Q-Adam-mini: Memory-Efficient 8-bit Quantized Optimizer for Large Language Model Training

Yizhou Han [*1,2] **Chaohao Yang** [*1] **Xingjian Wang** [1,2] **Congliang Chen** [1,2]
Ruoyu Sun [1,2]

Abstract

We propose **Q-Adam-mini**, a memory-efficient optimizer for Large Language Model (LLM) training that achieves **8**× reduction in GPU memory usage while maintaining performance parity with full-precision AdamW. Building upon Adam-mini (Zhang et al., 2024b), which reduces memory footprint of optimizer states by 50% compared to AdamW, we further improve memory efficiency through states quantization. We achieve this by: (i) quantizing the first-order momentum (m) to **INT8** and (ii) retaining the second-order momentum (v) in **FP32**, which occupies less than 1% of total memory. However, embedding layer exhibits weight norm instability. We analyze this issue and address it by applying stochastic rounding for momentum quantization exclusively to the embedding layer. We validate our approach on both pre-training and fine-tuning tasks, with the model size ranging from 60M to 8B. Our results demonstrate that Q-Adam-mini enables scalable LLM training with limited computational resources. Codes are available at `https://github.com/LouisCroix/Q-Adam-mini`

1. Introduction

In recent years, large language models (LLMs) (Radford et al., 2018; Grattafiori et al., 2024; Team et al., 2023; Liu et al., 2024) have demonstrated remarkable capabilities across various natural language processing (NLP) tasks, including text generation, dialogue, sentiment analysis, and reasoning (Brown et al., 2020; Kocoń et al., 2023; Zhang et al., 2024c; Koroteev, 2021). However, modern LLMs typically consist of billions or even trillions of parameters, leading to extremely high memory requirements during training. A key factor contributing to this challenge is the usage of AdamW (Kingma & Ba, 2014; Loshchilov & Hutter, 2017), the most widely adopted optimizer for training LLMs. AdamW requires additional storage for both the first-order momentum m and the second-order momentum v for each parameter, significantly increasing GPU memory consumption.

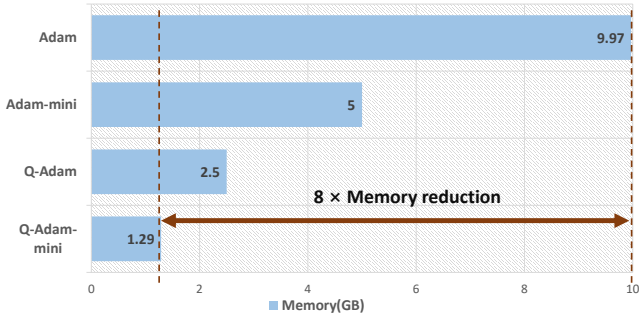


Figure 1. Optimizer states' memory consumption for pre-training Llama-1B

For example, memory usage for training a Llama 7B model using full precision AdamW is estimated as follows:

$$\text{Memory Usage} = \underbrace{28\text{GB}}_{\text{Weight}} + \underbrace{28\text{GB}}_{m} + \underbrace{28\text{GB}}_{v} = 84\text{GB}.$$

The enormous memory consumption prevents researchers from studying LLMs. In response to this challenge, many studies have aimed to reduce the memory footprint of optimizers. Among these, Adam-mini (Zhang et al., 2024b) achieves great improvement by enabling parameter groups to share learning rate (i.e., $1/\sqrt{v}$) while maintaining comparable performance. Specifically, Adam-mini groups parameters into blocks according to predefined rules and assigns a single second order momentum value v to all parameters within each block. This approach reduces memory consumption by approximately 50% compared to AdamW. Nonetheless, in Adam-mini, all optimizer states are stored in `FP32` format. As a result, for a model like Llama 7B, the optimizer states alone require more than 28 GB of memory,

[*]Equal contribution [1]The Chinese University of Hong Kong, Shenzhen, China [2]Shenzhen Research Institute of Big Data. Correspondence to: Ruoyu Sun <sunruoyu@cuhk.edu.cn>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

which remains a substantial challenge.

To address this issue, we propose Quantized Adam-mini (Q-Adam-mini), which leverages quantization techniques (Gray & Neuhoff, 1998) to further reduce memory and computational requirements.

Our approach makes four key contributions:

- We develop a hybrid precision scheme that maintains second-order momentum in full FP32 while quantizing first-order momentum to INT8, capitalizing on their different sensitivity to precision reduction.
- We demonstrate that quantization errors in first-order momentum will accumulate in embedding layer weights, ultimately leading to explosive growth in weight norms.
- We introduce stochastic rounding technique that alleviates this instability while adding negligible computational overhead.
- Through extensive experiments on models ranging from 60M to 8B parameters, we demonstrate that Q-Adam-mini reduces optimizer memory usage to just 1/8 of AdamW's requirements (Figure 1) while matching its performance on both pre-training and fine-tuning tasks.

2. Methodology

In Section 2.1, we introduce quantization basics and Adam-mini's design principles. Section 2.2 presents a precision selection scheme for the momentum terms. However, in later training stages, Q-Adam-mini underperforms original Adam-mini. For this issue, we provide empirical analysis in section 2.3.1 and propose a stochastic rounding approach to stabilize training dynamics in section 2.3.2.

2.1. Preliminaries on Quantization and Adam-mini

2.1.1. Quantization

Quantization enables efficient computation and reduced memory usage, which is crucial for deploying models on resource-constrained devices. In Q-Adam-mini, the **optimizer states** are saved in INT8, model weights and gradients are saved in FP32, activations are computed in BF16. We use INT8 quantization because it is universally supported, whereas FP8 requires advanced hardware support (e.g., Hopper GPUs (Elster & Haugdahl, 2022)) that are not yet widely adopted.

To convert data precision from FP32 to INT8, we use block-wise quantization as:

$$\text{Quant}(X) = \text{clamp}(\text{round}(\frac{X}{\Delta}) + z, -2^{n-1}, 2^{n-1} - 1), \quad (1)$$

where Δ and z are scaling factor (FP32) and zero-point (INT8) respectively, which are calculated within each block of the tensors.

2.1.2. Key Designs of Adam-mini

Adam-mini partitions parameters into groups following some predefined rules and assigns a shared (calculated as average) second-order momentum to each group, i.e.

$$\begin{cases} v = m = \mathbf{0} & \text{Initialization} \\ m = \beta_1 m + (1-\beta_1)\, g & \text{State 1 update} \\ g^2_{\text{mean}} = \text{Average}(g^2) & \text{Calculate Average} \\ v_{\text{mean}} = \beta_2 v_{\text{mean}} + (1-\beta_2)\, g^2_{\text{mean}} & \text{State 2 update} \\ w_t = w_{t-1} - \alpha \cdot \frac{m_t}{\sqrt{v_{\text{mean}}}+\epsilon} & \text{Weight update} \end{cases}$$

Adam-mini uniquely incorporates a "Calculate Average" step, using the averaged gradient to update second order momentum v_{mean}.

2.2. Precision Selection of Optimizer States

The precision selection for Adam-mini's momentum requires balancing memory efficiency and convergence. Following Q-Adam (Dettmers et al., 2021), which shows first-order momentum m exhibits strong quantization robustness while second-order momentum v is sensitive, we employ INT8 for m and evaluate two configurations for v_{mean}.

• **Case 1:** 8bit m + 8bit v_{mean}

The quantization of m follows the approach described in (Dettmers et al., 2021). For the second-order momentum v_{mean}, since we have adopted asymmetric quantization (1), to ensure computational efficiency, we first compute its average value and then apply lower-bit quantization. The quantization and update pipeline is illustrated in Figure 2.

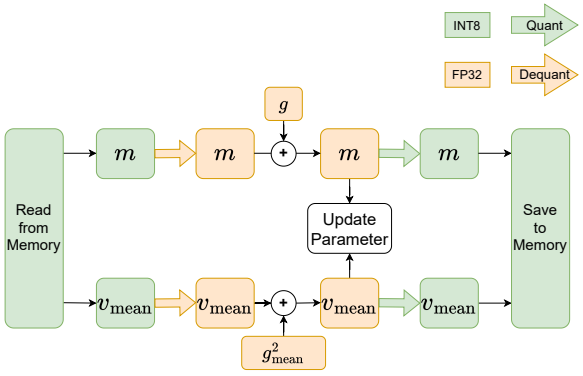


Figure 2. The pipeline of optimizer states quantization and update in one iteration for Case 1. INT8 data (green) remains permanently stored, whereas the FP32 data (yellow) are deleted right after parameter updates.

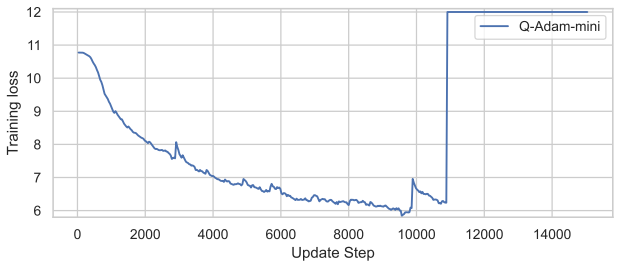


Figure 3. Adam-mini equipped with 8bit m and 8bit v_{mean}. It suffers loss spike after 10K iterations when training Llama 1B.

Our training monitoring revealed a characteristic loss spike occurring at approximately 10K iterations, followed by complete divergence of the optimization process. Training loss curve of Llama 1B is shown in Figure 3.

• **Case 2:** 8bit m + 32bit v_{mean}

In this case we maintain full-precision (FP32) representation for the second-order momentum v_{mean} throughout the optimization process. One might consider the memory overhead of v_{mean}; however, Adam-mini reduces its memory footprint to below 1%. We validate this through pre-training experiments on Llama 1B and 7B models (see Table 4 in Appendix). Consequently, the full-precision storage of v_{mean} imposes practically no additional memory burden. We show the update pipeline and training loss curve of Llama 1B respectively in Figure 4 and Figure 5. Under the configuration

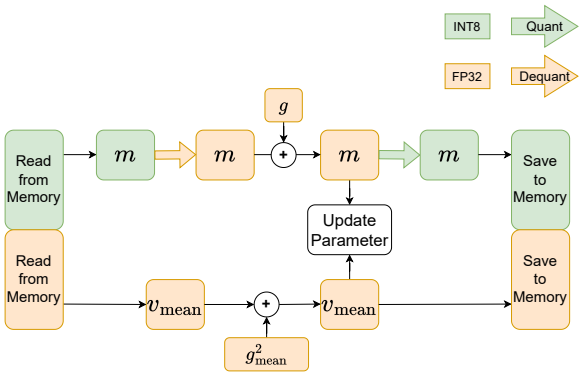


Figure 4. The update pipeline of Adam-mini for Case 2. In this case, we keep v_{mean} in FP32.

of Case 2, the training process remains stable. However, after exceeding 40K update iterations, the convergence rate of Q-Adam-mini exhibits a significant decrease compared to Adam-mini. This naturally leads to the question:

Question1: What slows down the convergence rate?

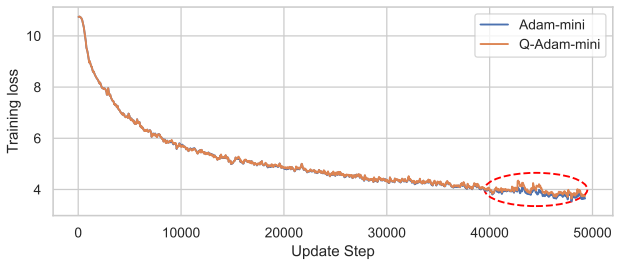


Figure 5. Adam-mini equipped with 8bit m and 32bit v_{mean}. The training process remains stable but convergence rate of Q-Adam-mini slows down after 40K iterations.

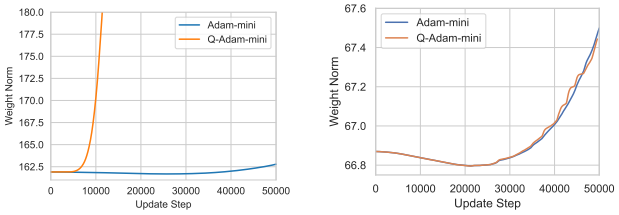
2.3. Weight Norm Explosion in Embedding Layer

To answer the question 2.2, firstly, in section 2.3.1 we perform a comparison of weight norm patterns across Q-Adam-mini and Adam-mini. Building upon these observations, in section 2.3.2 we propose a stochastic rounding strategy only for embedding layer. This approach not only maintains training efficiency but also enables Q-Adam-mini to achieve comparable convergence performance to full-precision optimizers throughout the entire training process.

2.3.1. Weight Norm of Embedding Layer Suffers Instability

To investigate why convergence rate of Q-Adam-mini slows down in the later training stage, we first monitored the weight norm across all model layers. This metric provides preliminary insights into potential anomalies in Q-Adam-mini's training behavior. Through experimental observations, we draw the following point:

- Q-Adam-mini has significant weight norm deviations in the embedding layer (Figure 6a), but not in the other layers (Figure 6b).



(a) Weight norm of embedding layer

(b) Weight norm of mlp down project in 10th layer

Figure 6. In the embedding layer (left), the weight norm begins to explode after 10K steps, whereas in other layers such as the 10th MLP layer(right), Q-Adam-mini and Adam-mini demonstrate nearly identical optimization properties.

In Q-Adam-mini, parameter updates are jointly determined by m and v_{mean}. To identify the factors contributing to the anomalous behavior in weight norm, we simultaneously monitored the norm of both optimizer states. The results show that:

- In the embedding layer, both the m norm (Figure 7a) and v_{mean} norm (Figure 7b) remain stable. Compared to Adam-mini, no significant difference is observed.

The results demonstrate that while Q-Adam-mini's optimizer states remain stable throughout training, the embedding layer exhibits explosive growth in weight norm. Based on the above observations, we raise the following question:

Question2: How explosion occurs in Embedding layer?

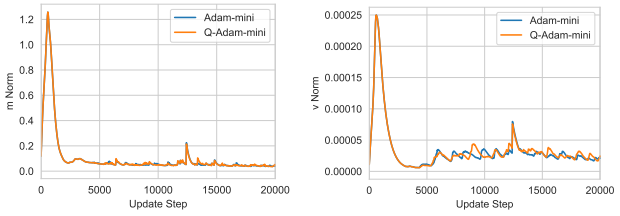
To formalize this phenomenon, we consider an LLM $f_{\theta,\hat{\theta}}$ where θ denotes embedding layer parameters and $\hat{\theta}$ represents other trainable parameters. During next-token prediction training, only a sparse subset $\theta_0 \subset \theta$ (corresponding to active input tokens X_{prompt}) receives non-zero gradients.

This sparsity induces critical differences between optimizer behaviors:

- *Non-momentum optimizers* (e.g., SGD): Parameters with zero gradients remain unchanged.
- *Momentum-based optimizers* (e.g., Adam): Historical momentum drives parameter updates even when $g_t = 0$ since:

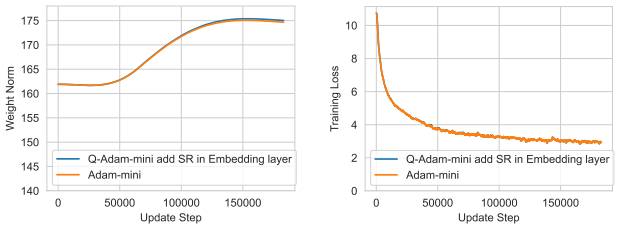
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_{t-1}$$

Now let $m^{\theta}, m^{\hat{\theta}}$ represents the momentum corresponding to θ and $\hat{\theta}$ respectively. Crucially, $m^{\hat{\theta}}$ benefit from gradient



(a) m norm of embedding layer (b) v norm of embedding layer

Figure 7. We specifically monitored the norm of m and v_{mean} in both Q-Adam-mini and Adam-mini optimizers during the first 20K steps. The empirical evidence indicates that although the momentum term m is stored in low precision during updates - which introduces quantization errors - its norm exhibits neither spikes nor explosion phenomena that could potentially affect weight norm.



(a) Weight norm of embedding layer (b) Training Loss Curve for 180K iterations

Figure 8. Implementing stochastic rounding exclusively in the embedding layer (left) maintains stable weight norm during training, while (right) enabling Q-Adam-mini to match the performance of full-precision optimizers in extended training sessions.

corrections that mitigate quantization errors in each iteration. However, the quantization error in m^{θ} will persist until the momentum receives a non-zero gradient update. Throughout this period, the momentum m^{θ}, now carrying quantization error, is utilized to update the parameters at each iteration, leading to progressive accumulation of quantization errors in the parameters.

Note that error does not accumulate in the momentum itself; rather, the erroneous momentum continuously update the parameters **in same direction**, which cause the error accumulation in parameters, ultimately manifesting as an explosion in weight norm, while the m norm remains stable.

2.3.2. Reducing Computational Error via Stochastic Rounding

To prevent the accumulation of errors in the same direction, we need to introduce randomness so that the quantization error does not always remain in one direction. Specifically, assuming the quantization function after introducing randomness is $\hat{Q}$, we hope:

1. For momentum m, $\hat{Q}(m) \in \{\lfloor m \rfloor, \lceil m \rceil\}$ is a random variable.
2. The quantization error does not increase, i.e.,

$$\left| \mathbb{E}\left[\hat{Q}(x)\right] - x \right| \leq |Q(x) - x|$$

for all x and deterministic quantization $Q(\cdot)$.

Following above two principles, we introduce stochastic rounding(SR), formulated as follows:

$$\hat{Q}(W) = \begin{cases} \lfloor W \rfloor & \text{with probability } p = \lceil W \rceil - W \\ \lceil W \rceil & \text{with probability } p = W - \lfloor W \rfloor \end{cases}$$

Stochastic rounding introduces randomness to the quantization process, and we can prove that the quantized value $\hat{Q}(W)$ satisfies :

$$\mathbb{E}[\hat{Q}(W) - W] = 0 \leq W.$$

Detailed proof is provided in the Appendix C.

Although stochastic rounding is a lossless quantization scheme, its substantially higher computational latency (over 80% compared to deterministic rounding; see Appendix D) makes it infeasible for all network layers. To balance stability and computational efficiency, we exclusively employ it in the embedding layer. Experimental results demonstrate that this approach effectively maintains Q-Adam-mini's convergence rate and optimization stability.

3. Experiments

We now validate the effectiveness of Q-Adam-mini in both pre-training and fine-tuning tasks. Detailed experimental setups are provided in Appendix E.

3.1. Pre-training

We trained LLaMA-based models from scratch using C4 dataset (Patel, 2020) with different optimizers. We report the loss on the validation dataset after training, along with the GPU memory usage. A theoretical analysis of memory consumption can be found in Section 3.3.

Table 1. Pretraining Results on C4 Dataset. We trained the Llama-series models with sizes ranging from 60M to 1B from scratch. For each result, we report the validation loss and the GPU memory usage of the optimizer states. All optimizers use 32-bit precision, except for Q-Adam and Q-Adam-mini.

Methods	60M		130M	
	Val. Loss	Memory	Val. Loss	Memory
AdamW	3.34	0.43G	3.07	1.0G
Q-AdamW	3.35	0.11G	3.08	0.26G
AdamMini	3.37	0.22G	3.08	0.51G
Q-AdamMini	3.38	**0.06G**	3.09	**0.13G**

Methods	350M		1B	
	Val. Loss	Memory	Val. Loss	Memory
AdamW	2.71	2.74G	2.55	9.97G
Q-AdamW	2.71	0.69G	2.55	2.50G
AdamMini	2.73	1.36G	2.55	5.0G
Q-AdamMini	2.73	**0.35G**	2.55	**1.29G**

3.2. Supervised Fine-tuning

In supervised fine-tuning, we fine-tune the pre-trained Llama3 8B model on two multi-task benchmarks: MMLU (Hendrycks et al., 2020) and GSM-8K (Cobbe et al., 2021). Experimental results show our Q-Adam-mini optimizer matches the performance of full-precision AdamW and Adam-mini. We evaluate using zero-shot fine-tuning, with results in Table 2. On MMLU, Q-Adam-mini outperforms full-precision optimizers, and on GSM-8K, it surpasses both Q-AdamW and LoRA.

Table 2. Zero-shot accuracy for Supervised Fine-tuning

Methods	MMLU	GSM-8K
AdamW	63.37%	56.86%
Q-AdamW	62.71%	55.57%
AdamMini	62.37%	56.56%
LoRA	61.64%	55.72%
Q-AdamMini	63.50%	56.03%

3.3. Memory Measurement

We perform the theoretical analysis of memory usage when using Q-Adam-mini training LLMs. These results demonstrate that our method can significantly improve memory efficiency compared to the current mainstream optimizers.

Table 3. To theoretically compare the memory footprint of optimizer states, we consider a parameter matrix of size $m \times n$ stored in FP32 format. When employing LoRA, the original matrix is decomposed into two low-rank matrices of dimensions $m \times r$ and $r \times n$ respectively ($r \ll \min(m, n)$).

Methods	Weights	Optimizer states
AdamW	$4mn$	$8mn$
Q-AdamW	$4mn$	$2mn$
AdamMini	$4mn$	$4mn + \mathcal{O}(1)$
LoRA	$4(mn + mr + nr)$	$8(mr + nr)$
Q-AdamMini	$4mn$	$mn + \mathcal{O}(1)$

4. Conclusion

We propose **Q-Adam-mini**, a memory-efficient optimizer that reduces GPU memory usage by 8× while matching the performance of full-precision AdamW. By strategically quantizing first-order momentum to INT8 and preserving second-order momentum in FP32, our approach maintains training stability. The introduction of stochastic rounding specifically addresses quantization error accumulation in embedding layers, enabling robust convergence. Extensive experiments on models ranging from 60M to 8B parameters demonstrate that Q-Adam-mini achieves memory efficiency without compromising model performance.

Acknowledgements

The work of Ruoyu Sun wa supported by NSFC (No. 12326608); Hetao Shenzhen-Hong Kong Science and Technology Innovation Cooperation Zone Project (No.HZQSWS-KCCYB-2024016); University Development Fund UDF01001491, the Chinese University of Hong Kong, Shenzhen; Guangdong Provincial Key Laboratory of Mathematical Foundations for Artificial Intelligence (2023B1212010001).

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Anil, R., Gupta, V., Koren, T., and Singer, Y. Memory efficient adaptive optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dettmers, T., Lewis, M., Shleifer, S., and Zettlemoyer, L. 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*, 2021.

Elster, A. C. and Haugdahl, T. A. Nvidia hopper gpu and grace cpu highlights. *Computing in Science & Engineering*, 24(2):95–100, 2022.

Fishman, M., Chmiel, B., Banner, R., and Soudry, D. Scaling fp8 training to trillion-token llms. *arXiv preprint arXiv:2409.12517*, 2024.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Gray, R. M. and Neuhoff, D. L. Quantization. *IEEE transactions on information theory*, 44(6):2325–2383, 1998.

Hao, Z., Guo, J., Shen, L., Luo, Y., Hu, H., Wang, G., Yu, D., Wen, Y., and Tao, D. Low-precision training of large language models: Methods, challenges, and opportunities. *arXiv preprint arXiv:2505.01043*, 2025.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kocoń, J., Cichecki, I., Kaszyca, O., Kochanek, M., Szydło, D., Baran, J., Bielaniewicz, J., Gruza, M., Janz, A., Kanclerz, K., et al. Chatgpt: Jack of all trades, master of none. *Information Fusion*, 99:101861, 2023.

Koroteev, M. V. Bert: a review of applications in natural language processing and understanding. *arXiv preprint arXiv:2103.11943*, 2021.

Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Luo, Y., Ren, X., Zheng, Z., Jiang, Z., Jiang, X., and You, Y. Came: Confidence-guided adaptive memory efficient optimization. *arXiv preprint arXiv:2307.02047*, 2023.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Patel, J. M. Introduction to common crawl datasets. In *Getting structured data from the internet: running web crawlers/scrapers on a big data production scale*, pp. 277–324. Springer, 2020.

Peng, H., Wu, K., Wei, Y., Zhao, G., Yang, Y., Liu, Z., Xiong, Y., Yang, Z., Ni, B., Hu, J., et al. Fp8-lm: Training fp8 large language models. *arXiv preprint arXiv:2310.18313*, 2023.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. Improving language understanding by generative pre-training. 2018.

Shazeer, N. and Stern, M. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pp. 4596–4604. PMLR, 2018.

Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Zhang, Y., Chen, C., Ding, T., Li, Z., Sun, R., and Luo, Z. Why transformers need adam: A hessian perspective. *Advances in Neural Information Processing Systems*, 37: 131786–131823, 2024a.

Zhang, Y., Chen, C., Li, Z., Ding, T., Wu, C., Kingma, D. P., Ye, Y., Luo, Z.-Q., and Sun, R. Adam-mini: Use fewer learning rates to gain more. *arXiv preprint arXiv:2406.16793*, 2024b.

Zhang, Y., Mao, S., Ge, T., Wang, X., de Wynter, A., Xia, Y., Wu, W., Song, T., Lan, M., and Wei, F. Llm as a mastermind: A survey of strategic reasoning with large language models. *arXiv preprint arXiv:2404.01230*, 2024c.

Zhang, Z., Jaiswal, A., Yin, L., Liu, S., Zhao, J., Tian, Y., and Wang, Z. Q-galore: Quantized galore with int4 projection and layer-adaptive low-rank gradients. *arXiv preprint arXiv:2407.08296*, 2024d.

A. Related Work

A.1. Low Precision Training

Low-precision training has become crucial for efficient deep learning, balancing memory savings with model performance (Hao et al., 2025). Early work introduced `FP16` training with loss scaling to address numerical limitations (Micikevicius et al., 2017). Recent advances explore `INT8` (Zhang et al., 2024d; Dettmers et al., 2021) and `FP8` quantization (Peng et al., 2023; Liu et al., 2024). Q-GaLore (Zhang et al., 2024d) and Q-Adam (Dettmers et al., 2021) reduce memory by quantizing optimizer states, while LM-FP8 (Peng et al., 2023) uses `FP8` gradients with mixed-precision states. DeepSeek V3 (Liu et al., 2024) accelerates training via `FP8` GEMM operations. (Fishman et al., 2024) enables full `FP8` training through redesigned activation functions, achieving comprehensive memory optimization for large language models.

A.2. Learning rate simplification methods

Adam has become the dominant choice for large language models due to its adaptive learning rate mechanism (Kingma & Ba, 2014). However, Adam's per-parameter learning rates incur substantial memory overhead, sparking research into reducing this cost without sacrificing performance. Existing approaches like Adafactor (Shazeer & Stern, 2018), its enhanced variant CAME (Luo et al., 2023), and SM3 (Anil et al., 2019) attempt to compress second-order moments through low-rank approximations or compact learning rate sets, but often with noticeable performance degradation. The recent Adam-mini optimizer (Zhang et al., 2024a) addresses this by exploiting Transformer networks' block-wise Hessian structure, introducing a parameter-block sharing mechanism where parameters within each block share identical second-order moments. This approach achieves a remarkable 99% reduction in memory footprint while maintaining performance comparable to standard Adam across multiple benchmarks.

B. Memory Usage of m and vmean in Adam-mini

During pre-training, we conducted end-to-end measurements of the GPU memory consumption for both m and v_{mean}. in Adam-mini, with the results presented below:

Table 4. Memory Usage of m and v_{mean} in Adam-mini

MODEL	m	v_{MEAN}
LLAMA 1B	1277.3MB	1.6MB
LLAMA 7B	6426.3MB	3.9MB

C. Proof of Stochastic Rounding

$$\begin{aligned}\mathbb{E}(\hat{Q}(W) - W) &= p_{\text{floor}} \cdot \lfloor W \rfloor + p_{\text{ceil}} \cdot \lceil W \rceil - W \\ &= (\lceil W \rceil - W) \cdot \lfloor W \rfloor + (W - \lfloor W \rfloor) \cdot \lceil W \rceil - W \\ &= W - W = 0.\end{aligned} \tag{2}$$

D. Latency Analysis of Stochastic Rounding

To evaluate the computational overhead of stochastic rounding, we conduct the following experiment:

Experimental Setup:

1. Generate a random matrix containing 1 million vectors, each with dimensionality 2048.
2. Apply both deterministic rounding and stochastic rounding to the matrix respectively.
3. Repeat steps 1-2 for 10 trials and compute the average execution time.

The result is shown as follow:

Table 5. Latency comparison between deterministic rounding and stochastic rounding

Method	Time	Latency
Deterministic Rounding	1.75e-2	1.00×
Stochastic Rounding	3.31e-2	1.88×

E. Experimental Setup

Network Architecture. For the pre-training task, we trained models based on the Llama3 (Grattafiori et al., 2024) architecture ,ranging from 60M to 1B parameters on the C4(Patel, 2020) dataset. For the supervised fine-tuning (SFT) task, we perform training based on the pre-trained Llama3 8B base model.

Baselines. To demonstrate the effectiveness of our approach, we compared the optimization performance of Q-Adam-mini against AdamW, Adam-mini, and Q-Adam in the pre-training task. For the fine-tuning experiments, in addition to the aforementioned optimizers, we incorporated LoRA (Hu et al., 2022) for further evaluation. Except for Q-Adam and Q-Adam-mini, all other optimizers are 32-bit.

Pre-training. The model sizes are $\{60\text{M}, 130\text{M}, 350\text{M}, 1\text{B}\}$, and the training data sizes are $\{1.1\text{B}, 2.2\text{B}, 6.4\text{B}, 13.1\text{B}\}$. The batch size is set to 256, and all other optimizer hyperparameters follow the same settings as in (Zhang et al., 2024b).

LoRA in Fine-tuning. LoRA can be expressed as $W \leftarrow W + BA$, where B and A are trainable parameters. We set LoRA's rank to 256, $\alpha = 128$, dropout rate (*drop*) to 0.05, and learning rate (*lr*) to 2×10^{-5}. All other hyperparameters remain consistent with (Zhang et al., 2024b).