Quantized Memory-Efficient Full-Parameter Tuning with Sign Descent Optimization

Xuezhi Zhao[1], Haichen Bai[2], Qiang Li[2*], Qi Wang[2*]
[1]*School of Computer Science and* [2]*School of Artificial Intelligence, Optics and Electronics,*
Northwestern Polytechnical University, Xi'an 710072, Shaanxi, P. R. China
{xuezhizhao1, liqmges, crabwq}@gmail.com, hcbai@mail.nwpu.edu.cn

***Abstract*—Full Parameter Fine-Tuning (FPFT) has become the preferred method for adapting LLMs to downstream tasks due to its exceptional performance. Current methods primarily utilize zeroth-order optimizers or integrate gradient computation and updates to conserve GPU memory. However, they fail to consider the optimizer states information (e.g., momentum, variance), leading to suboptimal convergence and instability during training. To address this, we propose a Quantized Memory-Efficient Full-Parameter Tuning with Sign descent optimization training framework (SQ-MEFT). Firstly, we construct a novel optimizer that uses the sign of momentum as the update amount to maximize the potential of momentum. In addition, to better maintain memory efficiency, we apply 4-bit quantization to the momentum while synchronously computing and updating gradients. When trained with mixed precision, our optimizer can reduce the total memory footprint by up to 7× compared to AdamW.**

***Index Terms*—Full Parameter Fine-Tuning, Memory Efficient, Quantization, Sign Descent**

I. Introduction

In recent years, Large Language Models (LLMs) [1]–[3] have garnered significant attention for their exceptional task generalization across diverse domains. With the exponential growth in model scale, some models have reached parameter counts in the billions. Fine-tuning the pre-trained models on specific downstream datasets can significantly enhance their task understanding and execution capabilities [4]–[6]. However, a large number of parameters has made memory capacity a primary bottleneck for training these large-scale models.

To address this, researchers introduce Parameter-Efficient Fine-Tuning (PEFT) strategies. PEFT update only a small subset of parameters or introduce minimal new ones, keeping most original parameters fixed. However, fewer trainable parameters or low-rank representations can lead to performance degradation. PEFT often exhibits a notable accuracy gap compared to Full Parameter Fine-Tuning (FPFT) across most benchmark tests. Therefore, researchers have increasingly focused on performing FPFT under resource-constrained conditions by employing techniques like zero-order optimization [7] and integrated gradient updates [8]. These methods fine-tune LLMs based on stochastic gradient descent (SGD) without requiring additional memory to store optimizer states, thereby demonstrating clear advantages in memory efficiency. However, in practical applications, these approaches often exhibit issues like poor training stability and suboptimal convergence. We attribute these problems to the insufficient utilization of optimizer states. Conversely, stateful optimizers like AdamW [9], [10] significantly improve training outcomes but demand substantial memory for optimizer states. For example, fine-tuning the LLaMA-7B model with the AdamW optimizer under mixed-precision settings may consume up to 100GB of memory, with optimizer state consumption accounting for a substantial portion of the total memory usage. Therefore, finding a balance between fully utilizing optimizer states and maintaining memory efficiency has become a critical challenge.

TABLE I
Trainable Parameter Quantity and Memory Usage in Mixed-Precision Training for AdamW, SGDM, and SQ-MEFT.

Method	Trainable Params (B)	Memory Usage (GB)			
		Params	Grads	Opt.State	Total
SGDM	M	$2M$	$2M$	$8M$	$12M$
AdamW	M	$2M$	$2M$	$12M$	$16M$
SQ-MEFT	M	$2M$	~ 0	$0.5M$	$\boldsymbol{\sim 2.5M}$

Adopting robust memory-efficient techniques is imperative. Given the simplicity and wide applicability of quantisation, we optimise memory using a low-precision numerical format to represent its optimiser state. Building on this, we introduce a Quantized Memory-Efficient Full-Parameter Tuning with Sign descent optimization method (SQ-MEFT) that effectively utilizes optimizer state information with memory-efficiency.

Mainstream stateful optimizers are often unsuitable for memory optimization. For instance, the widely used AdamW requires substantial memory for storing both momentum and variance, with the latter being difficult to compress via quantization [11]. While SGDM only tracks momentum, it performs poorly on large-scale tasks. To solve the above issue, inspired by recent progress in sign descent [12], we propose a novel sign-descent-based optimizer tailored for state compression in LLMs. Its key benefits include: 1) Simplified memory requirements: Only momentum direction is tracked, eliminating variance storage. 2) Consistent update magnitudes: By relying on momentum direction, updates are uniform across parameters, avoiding overly rapid or slow updates, promoting faster convergence and sustained stability. 3) Mitigation of imbalances and computational errors: Uniform updates magnitudes across parameters minimize imbalances and computational errors caused by limited precision post-

[*] Co-corresponding author. This work was supported by the National Natural Science Foundation of China under Grant 62301385, 62471394 and U21B2041.

quantization.

Moreover, the memory consumption from gradients is also significant. However, directly quantizing gradients to lower precision can cause overflow and reduce accuracy. To address this, we integrate gradient computation and parameter updates into a single step [8], while quantizing the optimizer states. This eliminates the need to store additional gradients during backpropagation, further reducing memory usage. Table I compares trainable parameters and RAM consumption for model states between SQ-MEFT, AdamW (with two optimizer states), and SGDM (with one optimizer state) under mixed-precision training.

In summary, our contributions are as follows:

- We analyze the advantages of sign descent in large model training and construct a novel optimizer that is suitable for quantization compression.
- We propose SQ-MEFT, a full-parameter fine-tuning framework for LLMs that optimizes memory efficiency by combining momentum quantization with synchronous gradient computation and updates.
- We evaluate the performance of LLMs fine-tuned with SQ-MEFT across three benchmark tests spanning different tasks and conduct a memory occupancy analysis on LLaMA-7B.

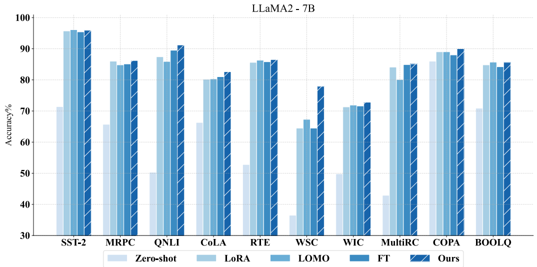


Fig. 1. LLaMA2-7B results on different fine-tuning strategies. We present the average performance over three runs with different random seeds for zero-shot, LoRA(PEFT), LOMO(FPFT), FT(AdamW) and our SQ-MEFT. SQ-MEFT generally outperforms on most tasks.

II. Preliminaries

In this section, we commence with an exposition of the foundational knowledge pertaining to the Quantization techniques for compressing the memory of momentum.

Non-linear Quantization: Quantization optimizes deep learning model storage by saving space at the expense of precision, employing quantizers and dequantizers to compress optimizer states. Mathematically, quantization involves mapping a k-bit integer to a real value within a domain D, expressed as $Q^{\text{map}} : [0, 2^k - 1] \mapsto D$. The transition from one data type to another in quantization comprises three key steps: normalization, mapping, and dequantization.

1) Normalization: In order to perform this dynamic quantization process, we use the normalization operator to scale each element of the input tensor X to the unit interval, that is $[0, 1]$. Normalization can have different granularities, such as per-tensor, per-token (row), per-channel (column) [13], group-wise and block-wise [14].

2) Mapping: The mapping step transforms normalized values into low-bit-width integers to improve storage efficiency. For an element $N_{\text{norm}}(X_j)$ obtained after normalization within the domain D, the process identifies the nearest corresponding value q_i in the quantization map using a binary search. The index i of q_i is then stored in the quantized output tensor X^Q. This can be formally described as

$$X_j^Q = \arg\min_{0 \le i < 2^k} \left| Q_i^{\text{map}} - N_{\text{norm}}(X_j) \right|, \tag{1}$$

where k is the quantization bit-width.

3) Dequantization: To obtain dequantized tensor X^D, the index stored in X^Q is retrieved, mapped to its normalized value using Q^{map}, and then denormalized with N_{norm}^{-1}, expressed as

$$X_j^D = Q^{\text{map}}(X_j^Q) \circ N_{\text{norm}}^{-1}. \tag{2}$$

III. Methodology

In this section, we introduce our proposed quantization-based memory-efficient full-parameter tuning framework SQ-MEFT.

A. Optimizer with Sign Descent for Compression

Considering the robustness of momentum against quantization and the significant memory savings achieved by retaining only the momentum, it is prudent to design an optimizer that leverages first-order momentum. Applying the sign operation to momentum when constructing update quantities ensures that each model parameter receives consistent update steps. This consistency prevents the ineffective updates that can arise from disproportionately large gradients in other parameters, thereby enhancing both the convergence rate of the model and the robustness of the training process. Moreover, the sign operation on momentum, by standardizing the update step size, mitigates the cumulative errors introduced by quantization during parameter updates. Therefore, we opt to construct a sign descent-based optimizer to support subsequent memory compression (Sec III-B). Let's first examine the classic implementation of sign-based descent with SGD [15],

$$\theta_t = \theta_{t-1} - \eta \cdot \text{sign}(g_t), \tag{3}$$

where $g_t = \nabla_\theta L(\theta_{t-1})$ is the gradient of the loss function, η is the global learning rate, and $\text{sign}(g_t)$ represents the update direction of θ_t. To facilitate faster convergence of gradient descent, incorporating momentum into the update formula is a more effective strategy, mathematically expressed as

$$m_t = \beta \cdot m_{t-1} + (1 - \beta) \cdot g_t, \tag{4}$$

$$\theta_t = \theta_{t-1} - \eta \cdot \text{sign}(m_t), \tag{5}$$

where m_t represents the first moment of the gradient, also known as momentum, and β is the EMA decay ratio. However, since sign-based descent results in uniform updates across all parameters, the global learning rate may not effectively accommodate the diverse requirements of each parameter, potentially leading to excessive update magnitudes for some. Such excessive updates can cause significant fluctuations in the loss function, leading to unstable training dynamics. To

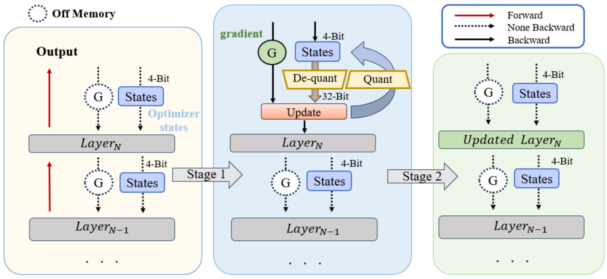


Fig. 2. Illustration of once Quantized momentum with Fused Backward.

address this issue, we propose adapting the learning rate for each parameter by utilizing the root mean square (RMS) of the parameter values from the previous iteration. This approach uses the magnitude of the parameters to scale the learning rate, reflecting their current scale and importance. As a result, it guides the model's parameters more effectively toward an optimal solution. Specifically, we introduce a new variable u to replace η, which controls the update magnitude of θ_t, u is defined as follows

$$u_t = \eta \cdot RMS(\theta_{t-1}). \tag{6}$$

By calculating the RMS of the weight parameters, we can customize the learning rate for each, enabling individualized precise adjustment of the update step size. This method allows for finer control over each parameter's update, promoting steady convergence of the model parameters towards the optimal solution, and thereby improving both the stability and efficiency of the training process. Therefore, the overall update rule is mathematically defined as

$$m_t = \beta \cdot m_{t-1} + (1 - \beta) \cdot g_t, \tag{7}$$

$$u_t = \eta \cdot RMS(\theta_{t-1}), \tag{8}$$

$$\theta_t = \theta_{t-1} - u_t \cdot \text{sign}(m_t). \tag{9}$$

As described, we propose an optimizer that incorporates only momentum, making it particularly well-suited for quantization compression. Our optimizer has just two hyperparameters, and its update process is remarkably concise. This reduction in algorithmic complexity significantly minimizes quantization error, thereby providing a great foundation for subsequent compression techniques.

B. Quantized Momentum with Fused Backward

To further enhance memory efficiency, we implement momentum quantization. We apply 4-bit quantization compression to the momentum. Recognizing that outliers can significantly distort the quantization scale in 4-bit quantization, we employ block-wise quantization to isolate outliers and thereby minimize quantization error. This method is further enhanced by dynamic exponent mapping. Specifically, block-wise quantization divides the tensor into smaller blocks of size B, allowing for independent normalization within each block. This approach not only reduces the computational cost of normalization, but also improves quantization accuracy by isolating outliers within each block. Compared to conventional quantization methods that directly normalize the input tensor to the range [-1, 1], which requires multiple synchronizations across GPU cores, block-wise quantization reduces overhead and boosts throughput. By treating the tensor as a one-dimensional array and dividing it into n/B blocks, normalization can be performed independently for each block, enhancing robustness against outliers in the tensor. The normalization operator used in block-wise quantization is defined as

$$N_{\text{block-wise}}(X_{bj}) = X_{bj} / \max |X_b|, 0 < j < B, \tag{10}$$

where b is the index of the block $0..n/B$. Then each block is quantized independently, Eq.1 can be modified as follows

$$X_{bj}^{Q} = \arg \min_{0 \leq i < 2^k} |Q_i^{\text{map}} - N_{\text{block-wise}}(X_{bj}), 0 < j < B. \tag{11}$$

To balance the trade-off between quantization error and memory overhead, we employ a smaller block size of 128. Compared to larger blocks, smaller blocks can provide a closer approximation of the momentums. During the training, gradients consume significant memory. The fused backward approach [8] computes gradients and updates parameters simultaneously during backpropagation, effectively reducing the memory footprint of gradients. In detail, we express standard backpropagation as $grad = \frac{\partial \mathcal{L}}{\partial p}$, $p = p - lr \cdot g$, while the fused backward is $p = p - lr \cdot \frac{\partial \mathcal{L}}{\partial p}$. Specifically, in a single backpropagation pass, the gradient of a given parameter is temporarily stored in GPU memory and is immediately released after the corresponding parameter has been updated [8]. We seamlessly integrate the aforementioned quantization compression of momentum into each instance of the fused backward process. The specific implementation is illustrated in Fig. 2.

Algorithm 1 Quantized Memory-Efficient Full-Parameter Tuning with Sign Descent Optimization

Input: model $f(\cdot)$ with parameter $\boldsymbol{\theta}$, initial first moment $\bar{m}_0 = 0$, learning rate η, max step T, training dataset D, loss function L, momentum parameter β.

```
for $t = 1$ to $T$ do
    sample batch $B = (\boldsymbol{x}, \boldsymbol{y}) \subset D$
    $\hat{\boldsymbol{y}} \leftarrow f(\boldsymbol{x}, \boldsymbol{\theta})$
    $\ell \leftarrow L(\boldsymbol{y}, \hat{\boldsymbol{y}})$
    for each parameter $\theta_i$ in the order of backpropagation do
        $g_{t,i} = \nabla_{\theta_{t-1,i}} \ell$
        $m_{t-1,i} \leftarrow$ decompress$(\bar{m}_{t-1,i})$
        $m_{t,i} \leftarrow \beta \cdot m_{t-1,i} + (1 - \beta) \cdot g_{t,i}$
        $u_{t,i} = \eta \cdot RMS(\theta_{t-1,i})$
        $\theta_{t,i} = \theta_{t-1,i} - u_{t,i} \cdot$ sign$(m_{t,i})$
        $\bar{m}_{t,i} \leftarrow$ compress$(m_{t,i})$
        $g_{t,i} \leftarrow$ None(clear gradients)
    end for
end for
```

TABLE II
RESULTS ON GLUE AND SUPERGLUE FOR DIFFERENT FINE-TUNING STRATEGIES (WITH 1000 EXAMPLES).

Method	SST-2	MRPC	QNLI	CoLA	RTE	WSC	WIC	MultiRC	COPA	Boolq	Avg.
OPT-125M (sequence length=512)											
Zero-shot	50.9	65.9	49.7	67.1	50.2	35.6	48.7	45.3	66.0	49.2	52.9
4bit-AdamW	86.4	76.0	80.4	70.0	63.9	64.6	61.0	67.0	70.0	62.3	**70.2**
AdamW	86.7	75.9	77.3	**71.4**	63.1	63.8	59.9	66.8	68.0	**62.4**	69.5
LoRA	86.9	75.4	78.0	69.0	59.3	63.5	59.7	64.9	67.0	**62.4**	68.6
DoRA	86.8	75.4	78.0	69.2	62.6	64.4	60.9	67.0	71.0	61.6	69.7
LOMO	87.0	**76.9**	**82.3**	68.9	62.5	63.6	59.5	66.6	67.0	62.0	69.6
Ours	**87.1**	75.7	79.6	69.0	**63.9**	**64.8**	**61.2**	**67.2**	**71.0**	61.9	70.1
OPT-1.3B (sequence length=512)											
Zero-shot	70.4	68.1	49.2	62.6	58.8	36.5	49.0	45.8	73.0	53.1	56.7
4-bit AdamW	**93.5**	79.1	83.1	78.8	75.1	**65.4**	65.5	71.7	77.0	73.5	76.3
AdamW	92.8	78.3	82.0	78.9	72.9	64.5	65.2	72.6	76.0	74.3	75.8
LoRA	93.0	**83.0**	82.4	79.1	75.4	63.4	66.4	73.3	75.0	75.1	76.6
DoRA	92.5	78.9	85.1	78.0	74.3	63.4	66.5	**74.3**	78.0	**76.2**	76.7
LOMO	92.3	79.9	**86.2**	**80.2**	71.5	64.4	65.3	68.7	78.0	76.1	76.3
Ours	**93.5**	81.2	85.9	79.9	**76.3**	**65.4**	**68.0**	73.5	**79.0**	75.4	**77.8**
LLaMA-7B (sequence length=1024)											
Zero-shot	61.9	67.3	52.0	68.9	57.0	36.5	49.7	42.3	85.0	66.5	58.7
4bit-AdamW	96.2	**84.8**	90.2	79.8	85.9	63.5	68.5	84.4	89.0	86.4	82.9
LoRA	96.1	84.0	90.5	**83.2**	85.9	64.4	65.5	84.8	87.0	85.2	82.7
DoRA	95.3	84.1	90.0	82.5	87.6	68.0	71.4	**86.1**	88.0	86.9	84.1
LOMO	92.3	82.5	89.5	81.1	86.6	66.4	71.2	84.0	89.0	87.5	83.0
Ours	**96.4**	84.3	**91.0**	82.6	**88.1**	**70.2**	**72.1**	85.3	**90.0**	**87.6**	**84.8**

TABLE III
PERFORMANCE OF THE LLAMA-7B ON MMLU AFTER INSTRUCTION-TUNING ON THE ALPACA AND FLAN V2 DATASETS.

Model	Method	MMLU (5shot)	
		Alpaca	FLan V2
LLaMA-7B	N/A	33.4	33.4
	LoRA	36.8	42.4
	QLoRA	37.4	43.2
	32-bit AdamW	38.7	-
	4-bit AdamW	38.9	45.1
	LOMO	34.9	40.8
	SQ-MEFT (ours)	38.7	45.3

C. Overall Framework

In this section, we integrate the optimizer based on sign descent and the memory compression method from the Sec. III-B to introduce a memory-efficient fine-tuning framework for LLMs. We provide a comprehensive description of the training process, as depicted in Algorithm 1. Where θ_t represents the parameters of the model at the t^{th} training step, with $\theta_{t,i}$ denoting the parameters of θ_t during the i^{th} gradient computation in a single backpropagation pass. Similarly, $g_{t,i}$ denotes the gradient of $\theta_{t,i}$, $m_{t,i}$ represents the corresponding momentum, and $u_{t,i}$ indicates the update quantity. The terms $compress$ and $decompress$ refer to quantization compression and dequantization, respectively. During the update process, only the compressed state $\bar{m}$ is persistently stored in the GPU.

IV. Experiments

In this section, we perform experimental comparisons with other methods.

A. Experiment Setup

Models, Benchmarks, Baselines and Datasets. We conduct extensive experiments on the SuperGLUE [16], GLUE [17], and Multi-Task Language Understanding (MMLU) benchmarks [18] to evaluate the effectiveness of the proposed SQ-MEFT method in fine-tuning LLMs for downstream tasks and instruction tuning scenarios. In downstream tasks, we fine-tune models including OPT-125M, OPT-1.3B, LLaMA-7B [19], and LLaMA2-7B [20]. Meanwhile, SQ-MEFT is compared against Zero-shot (a method that requires no fine-tuning), LoRA (a partial parameter fine-tuning approach) [21], LOMO (a memory-efficient full-parameter fine-tuning approach) [8], DoRA (an improved variant of LoRA) [22], 4-bit quantized AdamW [11], and the traditional 32-bit AdamW [10]. For instruction tuning, we additionally include a comparison with QLoRA [23]. When evaluating model performance using the SuperGLUE benchmark, our focus is directed towards the RTE, WSC, WiC, MultiRC, Boolq and COPA. In the GLUE benchmark, the selected tasks are SST-2, MRPC, QNLI, and CoLA. For our instruction tuning experiments, the Alpaca [24] dataset and the FLan V2 [25] dataset were utilized as training resources, and the breadth of our modeling capabilities was subsequently evaluated using the large-scale MMLU benchmark.

B. Implementation

In assessing model performance on downstream tasks, we randomly sample 1000 training instances from the training set and 1000 test instances from the validation set. Each experiment is conducted three times under different random seeds, and the average performance is reported. All experiments are conducted

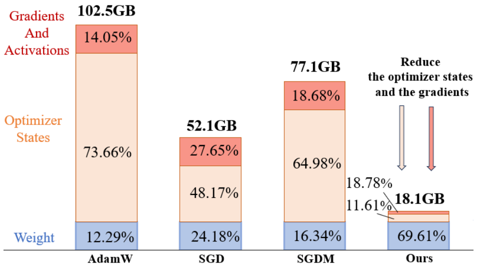


Fig. 3. The memory ratio of each part when fine-tuning LLaMA-7B with different methods. Our approach significantly reduces the memory occupied by the optimizer states.

TABLE IV
MEMORY USAGE (GB) WHEN FINE-TUNING THE LLAMA-7B MODEL USING DIFFERENT METHODS ON THE SST-2 DATASET. THE SEQUENCE LENGTH AND BATCH SIZE ARE SET TO 512 AND 8. THE ACTIVATION IS THE RESULT OF INTEGRATION WITH ACTIVATION CHECKPOINTING.

Method	Params	Gradients	States	Acti.	Total
AdamW	12.6	12.6	75.5	1.8	102.5
SGD	12.6	12.6	25.1	1.8	52.1
SGDM	12.6	12.6	50.1	1.8	77.1
LOMO	12.6	0.3	0	1.8	14.7
SQ-MEFT	12.6	0.3	3.4	1.8	18.1

using PyTorch on eight NVIDIA RTX 3090 GPUs and two Tesla A100 GPUs.

C. Performance Evaluation

Main Result on Glue and SuperGlue Benchmarks. As shown in Table II and Fig. I. The results demonstrate that SQ-MEFT establishes new state-of-the-art results in parameter-efficient fine-tuning on both the GLUE and SuperGLUE benchmarks. SQ-MEFT consistently outperformed Zero-shot across all datasets and exhibited stronger performance compared to other methods in most cases. The comparison with LoRA confirmed that full-parameter fine-tuning yields better results due to the adjustment of a larger number of parameters. Compared to LOMO, SQ-MEFT significantly improved fine-tuning performance by incorporating compressed optimizer states, albeit with a slight increase in memory consumption. Moreover, SQ-MEFT achieved better results than AdamW while drastically reducing memory usage. In summary, SQ-MEFT strikes an excellent balance between performance and memory consumption.

Main Result on MMLU Benchmark. The experimental results on the MMLU benchmark are shown in Table III. Fine-tuning the LLaMA-7B model on the Alpaca dataset using SQ-MEFT achieved an accuracy of 38.7% on the MMLU dataset. Our approach matches the performance of the traditional full-precision AdamW while reducing memory consumption by a factor of 7.

D. Memory Analysis

In this study, we exhibit the memory usage of various optimizers during the training of the LLaMA-7B model under mixed-precision strategies, as shown in Table IV. Our analysis begins with a comparison between SQ-MEFT and the traditional AdamW optimizer, where memory usage was significantly reduced from 102.5GB to 18.1GB. Specifically, when using the AdamW optimizer, a large portion of memory is allocated to optimizer states, including momentum, variance, and full-precision copies of the weights. These components are crucial for maintaining optimization accuracy and stability, consuming approximately six times the memory of the model weights, as illustrated in Fig. 3, leading to substantial resource demands. By replacing AdamW with the SGD optimizer, the memory allocated to optimizer states was significantly reduced, decreasing GPU memory usage from 102.5GB to 52.1GB. This reduction is primarily due to the SGD optimizer's lack of full-precision momentum and variance storage. Compared to SGD, the SGDM optimizer introduces momentum storage, enhancing training efficiency and adaptability but at the cost of an additional 25GB of memory usage. In contrast, our method retains momentum while quantizing it to 4 bits and merges gradient computation with parameter updates into a single step, drastically reducing the memory footprint for optimizer states and gradients. As shown in Table IV, this approach requires only 0.3GB for gradient storage and 3.4GB for optimizer state space, resulting in an overall memory reduction of 50GB compared to SGD. Compared to LOMO, which does not store any optimizer states, our method consumes only an additional 3.4GB of memory but significantly improves performance.

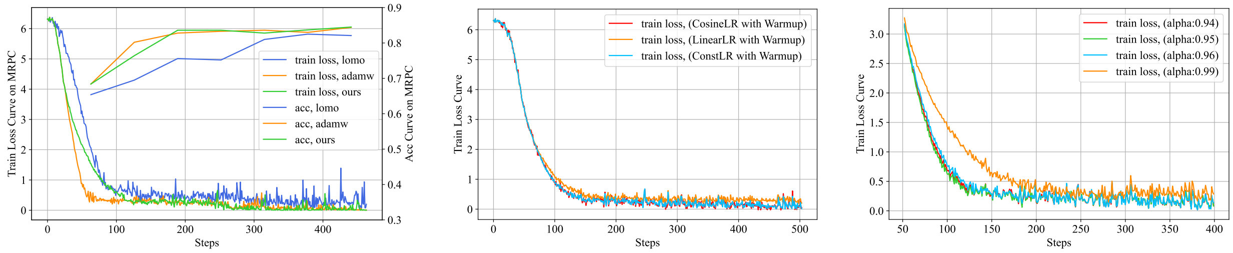
E. Ablation Study and Analysis

In this section, we conduct a comprehensive ablation study validated on the MRPC dataset using the LLaMA-7B model, with a learning rate η set to 8e-5.

Convergence Analysis Comparison. To investigate the convergence of our method, we conduct empirical comparisons with AdamW and LOMO, with convergence results shown in Fig. 4(a). Compared to LOMO, SQ-MEFT achieve superior convergence. Our method converges slowly than AdamW, it eventually reach a consistent convergence.

Examining the Impacts of Learning Scheduler and RMS. To verify the effectiveness of incorporating RMS adjustments into the learning rate strategy, we design and conduct a series of ablation experiments. Table V shows the introduction of RMS-adjusted learning rates improve accuracy. Furthermore, we explored the impact of various learning rate scheduling strategies. As illustrated in Fig. 4(b), a rapid linear increase in the learning rate may lead to underfitting, while using ConstLR or CosineLR yields optimal training results.

Examining the Impacts of EMA and Sign Descent. In order to verify the effectiveness of EMA, we perform ablation experiments using a very low decay coefficient, with results as shown in the table V. To determine the optimal range of the EMA coefficient β, we set a range of β values between 0.9 and 0.99 for the experiment. Fig. 4(c) shows that β value in the range of 0.94 to 0.96 work best. Additionally, to thoroughly investigate the impact of sign descent, we remove the sign operation from Eq. 9 while keeping all other factors constant and use momentum directly as the update quantity. Table V effectively confirms the positive impact of sign descent on enhancing optimizer performance.



(a) Fine-tuning LLaMA-7B with SQ-MEFT exhibits great convergence.

(b) Training loss curves for different Learning rate Schedulers.

(c) Training loss curve for different β ratios.

Fig. 4. Ablation study of fine-tuning LLaMA-7B on the MRPC dataset.

TABLE V
ABLATION STUDY OF SIGN DESCENT, EMA AND RMS: AVERAGE ACCURACY PERFORMANCE OF LLAMA-7B FINE-TUNING ON QNLI, COLA, AND RTE DATASETS.

Sign descent	EMA	RMS	Acc. (%)
-	-	-	85.7
-	✓	-	86.1
-	✓	✓	86.3
✓	✓	-	86.8
✓	✓	✓	**87.2**

V. CONCLUSION

We propose a new full-parameter fine-tuning framework, SQ-MEFT, designed to balance memory efficiency and model performance. Specifically, we develop a novel optimizer based on sign descent that is well-suited for quantization and compression. On the basis, we apply 4-bit quantization to the optimizer's momentum while integrating gradient computation and parameter updates into a single step. The effective combination of these strategies significantly reduces the memory footprint of both the optimizer states and gradients. While empirical results highlight its benefits, the theoretical foundation of sign descent remains unexplored, and performance under large batch sizes is yet to be validated due to computational constraints.

REFERENCES

[1] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, pp. 681–694, 2020.

[2] L. Ouyang, J. Wu *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.

[3] A. Chowdhery, Narang *et al.*, "Palm: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.

[4] T. Liu, X. Liu, S. Huang, H. Chen, Q. Yin, L. Qin, D. Wang, and Y. Hu, "Dara: Domain- and relation-aware adapters make parameter-efficient tuning for visual grounding," in *2024 IEEE International Conference on Multimedia and Expo (ICME)*, 2024, pp. 1–6.

[5] J. Yang, Z. Li, S. Xie, W. Zhu, W. Yu, and S. Li, "Cross-modal adapter: Parameter-efficient transfer learning approach for vision-language models," in *2024 IEEE International Conference on Multimedia and Expo (ICME)*, 2024, pp. 1–6.

[6] Z. Xie, B. Guan, W. Jiang, M. Yi, Y. Ding, H. Lu, and L. Zhang, "Pa-sam: Prompt adapter sam for high-quality image segmentation," in *2024 IEEE International Conference on Multimedia and Expo (ICME)*, 2024, pp. 1–6.

[7] S. Malladi, T. Gao, E. Nichani, A. Damian, J. D. Lee, D. Chen, and S. Arora, "Fine-tuning language models with just forward passes," *Advances in Neural Information Processing Systems*, vol. 36, pp. 53 038–53 075, 2023.

[8] K. Lv, Y. Yang, T. Liu, Q. Gao, Q. Guo, and X. Qiu, "Full parameter fine-tuning for large language models with limited resources," *arXiv preprint arXiv:2306.09782*, 2023.

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

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

[11] B. Li, J. Chen, and J. Zhu, "Memory efficient optimizers with 4-bit states," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[12] F. Kunstner, J. Chen, J. W. Lavington, and M. Schmidt, "Noise is not the main factor behind the gap between sgd and adam on transformers, but sign descent might be," *arXiv preprint arXiv:2304.13960*, 2023.

[13] Y. Bondarenko, M. Nagel, and T. Blankevoort, "Understanding and overcoming the challenges of efficient transformer quantization," *arXiv preprint arXiv:2109.12948*, 2021.

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

[15] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signsgd: Compressed optimisation for non-convex problems," in *International Conference on Machine Learning*, 2018, pp. 560–569.

[16] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "Superglue: A stickier benchmark for general-purpose language understanding systems," *Advances in neural information processing systems*, vol. 32, 2019.

[17] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

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

[19] H. Touvron, T. Lavril *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[20] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[21] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[22] S.-Y. Liu, C.-Y. Wang, H. Yin, P. Molchanov, Y.-C. F. Wang, K.-T. Cheng, and M.-H. Chen, "Dora: Weight-decomposed low-rank adaptation," *arXiv preprint arXiv:2402.09353*, 2024.

[23] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[24] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, "Stanford alpaca: An instruction-following llama model," 2023.

[25] S. Longpre, L. Hou, T. Vu, A. Webson, H. W. Chung, Y. Tay, D. Zhou, Q. V. Le, B. Zoph, J. Wei *et al.*, "The flan collection: Designing data and methods for effective instruction tuning," in *International Conference on Machine Learning*. PMLR, 2023, pp. 22 631–22 648.