques.

Model	Bits A-C-W	Method	ARC-e	ARC-c	BoolQ	PIQA	SIQA	HellaS.	OBQA	WinoG.
Llama-3-8B	16-16-16	Baseline	77.9	53.0	81.0	81.0	46.9	79.2	45.0	72.7
	8d-8-4	SmoothQuant*	63.05	38.57	72.97	74.43	43.76	70.56	39.20	67.32
		SpinQuant	74.66	47.44	72.51	78.62	45.50	76.84	43.40	72.77
		SiLQ	79.88	52.47	81.44	79.82	46.47	78.26	45.60	73.64
Llama-3.1 -Tulu-3.1-8B	16-16-16	Baseline	81.23	55.29	85.29	80.63	55.83	79.29	49.40	72.30
	8d-8-4	SmoothQuant*	73.06	47.35	82.39	77.15	49.95	73.47	43.60	66.61
		SpinQuant	78.24	52.56	76.88	80.90	51.54	79.31	47.20	73.16
		SiLQ	81.23	54.18	85.38	81.07	55.17	78.74	48.00	72.93
Granite-3.1-8B -Instruct	16-16-16	Baseline	75.08	53.41	88.87	78.84	55.17	77.89	49.60	68.82
	8d-8-4	SmoothQuant*	65.82	45.90	85.78	75.08	48.36	73.17	42.00	65.35
		SpinQuant	69.49	47.53	85.99	78.67	50.97	75.78	45.80	73.48
		SiLQ	75.04	53.24	88.41	78.62	54.71	76.87	48.40	68.98
	8s-8-4	SmoothQuant*	46.89	34.47	64.71	64.58	40.23	52.60	34.80	54.22
		SiLQ	73.48	52.39	88.17	78.45	54.35	76.86	46.4	69.14
	8d-4-4	SmoothQuant*	59.01	41.13	75.54	71.22	45.19	64.64	38.80	56.35
		SpinQuant	68.01	45.65	85.41	76.06	48.87	71.38	42.40	67.17
		SiLQ	74.37	52.90	88.38	78.73	55.12	76.61	47.80	69.46

*head not quantized

Table 6: Comparison of accuracy on OLLMv1 tasks with leading PTQ techniques.

Model	Bits A-C-W	Method	ARC-c	HellaS.	MMLU	TruthfulQA	WinoG.	GSM8K
Llama-3-8B	16-16-16	Baseline	59.04	81.97	65.34	43.95	76.80	48.82
	8d-8-4	SmoothQuant*	46.08	67.63	46.34	35.85	70.09	4.93
		SpinQuant	56.14	79.13	60.99	40.97	75.14	35.56
		SiLQ	56.48	81.14	63.79	44.20	76.16	45.03
Llama-3.1 -Tulu-3.1-8B	16-16-16	Baseline	57.85	81.84	61.87	59.86	74.66	81.27
	8d-8-4	SmoothQuant*	49.49	74.51	47.50	54.69	71.19	51.33
		SpinQuant	55.55	81.15	58.39	52.81	74.35	79.23
		SiLQ	59.64	81.36	62.36	58.20	74.27	82.94
Granite-3.1-8B -Instruct	16-16-16	Baseline	62.62	84.48	65.34	66.23	75.37	73.84
	8d-8-4	SmoothQuant*	57.59	78.96	55.96	63.25	73.24	40.94
		SpinQuant	58.70	79.79	58.35	59.74	78.77	57.77
		SiLQ	66.98	84.15	65.46	64.66	76.16	71.49
	8s-8-4	SmoothQuant*	27.65	40.31	27.04	55.32	50.20	10.92
		SiLQ	65.87	84.01	64.7	65.61	74.74	70.2
	8d-4-4	SmoothQuant*	38.99	59.98	28.96	53.10	54.93	2.73
		SpinQuant	54.69	74.80	44.26	57.55	72.69	36.01
		SiLQ	65.44	84.00	65.04	65.81	74.90	70.36

*head not quantized

Table 7: Comparison of accuracy on OLLMv2 tasks with leading PTQ techniques.

Model	Bits A-C-W	Method	BBH	GPQA	IFEval	MATH	MMLU-Pro	MUSR
Llama-3-8B	16-16-16	Baseline	23.92	7.61	15.64	4.46	25.01	5.55
	8d-8-4	SmoothQuant*	6.10	1.68	16.95	3.20	6.53	5.46
		SpinQuant	21.12	2.85	15.31	3.78	21.31	9.28
		SiLQ	22.67	6.15	14.33	4.31	21.90	6.61
Llama-3.1 -Tulu-3.1-8B	16-16-16	Baseline	18.25	2.74	82.40	22.36	21.11	11.82
	8d-8-4	SmoothQuant*	11.70	1.34	76.22	19.26	9.82	4.60
		SpinQuant	17.53	3.02	78.44	16.99	16.84	11.37
		SiLQ	20.60	1.73	82.80	23.64	21.11	12.71
Granite-3.1-8B -Instruct	16-16-16	Baseline	33.84	8.72	69.80	21.22	28.14	17.75
	8d-8-4	SmoothQuant*	20.55	5.70	49.17	17.90	17.33	9.79
		SpinQuant	25.37	7.94	54.95	6.57	18.90	14.38
		SiLQ	33.12	8.05	70.19	18.81	28.40	16.28
	8s-8-4	SmoothQuant*	4.20	2.96	39.51	3.71	1.62	1.60
		SiLQ	33.18	9.17	68.7	18.2	27.66	17.24
	8d-4-4	SmoothQuant*	5.13	0.06	29.45	0.64	1.50	6.10
		SpinQuant	13.43	3.80	46.41	2.64	9.25	11.27
		SiLQ	33.15	9.40	70.29	18.05	27.43	16.43

*head not quantized

settings. We chose alpha=0.4 as the best among 0.2, 0.3, ... 0.7 for the Granite-3.1-8b-instruct A8d-C8-W4 model, and used the same value for the remaining models. Note that the final head linear layer remains unconstrained in their code.

E LLM-QAT Comparison

We use the original code from the authors for comparison using default parameter settings for both data generation and training (https://github.com/facebookresearch/LLM-QAT). Note that the LLM head is not quantized in their code. We train for 750 steps for our method in order to compare with the same number of samples, 96,000 as in (Liu et al., 2023), and adjust the learning rate to 1.63e-5 accordingly. We perform a second run training for 47,500 steps at learning rate 2e-6 to approximately match the total wall clock time used by LLM-QAT for data generation and training. We set the max sequence length to 2048 to match LLM-QAT. We compare our method with LLM-QAT only on llama-2-7b, because the code provided does not support Llama-3 models.