# Exploring Quantization for Efficient Pre-Training of Transformer Language Models

Anonymous ACL submission


## Abstract

The increasing scale of Transformer models has led to an increase in their pre-training computational requirements. While quantization has proven to be effective after pre-training and during fine-tuning, applying quantization in Transformers during pre-training has remained largely unexplored at scale for language modeling. This study aims to explore the impact of quantization for efficient pre-training of Transformers, with a focus on linear layer components. By systematically applying straightforward linear quantization to weights, activations, gradients, and optimizer states, we assess its effects on model efficiency, stability, and performance during training. By offering a comprehensive recipe of effective quantization strategies to be applied during the pre-training of Transformers, we promote high training efficiency from scratch while retaining language modeling ability.


## 1 Introduction

Transformers (Vaswani et al., 2017) have become the dominant model for natural language processing, with the GPT family of models (Radford et al., 2019) showcasing their effectiveness across various tasks, from language understanding to code generation. As the performance of Transformers scales nicely with the number of parameters and the data size, the current state-of-the-art models have reached unprecedented computational requirements both during training and inference (Tay et al., 2020). For instance, pre-training a 175B parameter GPT-3 model requires a staggering number of 10,000 V100 GPUs for 14.8 days (Patterson et al., 2021). As a result, pre-training has become extremely expensive, beyond the reach of most research groups, and has raised concerns over sustainability due to CO2 emissions from extensive GPU usage (Luccioni et al., 2023).

To improve the efficiency of Transformers, quantization has gained significant traction due to its recent successes in both post-training (Ashkboos et al., 2024; Dettmers et al., 2022; Frantar et al., 2022) and during fine-tuning (Li et al., 2023). Quantized pre-training, where certain parts of the computational graph and model parameters are quantized from the beginning of training, remains a challenging problem. In such scenarios, the training instabilities caused by substantial changes in model parameters and emerging model behaviors do not pair well with the added noise introduced by quantization (Nagel et al., 2022). Additionally, quantizing model components without compromising performance becomes increasingly difficult at larger scales (Dettmers and Zettlemoyer, 2023). Despite its importance, quantized pre-training of Transformer language models remains largely unexplored at scale.

In this paper, we present the first in-depth study on the effects of quantizing Transformer language models during pre-training and at scale. Our primary aim is to provide a recipe for quantized pre-training by conducting a controlled study that investigates the impact of quantization on weights, activations, gradients, and optimizer states on model efficiency, stability, and performance, using a simple linear quantization with 4 and 8 bits. Our findings demonstrate that 8-bit quantization for weights and activations can be effectively combined to provide significant memory savings and potential speedup, achieving performance comparable to the baseline model. However, extending quantization to gradients to utilize computational speedup in backward matrix multiplications or reducing precision to 4 bits results in notable training instability.

Specifically, 4-bit quantization introduces a sharper loss landscape for weights (§4.1) and persistent outliers in the channel dimension of activations (§4.2), significantly degrading performance despite attempts to manage them through per-channel quantization. Additionally, gradient quantization is particularly problematic due to spikes in gradient norms during early training phases and the unstructured and sparse nature of gradients throughout training (§4.3). While 8-bit gradient quantization does not hurt model convergence, transitioning to 4 bits results in non-convergence. Finally, quantizing the first-order moments of Adam to even 4 bits is feasible without significant performance loss, but the second-order moments require a more complex quantization scheme to avoid instabilities in the Adam update, even when using 8-bits qauntization (§4.4).

## 2 Related Work

In recent years, numerous methods have been studied to improve the efficiency of neural networks. Among these, Quantization-aware training (QAT) emerges as an acceleration technique for inference, as parameters are stored and operations are conducted with higher precision during training. The induced quantization error during training serves as a regularizer, as demonstrated in Gholami et al. (2022), ultimately facilitating the development of a more quantization-friendly model.

In contrast, Fully Quantized Training (FQT) harnesses the accelerated gains derived from higher throughput in INT8 or INT4 operations supported by modern GPUs during training. Additionally, FQT capitalizes on memory savings by storing parameters in lower precisions. As exemplified by Li et al. (2024) and Dettmers et al. (2021), states of the Adam optimizer are stored in 4 and 8 bits, respectively, to minimize memory footprint. Another strategy introduced by Markov et al. (2023) involves quantizing both weights and gradients to reduce bandwidth usage in distributed training. Furthermore, (Wortsman et al., 2024) and (Kim et al., 2021) advocate the replacement of linear operations with INT8 matrix multiplications to achieve substantial speedups.

Despite the considerable advantages in accelerating the training process, FQT poses challenges attributable to numerical stability and optimization issues inherent in training quantized networks. Many existing methods predominantly focus on fine-tuning Large Language Models (LLMs) using FQT, leveraging the inherent stability of pre-trained models in each gradient update. The evolution from 16-bit to FP8 data formats, as evidenced by remarkable results in mixed precision training on LLMs (Peng et al., 2023), showcases the potential of FQT. However, the scarcity and difficulty in obtaining hardware that supports FP8 formats pose significant challenges to its widespread adoption.

Wortsman et al. (2024) employ 8-bit quantization for linear operations in both forward and backward passes, achieving remarkable results in pre-training large-scale vision language models. Their approach incorporates row-wise quantization for activations and gradients, mitigating the impact of quantization errors on other parameters. However, vision language models significantly differ from large textual language models. Xi et al. (2024b) proposed 4-bit quantization in FQT with Hadamard transformation to address outliers in activations. Acknowledging the sparsity of gradients during training, they propose bit splitting for quantizing gradients in 4-bit precision. However, they predominantly focus on fine-tuning, and their experiments on pre-training only consider small models (60M) on a limited dataset (WMT 14 En-De).

The generalization of these findings to large-scale language models (LLMs) remains a challenge, especially considering that training such models involves unique complexities. Additionally, their reliance on Hadamard transformations imposes restrictions on activation dimensions, limiting applicability to power-of-two dimensions, a constraint not easily met by recent LLM architectures such as LLaMa.

In contrast to the aforementioned complex quantization methods, such as quantile quantization and learnable quantization parameters, this study focuses on a more straightforward implementation. The objective is to investigate the effects of quantization and explore the feasibility of training LLMs with full integer operations. The primary goal of this paper is to offer detailed insights into quantizing different components of the model. While more sophisticated quantization methods have the potential to enhance the final performance, our work serves as a foundational investigation, providing valuable insights and paving the way for further exploration in the realm of quantization for LLMs.

## 3 Quantization Methodology

In this work, we explore various quantization schemes (§3.1) and granularities (§3.2) on model components. This controlled approach during pre-training allows us to examine the impacts on language modeling and downstream task performance.

### 3.1 Quantization Scheme

We start by introducing the quantization procedure used in our study, which is applied to all linear layers of Transformers. We perform fake quantization, where all values and computations are stored with higher precision, and every quantization operation is followed by de-quantization to introduce quantization error. Simulating low-precision pretraining fits the purpose of this study since we aim to analyze the effects of quantizing different model components without focusing on actual training speedups that can be obtained by implementing custom GPU kernels.

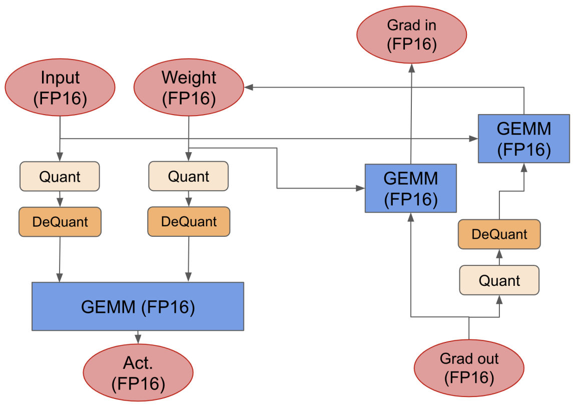


Figure 1: Overview of the quantization process in forward and backward passes.

Figure 1 illustrates how quantization error is introduced during both the forward and backward passes. Specifically, the quantization error is injected into either the weights or input activations during the forward pass. In the backward pass, quantization is applied to the output gradient for computing weight updates due to only the gradient of the weight is stored. Nonetheless, the real-valued output gradient is used to compute the quantized input gradient since we observed an increase in training instability when propagating the quantization error through the entire backward path. Additional details about gradient quantization are presented in Section 4.3.

We employ linear quantization for our experiments since this is a popular approach compatible with existing hardware, and using more complex methods could potentially hinder the practical relevance of our study. Specifically, we map real-valued vectors $\boldsymbol{X}$ to a discrete grid of integers as follows:

$$X_{\text{int}} = \text{clip}\left(\left\lfloor \frac{X}{s} \right\rceil - z; N, P\right),$$
$$\widehat{\boldsymbol{X}} = s(X_{\text{int}} + z), \tag{1}$$

where $\lfloor \cdot \rceil$ is the round-to-nearest integer operator, $N$ and $P$ represent the quantization range, with $N = -2^{b-1}$, $P = 2^{b-1} - 1$, and $b$ is the bitwidth, since we deal with signed data in our experiments. The scaling factor $s$ is set to the maximum absolute value of $\boldsymbol{X}/P$. Unless specified, we perform symmetric quantization by setting the offset $z$ to 0, which has less overhead than asymmetric quantization where $z$ is set to $\lfloor \min(X)/s \rceil$ (Nagel et al., 2021). During backpropagation, we employ the well-known straight-through estimator (STE) (Bengio et al., 2013) mechanism to update the weights.

### 3.2 Quantization Granularity

We can choose scaling factors with different granularity: per-tensor, per-channel, and per-token quantization, where each quantization granularity leads to a specific trade-off between efficiency and performance. Specifically, per-tensor quantization offers the highest efficiency since it performs a single element-wise floating-point multiplication for the de-quantization step. However, since only a single value is used to rescale the entire tensor, performance degradation is likely to occur due to such uniform scaling across the tensor elements.

On the other hand, per-channel and per-token quantization offer a finer-grained scaling, where different scaling factors are tailored to specific tensor element groups (*i.e.* channels and tokens, respectively). Even though such approaches help in terms of performance, they introduce an overhead during the de-quantization step. It is worth noting that, in certain instances, these quantization granularities cannot be efficiently implemented by hardware-accelerated GEMM kernels. For example, using per-channel quantization for both weights and activations can not be efficiently implemented.

### 3.3 Quantization Efficiency

To show the potential memory saving in quantized pre-training, we explore the memory consumption of various components within GPT-2 models during training using the PyTorch Memory Profiler. We analyze peak memory usage, as shown in Figure 2, which illustrates memory usage for different

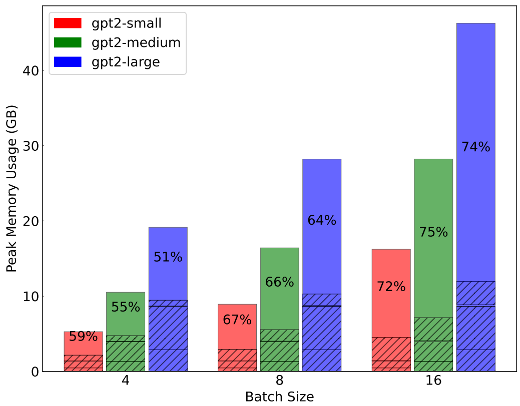


Figure 2: Distribution of peak memory usage across different model sizes (GPT-2 Small, Medium, and Large) for a constant context length of 1024, with varying batch sizes. Memory usage for model components other than activations is hatched.

batch sizes with a fixed context length of 1024 across various model sizes. We observe that when a model can fit within the GPU memory, the majority of the memory at peak times is consumed by activations, particularly with large batch sizes and sequence lengths. Under these conditions, gradients do not contribute to peak memory usage. More details are in Appendix B.

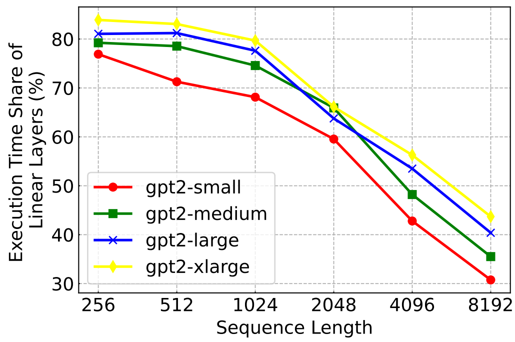


Figure 3: Proportion of total execution time consumed by linear layers in the attention block of GPT-2 models (Small, Medium, Large, and X-Large) across different sequence lengths.

We also profile the execution time of kernels using the Nvidia Nsight Profiler to assess the potential speedup from quantizing linear layers. Figure 3 shows the proportion of total execution time consumed by linear layers in the attention block of GPT-2 models of varying sizes across different sequence lengths. This profiling includes both the forward and backward passes. We observe that for small sequence lengths, linear layers consume a significant portion (more than 80%) of the execution time. As the model size increases, this proportion typically rises, but as sequence lengths increase, the proportion of time spent in linear layers decreases, suggesting that self-attention, due to its quadratic computational complexity, becomes the dominant factor in execution time. This indicates that while quantizing linear layers can offer substantial speedup, the potential gains are more pronounced with smaller sequence lengths.

## 4 Experimental Results

We used GPT-2 small (Radford et al., 2019) (124M) with FlashAttention2 (Dao et al., 2022) for our experiments due to its popularity and remarkable performance-to-compute ratio. For the following experiments, we pre-trained 30 models from scratch on OpenWebText (Gokaslan and Cohen, 2019) for 300k gradient steps with a global batch size of 512 samples and a context length of 1024 tokens. For our evaluation setup, we evaluate the performance of the models on a range of language tasks, including ARC-Easy (Yadav et al., 2019), ARC-Challenge (Yadav et al., 2019), Hellaswag (Zellers et al., 2019), LAMBADA (Paperno et al., 2016), and GLUE score (Wang et al., 2018). Additional details about our training and evaluation setups are provided in Appendix A. [1]

### 4.1 Weight Quantization

The validation loss curves of applying per-tensor and per-channel quantization to weights with 4 and 8 bits are presented in Figure 4 (down). We observe that per-channel weight quantization with 8 bits outperforms the floating-point baseline since the beginning of training in terms of validation loss, while per-tensor weight quantization with 8 bits shows competitive performance. When quantizing to 4 bits, there is a substantial difference between the different granularities, with per-channel weight quantization significantly outperforming per-tensor quantization, as previously discussed in § 3.2.

We also evaluate the downstream task performance of the quantized pre-trained models in Figure 4 (top). We observe that 8-bit weight quantization outperforms 4-bit and achieves competitive performance compared to the floating point baseline, independently of the granularity used. Once again, per-channel weight quantization with 8 bits achieves the best performance among the tested quantization schemes. Overall, we observe similar

[1] We will release the code upon the acceptance of the paper.

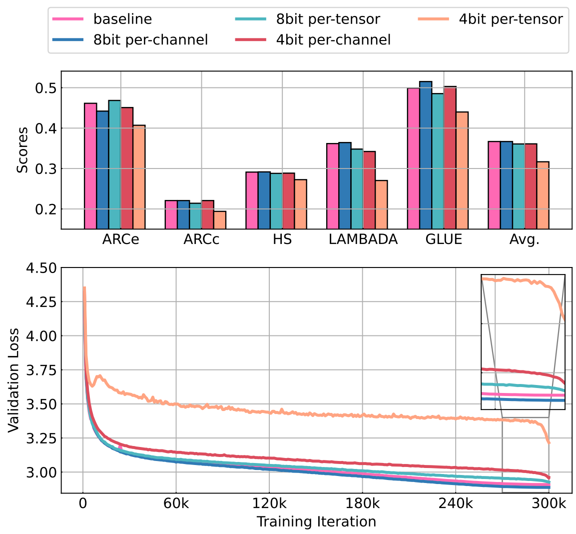


Figure 4: Comparison of different **Weight Quantization** schemes. (Down) Validation loss across training iterations for 4-bit and 8-bit quantization, both per-tensor and per-channel, alongside the baseline. (Top) Performance on downstream tasks for the corresponding quantization approaches, demonstrating the efficacy of 8-bit per-channel weight quantization.

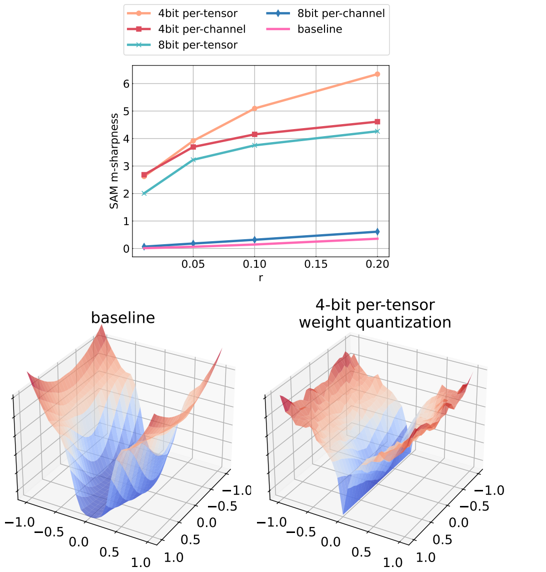


Figure 5: Sharpness comparison between baseline model and 4-bit weight quantization. (Top) $m$-sharpness. (Down) Loss surfaces.

findings when comparing the performance of the different methods during the pre-training and downstream phases. Despite the success in quantizing weights to 8 bits from scratch, we note that only performing 8-bit quantization post-training also works well, as shown in Appendix C (Table 10). However, when it comes to 4-bit quantization, applying quantization from scratch leads to significantly better performance.

We note a pronounced drop in validation loss at the end of training with the 4-bit weight quantization schemes, as seen in the final training iterations in Figure 4 (down). We hypothesize that this is related to reducing the learning rate below $1e-6$ in the final steps of training in our setup. However, this is not observed in all the schemes. Nagel et al. (2022) suggested that the oscillations in weights originated from performing quantization with STE during training may result in weight movements around decision thresholds. Hence, in our use case, we hypothesize that the drop in validation loss stems from the presence of sharp minima when quantizing weights to lower bit-widths and that the lower learning rate regime at the end of training helps convergence to minima with lower loss.

To further investigate this, we compare the sharpness of the different models at the end of pre-training using $m$-sharpness (Foret et al., 2021) with varying radii in Figure 5 (down). We observe that all quantized models converge to sharper minima compared to the floating-point, unquantized baseline. Moreover, there is a direct relation between the sharpness of each quantized model and the relative drop in validation loss observed in Figure 4 (down). Specifically, per-tensor weight quantization with 4 bits shows the highest sharpness and also the highest drop in validation loss. This correlation is also observed on a smaller scale for the per-tensor weight quantization model to 4 bits and the per-tensor weight quantization model to 8 bits. To visualize the loss surfaces, we employ the visualization method introduced by Li et al. (2018). The loss surfaces of the baseline and the per-tensor weight quantization model to 4-bits are shown in Figure 5 (right), further illustrating the impact that quantization during pre-training has on the sharpness of the final pre-trained model.

## 4.2 Activation Quantization

We present the validation loss when quantizing activations during pre-training in Figure 7 (down). We note that quantizing activations to 4 bits is more challenging than training with 4-bit weights, as noticed by previous work (Xiao et al., 2023) and as seen by the divergence behavior of per-token and per-tensor activation quantization. On the other hand, quantizing activations to 8 bits works well, especially if performed per-token, which achieves

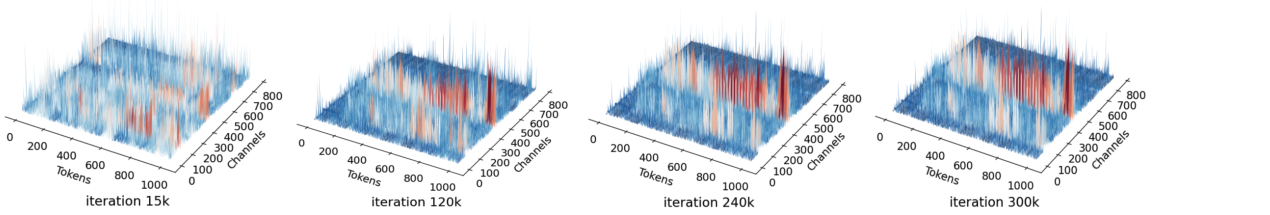


Figure 6: Training progression of activation distributions across selected iterations, showing persistent channel-specific outliers

lower validation loss than the floating-point baseline. The performance of downstream tasks of the baseline and quantized models is presented in Figure 7 (top). We notice a similar performance trend across the models, with 8-bit per-token activation reaching competitive performance with the baseline model.

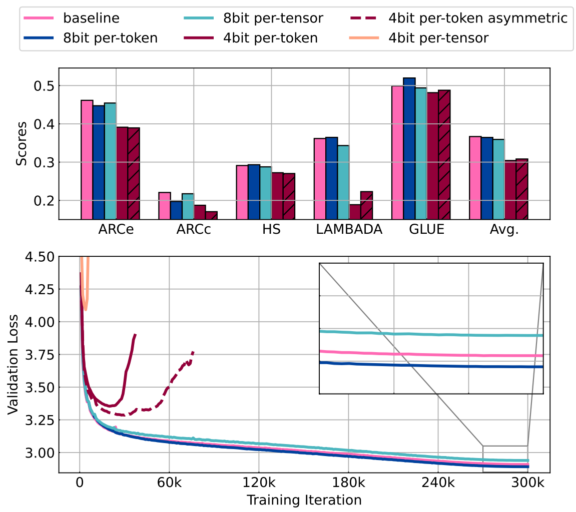


Figure 7: Pre-training **Activation Quantization** effects: (Down) Validation loss curves for various quantization schemes; (Top) Downstream task Performance across Datasets, showing 8-bit quantization closely aligns with baseline performance.

To investigate 4-bit activation quantization further, we also tried applying an asymmetric scheme. The intuition is that, while most activations exhibit symmetry around zero, this is not the case for activation after GELU activation functions (Hendrycks and Gimpel, 2016). Hence, having an asymmetric scheme can lead to a better utilization of the available bits for representation. However, we observe that while the asymmetric scheme provides an improvement over 4-bit per-token symmetric quantization, the model still diverges. We analyze the activation distributions of the output projection layer within the attention block of layer 7 in Figure 6. We see that outliers predominantly reside within specific channels and persistently affect the same channels throughout training. Given our use of per-tensor and per-token quantization for activations, it is evident that such outliers can influence all tokens, given their consistent pattern across the channel dimension.

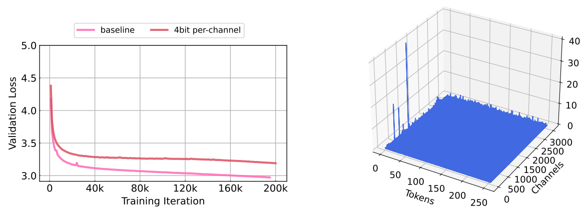


Figure 8: Validation loss and activation outliers for 4-bit per-channel quantization. (Left) Validation loss shows convergence but underperforms compared to the baseline. (Right) Histogram of activation with massive outliers.

Since activation outliers are mostly predominant in particular channels during training, we explore the efficacy of per-channel 4-bit activation quantization in Figure 8 (left). We observe that this variant does converge, even though it fails to be competitive with the floating-point baseline. Such degradation in validation loss can be attributed to the presence of massive outlier activations in specific layers. Despite being important for the model's performance (Sun et al., 2024), these big activations pose a challenge for both per-token and per-channel quantization. An example of these is presented in Figure 8 (right), showing the presence of large activations in the FC2 layer in the final attention block.

### 4.3 Gradient Quantization

We perform gradient quantization and present the validation losses obtained during pre-training in Figure 9 (down). We observe that, with 4-bit gradient quantization, training becomes highly unstable or completely fails to converge. With 8 bits, only per-token quantization converges despite showing worse performance compared to our baseline model. The observations are similar when measuring the performance of the different models on downstream tasks, as shown in Figure 9 (top).

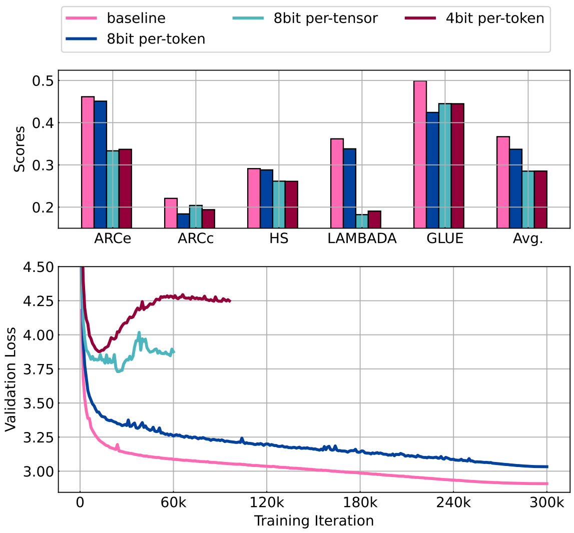


Figure 9: **Gradient Quantization**. (Down) Validation loss showing non-convergence for 4-bit and 8-bit per-tensor quantization. (Top) Performance on downstream tasks, with only 8-bit per-token approaching baseline performance.

As previously discussed in §3.1, we quantize the output gradients only for the weight updates, avoiding the instability in training that can result from propagating quantization errors when quantizing activation gradients. This is illustrated in Figure 10 (top), where quantizing activation gradients in the initial training stages leads to an explosion in the validation loss, followed by divergence. Moreover, we observe an increase in the L2 norm between the floating-point gradients and the quantized counterparts when quantizing activation gradients compared to weight gradients.

To further analyze the subpar performance of 8-bit per-token gradient quantization, even when only applied to weight gradients, we analyze the gradients for the QKV projection at the first layer of the model early on in training in Figure 10 (down). We observe that gradients are mostly sparse during training and are prone to induce high quantization errors, rendering instabilities.

### 4.4 Optimizer States Quantization

We quantize optimizer states, particularly the first and second moments in the Adam optimizer. Specifically, the quantized values of each state are

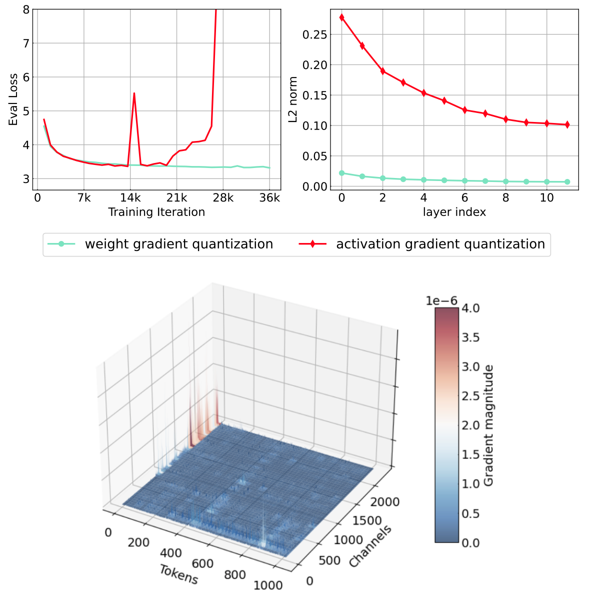


Figure 10: (Top) Validation loss spike illustrating instability from quantizing activation gradients. (Down) Gradient magnitude histogram for a linear layer, highlighting sparsity and potential for quantization error.

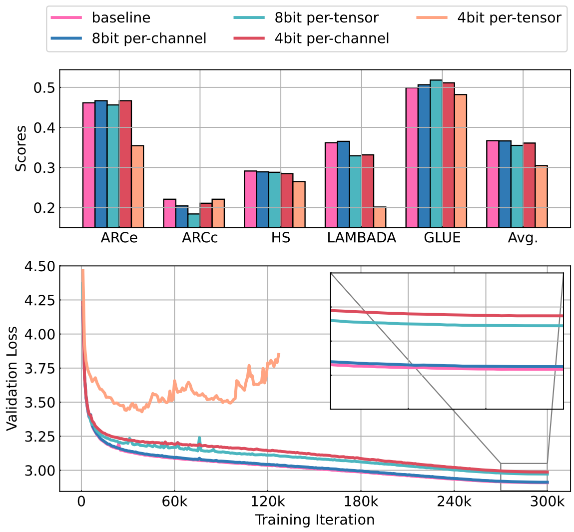


Figure 11: Quantization of **Adam optimizer**'s first-order moments. (Down) Comparison of validation loss for different quantization approaches; (Top) Performance on downstream tasks, with 8-bit per-channel closely matching baseline.

stored until the next training iteration, which are then dequantized and used for Adam's update. To better assess the effect of quantizing each state, we quantize them separately and individually. The validation losses when quantizing Adam's first state are presented in Figure 11 (down). We observe that per-channel quantization to 8 bits works well, achieving performance similar to that of the baseline model. Notably, only per-tensor quantization to 4 bits failed to converge out of all tested configurations. Similar findings are found when evaluating the downstream performance of the different quantization schemes in Figure 11 (right).

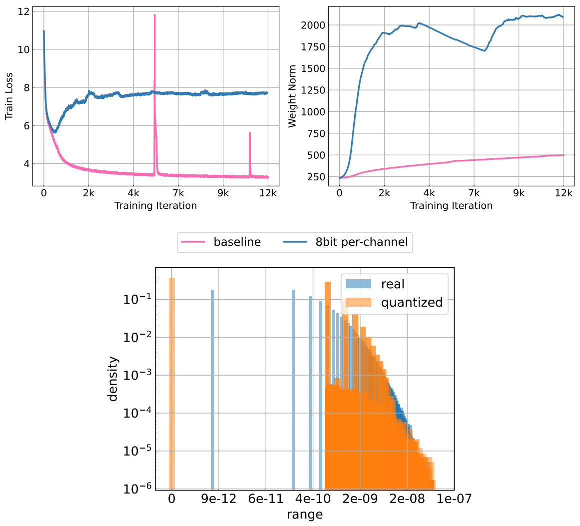


Figure 12: Second-order moments of Adam quantization. (Top) Validation loss quickly diverges. (Down) Histogram showing the concentration of quantized values in the zero bin, which contributes to instability in weight updates.

The results of quantizing Adam's second state are presented in Figure 12 (left). We observe that the quantized model failed to converge smoothly throughout training, even when applying per-channel quantization with 8 bits. This can be explained by the usage of a linear symmetric quantization function around zero in our scheme. This causes all small values to be set to zero after quantization, hurting performance, as presented in Figure 12 (right). Given that the second state plays a pivotal role in the denominator of Adam's update, such clustering to zero leads to excessively large weight updates, causing training to diverge from the onset, as observed in Figure 12 (left).

### 4.5 Multiple Components Quantization

As our last studied components, we trained models using 8-bit quantization for weights, activations, and gradients, as well as isolating quantization to only weights and activations with per-channel granularity for weights and per-token granularity for activations and gradients. Our findings, as depicted in Figure 13, demonstrate that quantizing both weights and activations to 8-bit allows performance to closely align with the baseline model.

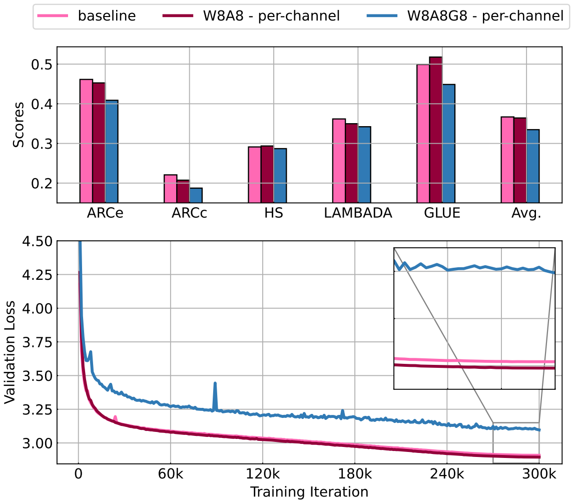


Figure 13: Validation loss across training iterations for weight, activation, and gradient quantization together.

However, extending quantization to include gradients results in a notable decrease in performance. This observation is consistent with earlier results from our independent quantization of gradients, highlighting the significant challenges this introduces.

## 5 Conclusion

This study presents an extensive analysis of the impact of quantizing specific Transformer components in 4 and 8 bits during pre-training, in contrast to concurrent work that focuses on individual methods without comprehensive ablations (Xi et al., 2024a). Our study reveals that quantizing weights to 8-bits from the beginning of pre-training is generally successful. However, 4-bit weight quantization can significantly affect model convergence due to a sharper loss landscape. We also found that carefully managing activation outliers is crucial to avoid performance drops with lower-precision quantization. Additionally, we explain the sensitive nature of gradient quantization and its potential to fail at lower bit-widths. We observed that while the first-order moments of the Adam optimizer can be effectively quantized to 4-bits, the second-order moments pose a greater challenge even for 8-bit quantization. Overall, our work establishes an important foundation for future developments in quantization approaches tailored to Transformers, opening the door to efficiently train large-scale models from scratch for improved accessibility.

## Limitation

We acknowledge the limitations of our work:

- We employed linear quantization for our experiments. While this approach is widely used and allows for a controlled study, it may not capture the full potential of more sophisticated quantization methods.
- Due to the cost of pre-training and the number of experiments, we limited our study to GPT-2 small, and our findings may not generalize to larger models.
- The efficiency gains discussed are estimated using profiling data, and implementing these improvements in practice is challenging due to the complexity of kernel optimizations.

## References


Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. 2024. Quarot: Outlier-free 4-bit inference in rotated llms. *arXiv preprint arXiv:2404.00456*.

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

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2014. One billion word benchmark for measuring progress in statistical language modeling. *Preprint*, arXiv:1312.3005.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 2021. 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*.

Tim Dettmers and Luke Zettlemoyer. 2023. The case for 4-bit precision: k-bit inference scaling laws. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. 2021. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. GPTQ: Accurate post-training compression for generative pretrained transformers. *arXiv preprint arXiv:2210.17323*.

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

Aaron Gokaslan and Vanya Cohen. 2019. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2021. I-bert: Integer-only bert quantization. In *International conference on machine learning*, pages 5506–5518. PMLR.

Bingrui Li, Jianfei Chen, and Jun Zhu. 2024. Memory efficient optimizers with 4-bit states. *Advances in Neural Information Processing Systems*, 36.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. 2018. Visualizing the loss landscape of neural nets.

Zhikai Li, Xiaoxuan Liu, Banghua Zhu, Zhen Dong, Qingyi Gu, and Kurt Keutzer. 2023. Qft: Quantized full-parameter tuning of llms with affordable resources. *arXiv preprint arXiv:2310.07147*.

Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. 2023. Estimating the carbon footprint of bloom, a 176b parameter language model. *Journal of Machine Learning Research*, 24(253):1–15.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Ilia Markov, Adrian Vladu, Qi Guo, and Dan Alistarh. 2023. Quantized distributed training of large models with convergence guarantees. *arXiv preprint arXiv:2302.02390*.

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

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

Markus Nagel, Marios Fournarakis, Yelysei Bondarenko, and Tijmen Blankevoort. 2022. Overcoming oscillations in quantization-aware training. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 16318–16330. PMLR.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernandez. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, Berlin, Germany. Association for Computational Linguistics.

David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. 2021. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*.

Houwen Peng, Kan Wu, Yixuan Wei, Guoshuai Zhao, Yuxiang Yang, Ze Liu, Yifan Xiong, Ziyue Yang, Bolin Ni, Jingcheng Hu, et al. 2023. Fp8-lm: Training fp8 large language models. *arXiv preprint arXiv:2310.18313*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey.(2020). *arXiv preprint cs.LG/2009.06732*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Mitchell Wortsman, Tim Dettmers, Luke Zettlemoyer, Ari Morcos, Ali Farhadi, and Ludwig Schmidt. 2024. Stable and low-precision training for large-scale vision-language models. *Advances in Neural Information Processing Systems*, 36.

Haocheng Xi, Yuxiang Chen, Kang Zhao, Kaijun Zheng, Jianfei Chen, and Jun Zhu. 2024a. Jetfire: Efficient and accurate transformer pretraining with int8 data flow and per-block quantization. *arXiv preprint arXiv:2403.12422*.

Haocheng Xi, Changhao Li, Jianfei Chen, and Jun Zhu. 2024b. Training transformers with 4-bit integers. *Advances in Neural Information Processing Systems*, 36.

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.

Vikas Yadav, Steven Bethard, and Mihai Surdeanu. 2019. Quick and (not so) dirty: Unsupervised selection of justification sentences for multi-hop question answering. *arXiv preprint arXiv:1911.07176*.

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

# A Experimental Setup

## A.1 Model and Training Configuration

Our experiments leverage the computational enhancements of the FlashAttention library (Dao et al., 2022), utilizing its GPT-2 implementation within the HuggingFace Trainer framework [2] for our training processes.

Training was conducted on the OpenWebText corpus (Gokaslan and Cohen, 2019), adopting a set of training configurations similar to (Dao et al., 2022) and nanoGPT [3], without hyperparameter tuning due to computational constraints. We use AdamW optimizer with a learning rate of $6e-4$, combined with a cosine learning rate scheduler set to a half cycle. We adopt mixed precision training in bfloat16. The experiments are conducted with a fixed batch size of 512, employing gradient accumulation as necessary to accommodate the computational constraints of our setup across 4xA100 80G GPUs. This training configuration remains consistent across all experiments, culminating in a training duration averaging 4.3 days for completion of 300k steps.

## A.2 Data and Evaluation Metrics

The OpenWebText corpus was randomly split into training and validation sets, with 0.5% of the data reserved for validation. Following the methodology of Radford et al. (2019), the performance of our pretrained models was evaluated on several benchmark datasets. These evaluations focused on perplexity

[2] https://huggingface.co/docs/transformers/en/main_classes/trainer

[3] https://github.com/karpathy/nanoGPT

measurements on well-known datasets including WikiText (Merity et al., 2016), PTB (Marcus et al., 1993), and 1BW (Chelba et al., 2014). Our models were further evaluated using a range of downstream tasks:

- GLUE (Wang et al., 2018): A collection of natural language understanding tasks including question answering, sentiment analysis, and textual entailment, designed to benchmark the generalization capabilities of models across a diverse range of linguistic challenges.
- ARC (Yadav et al., 2019): Comprising the ARC-Easy and ARC-Challenge, these datasets test the model's reasoning ability through science-based question answering. ARC-Easy contains simpler questions, while ARC-Challenge includes more complex queries demanding deeper reasoning.
- Hellaswag (Zellers et al., 2019): This dataset challenges models to predict the most plausible continuation of a narrative from a large corpus of everyday contexts and movie scripts, testing the commonsense reasoning ability of the models.
- LAMBADA (Gokaslan and Cohen, 2019): Evaluates the model's capability to predict the final word of a textual passage, focusing on the contextual understanding of the language.

For evaluation on downstream tasks, we adopted a few-shot approach, utilizing a 5-shot prompting method with greedy decoding. Each prompt was structured with a task instruction followed by the five examples of training data and we compute accuracy on validation set. We implemented our few-shot evaluation protocol following the guidelines provided by the lm_evaluation_harness library [4].

### A.3 Comparison of Baseline and Pre-trained Model

Table 1: Comparison of Baseline and Pre-trained Models: The latter trained for at least twice the duration.

| | LAMBADA (accuracy) | LAMBADA (ppl) | WikiText2 (ppl) | WikiText103 (ppl) | PTB (ppl) | 1BW (ppl) |
|---|---|---|---|---|---|---|
| Pre-trained model | 33.3 | 38.48 | 24.67 | 29.17 | 35.86 | 45.87 |
| Baseline | 35.97 | 34.80 | 34.32 | 39.94 | 35.13 | 44.03 |

[4] https://github.com/zphang/lm_evaluation_harness

To establish a solid baseline for our experiments, we benchmark our trained model against the pre-trained GPT-2 weights provided by OpenAI across several downstream tasks. The comparison, detailed in Table 1, reveals that our model achieves results closely aligned with the original, validating the efficacy of our training approach. In the following sections, we will delve into the individual components of the model, discussing the impact and outcomes of our quantization experiments on each.

## B Memory Analysis

In this section, we explore the memory consumption patterns of various components within GPT-2 models during training, using the PyTorch Memory Profiler. This profiling tool allows for precise monitoring of memory usage throughout the lifecycle of a model's operation, particularly during the forward and backward passes, and during optimization steps. "Peak memory" in this context refers to the maximum memory usage observed at any point in these stages, providing insights into how different model configurations impact overall memory requirements. We profiled all attention blocks as well as lm head of the Transformer model.

Figure 14 shows how memory usage changes with different batch sizes for a fixed context length of 1024, across various model sizes. Notably, as the batch size increases, the memory allocated for activations becomes more dominant, especially in larger models. This trend is primarily due to the need to store activations for the computation of gradients during the backward pass, which increases with larger batch sizes.

Figure 15 examines the memory usage across various sequence lengths while keeping the batch size constant at 4. When both batch size and sequence length are small, peak memory typically occurs towards the end of the backward propagation phase. At this stage, memory includes the parameters, optimizer states, gradients from all layers, and activations from the initial layers. However, as the sequence length and batch size increase, peak memory usage shifts to the beginning of the backward propagation. At this point, the memory comprises parameters, optimizer states, all activations, and notably, the output gradient of the final layers, which matches the size of the logits (proportional to batch size * sequence length * vocabulary size).

In conclusion, the analysis reveals that when a

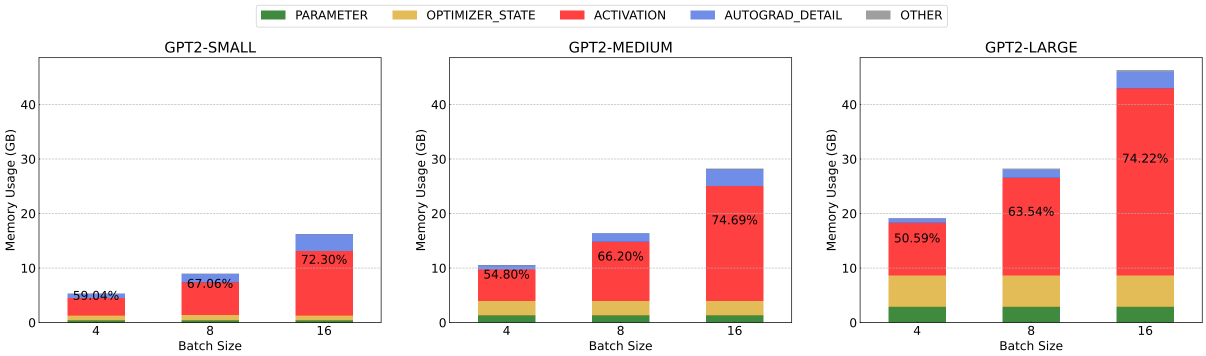


Figure 14: Distribution of peak memory usage across different model sizes (GPT-2 Small, Medium, and Large) for a constant context length of 1024, with varying batch sizes. This figure demonstrates how memory dedicated to activations increases as batch size increases, highlighting the impact of batch size on memory allocation dynamics in large-scale models.

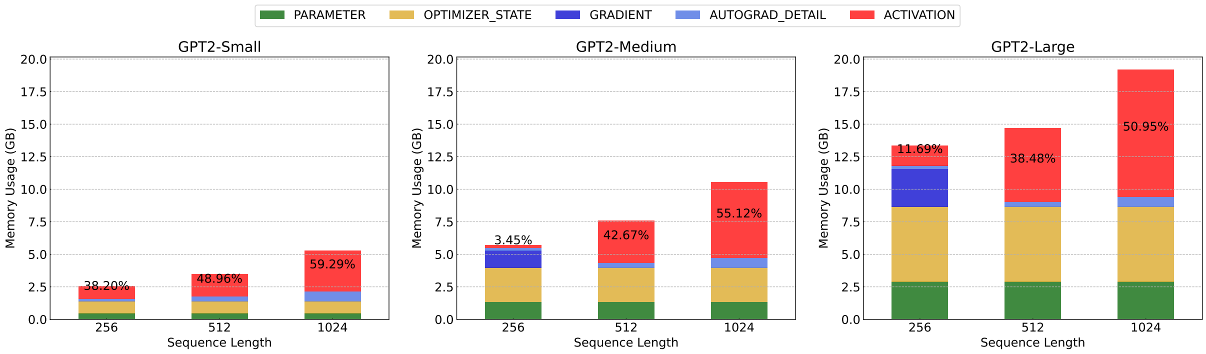


Figure 15: Peak memory usage profile for different sequence lengths while maintaining a constant batch size of 4, across different model sizes (GPT-2 Small, Medium, and Large). This figure illustrates the shift in peak memory usage from gradients to activations as sequence length increases, and the significant impact of sequence length on memory dynamics during training.

model can fit within the GPU memory, the majority of the memory at peak times is consumed by activations, especially when working with sufficiently large batch sizes and sequence lengths. Under these conditions, gradients do not contribute to peak memory usage. Consequently, quantizing gradients will not lead to significant memory savings.

## C Quantization Results

This appendix presents the granular results from our experiments on weight, activation, gradient, and optimizer states quantization. The tables detail the performance metrics, like perplexity and accuracy, under various quantization settings. Finally, we present post-training quantization results of our baseline model in Tables 10,11.

Table 2: **Weight Quantization**: evaluation of perplexity across multiple datasets.

| #bit | granularity | WikiText103 | WikiText2 | PTB | 1BW |
|---|---|---|---|---|---|
| baseline | | 39.94 | 34.32 | 35.13 | 44.03 |
| 4 bit | per-tensor | 55.50 | 46.70 | 52.38 | 59.14 |
| | per-channel | 56.43 | 47.32 | 38.18 | 46.30 |
| 8 bit | per-tensor | 48.52 | 40.01 | 37.02 | 45.04 |
| | per-channel | 42.43 | 35.94 | 34.81 | 43.47 |

Table 3: **Activation Quantization**: evaluation of perplexity across multiple datasets.

| #bit | granularity | WikiText103 | WikiText2 | PTB | 1BW |
|---|---|---|---|---|---|
| baseline | | 39.94 | 34.32 | 35.13 | 44.03 |
| 4 bit | per-tensor | 418.63 | 264.07 | 261.64 | 310.43 |
| | per-token | 69.82 | 54.53 | 56.58 | 72.06 |
| 8 bit | per-tensor | 64.77 | 51.06 | 37.96 | 45.26 |
| | per-token | 42.86 | 37.17 | 35.43 | 43.38 |

Table 4: **Gradient Quantization**: evaluation of perplexity across multiple datasets.

| #bit | granularity | WikiText103 | WikiText2 | PTB | 1BW |
|---|---|---|---|---|---|
| baseline | | 39.94 | 34.32 | 35.13 | 44.03 |
| 4 bit | per-tensor | 17990.70 | 15560.03 | 6632.20 | 6393.07 |
| | per-token | 128.71 | 92.74 | 106.73 | 110.14 |
| 8 bit | per-tensor | 123.08 | 87.81 | 104.90 | 111.51 |
| | per-token | 59.24 | 47.50 | 42.28 | 51.89 |

Table 5: **Adam Optimizer's First Moments**: evaluation of perplexity across multiple datasets.

| #bit | granularity | WikiText103 | WikiText2 | PTB | 1BW |
|---|---|---|---|---|---|
| baseline | | 39.94 | 34.32 | 35.13 | 44.03 |
| 4 bit | per-tensor | 78.78 | 62.08 | 66.50 | 85.60 |
| | per-channel | 43.02 | 36.70 | 38.57 | 47.90 |
| 8 bit | per-tensor | 42.93 | 36.91 | 39.72 | 46.63 |
| | per-channel | 39.84 | 33.78 | 35.67 | 44.29 |

Table 6: **Weight Quantization**: accuracy on downstream tasks.

| | | GLUE Score | | | | | | ARC | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # of bit | granularity | MNLI | MRPC | RTE | QNLI | SST | WNLI | Easy | Challenge | LAMBADA | Hellaswag | **Average** |
| baseline | | 33.31 | 64.46 | 49.82 | 49.13 | 52.06 | 50.70 | 46.14 | 22.07 | 36.17 | 29.12 | 43.30 |
| 4 bit | per-tensor | 31.76 | 50.49 | 45.49 | 49.13 | 50.46 | 36.62 | 40.70 | 19.40 | 27.03 | 27.25 | 37.83 |
| | per-channel | 33.51 | 59.07 | 50.90 | 49.70 | 52.29 | 56.34 | 45.09 | 22.07 | 34.21 | 28.86 | 43.20 |
| 8 bit | per-tensor | 33.12 | 59.07 | 46.57 | 48.95 | 54.24 | 49.30 | 46.84 | 21.40 | 34.81 | 28.80 | 42.31 |
| | per-channel | 34.78 | 67.16 | 53.07 | 49.42 | 54.01 | 50.70 | 44.21 | 22.07 | 36.43 | 29.17 | 44.10 |

Table 7: **Activation Quantization**: accuracy on downstream tasks.

| | | GLUE Score | | | | | | ARC | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # of bit | granularity | MNLI | MRPC | RTE | QNLI | SST | WNLI | Easy | Challenge | LAMBADA | Hellaswag | **Average** |
| baseline | | 33.31 | 64.46 | 49.82 | 49.13 | 52.06 | 50.70 | 46.14 | 22.07 | 36.17 | 29.12 | 43.30 |
| 4 bit | per-token | 34.75 | 58.09 | 46.93 | 49.20 | 49.20 | 50.70 | 39.12 | 18.73 | 18.88 | 27.24 | 39.29 |
| | per-token asymmetric | 34.12 | 57.35 | 49.82 | 49.28 | 50.00 | 52.11 | 38.95 | 17.06 | 22.30 | 27.03 | 39.80 |
| 4 bit | per-tensor | 34.73 | 31.62 | 56.32 | 49.42 | 49.66 | 59.15 | 32.28 | 19.40 | 1.53 | 26.04 | 36.02 |
| 8 bit | per-token | 34.08 | 61.03 | 51.62 | 50.16 | 55.85 | 59.15 | 44.74 | 19.73 | 36.46 | 29.33 | 44.22 |
| | per-tensor | 32.84 | 54.90 | 48.74 | 49.15 | 52.98 | 57.75 | 45.44 | 21.74 | 34.33 | 28.77 | 42.66 |

Table 8: **Gradient Quantization**: accuracy on downstream tasks.

| # of bit | granularity | GLUE Score MNLI | GLUE Score MRPC | GLUE Score RTE | GLUE Score QNLI | GLUE Score SST | GLUE Score WNLI | ARC Easy | ARC Challenge | LAMBADA | Hellaswag | **Average** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| baseline | | 33.31 | 64.46 | 49.82 | 49.13 | 52.06 | 50.70 | 46.14 | 22.07 | 36.17 | 29.12 | 43.30 |
| 4 bit | per-token | 34.19 | 35.29 | 49.10 | 49.46 | 49.54 | 49.30 | 33.68 | 19.40 | 19.06 | 26.11 | 36.51 |
| 8 bit | per-token | 33.35 | 42.89 | 52.71 | 49.42 | 50.80 | 25.35 | 45.09 | 18.39 | 33.81 | 28.81 | 38.06 |
| | per-tensor | 35.25 | 31.13 | 45.85 | 49.48 | 50.34 | 54.93 | 33.33 | 20.40 | 18.22 | 26.15 | 36.51 |

Table 9: **Quantization of Adam Optimizer's First-Order Moments**: accuracy on downstream tasks.

| # of bit | granularity | GLUE Score MNLI | GLUE Score MRPC | GLUE Score RTE | GLUE Score QNLI | GLUE Score SST | GLUE Score WNLI | ARC Easy | ARC Challenge | LAMBADA | Hellaswag | **Average** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| baseline | | 33.31 | 64.46 | 49.82 | 49.13 | 52.06 | 50.70 | 46.14 | 22.07 | 36.17 | 29.12 | 43.30 |
| 4 bit | per-channel | 33.98 | 66.42 | 52.35 | 49.46 | 51.03 | 53.52 | 46.67 | 21.07 | 33.15 | 28.47 | 43.61 |
| | per-tensor | 32.36 | 66.42 | 46.93 | 50.32 | 50.92 | 42.25 | 35.44 | 22.07 | 20.12 | 26.49 | 39.33 |
| 8 bit | per-channel | 33.25 | 67.40 | 50.18 | 49.48 | 52.87 | 50.70 | 46.67 | 20.40 | 36.52 | 28.91 | 43.64 |
| | per-tensor | 34.15 | 61.76 | 50.18 | 49.53 | 54.70 | 60.56 | 45.61 | 18.39 | 32.91 | 28.80 | 43.66 |

Table 10: Post-training weight quantization results

| #bit | granularity | WikiText103 (ppl) | WikiText2 (ppl) | PTB (ppl) | 1BW (ppl) |
|---|---|---|---|---|---|
| baseline | | 39.94 | 34.32 | 35.13 | 44.03 |
| 4 bit | per-tensor | 16196.10 | 17256.89 | 17471.35 | 13761.79 |
| | per-column | 98.39 | 75.56 | 81.28 | 94.40 |
| 8 bit | per-tensor | 46.45 | 39.23 | 41.18 | 52.15 |
| | per-column | 40.15 | 34.45 | 35.23 | 44.11 |

Table 11: Post-training activation quantization results

| #bit | granularity | WikiText103 (ppl) | WikiText2 (ppl) | PTB (ppl) | 1BW (ppl) |
|---|---|---|---|---|---|
| baseline | | 39.94 | 34.32 | 35.13 | 44.03 |
| 4 bit | per-tensor | - | - | - | - |
| | per-column | 14022.78 | 17933.29 | 13392.28 | 8763.06 |
| 8 bit | per-tensor | 70.07 | 58.45 | 64.99 | 149.35 |
| | per-column | 40.09 | 34.44 | 35.43 | 44.37 |