# Periodical Moving Average Accelerates Gradient Accumulation for Post-Training

**Yumou Liu**[*1] **An Li**[1] **Chaojie Li**[1] **Fei Yu**[1] **Benyou Wang** [†1]

[1]School of Data Science
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China

## Abstract

High gradient variance presents a significant obstacle to efficient post-training of large language models (LLMs) on memory-constrained devices. Existing practical strategiessuch as reducing batch sizes or adopting gradient accumulation (GA)suffer from an inherent trade-off: smaller batches exacerbate convergence issues due to increased gradient noise, while GA substantially prolongs training time owing to its sequential processing. In this work, we reveal that the Exponential Moving Average (EMA) in momentum-based optimizers exponentially discounts historical gradients, thereby limiting their effectiveness in stabilizing parameter updates, especially during post-training when parameter drift is minimal. Motivated by this, we propose integrating the core idea of GA directly into momentum updates via a novel *Periodical Moving Average* (PMA) mechanism, which structures training into fixed periods and replaces EMA with a uniform moving average within each period. We instantiate PMA within AdamW and Lion, resulting in the AdamW-PMA and Lion-PMA optimizers. Theoretical analysis establishes that AdamW-PMA matches the convergence guarantees of standard Adam. Extensive empirical evaluation on supervised fine-tuning and direct preference optimization tasks demonstrates that PMA-based methods achieve approximately $2\times$ faster training compared to GA, while yielding consistently better performance on downstream evaluations.

[*]yumouliu@link.cuhk.edu.cn
[†]Benyou is the corresponding author

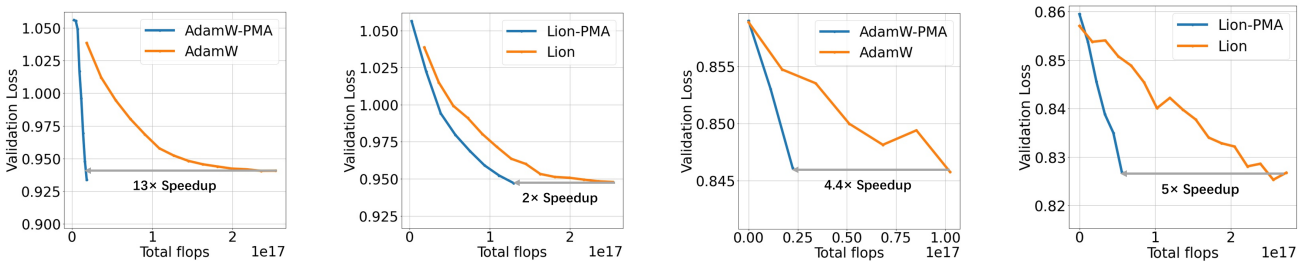
## 1 INTRODUCTION

Scaling large language models (LLMs) has consistently propelled advances in model capability and generalization [Radford et al., 2019, Kaplan et al., 2020, Brown et al., 2020, Hoffmann et al., 2022, Zhang et al., 2022, Touvron et al., 2023a,b, Achiam et al., 2023, Bi et al., 2024]. Modern post-training stages—including supervised fine-tuning (SFT) and reinforcement learning from human feedback [OpenAI, 2024]—exert substantial computational and memory burdens, typically necessitating powerful multi-GPU clusters [Lee and Sengupta, 2022]. However, scaling intensifies GPU memory constraints, presenting major challenges for training or adaptation on memory-limited devices.

Practical strategies for training LLMs under memory constraints are hampered by efficiency limitations. Simple batch size reduction increases gradient variance, significantly slowing convergence. Alternatively, gradient accumulation (GA) alleviates memory bottlenecks by splitting a large batch into multiple smaller updates, accumulating gradients over several micro-batches before a single parameter update. While GA emulates large-batch training, it transforms parallel computation into sequential updates, substantially elongating training time.

In this work, we introduce the *Periodical Moving Average* (`PMA`), a novel momentum update scheme tailored to accelerate post-training of LLMs on memory-limited hardware by addressing both variance and computational overhead.

Existing approaches involve a fundamental trade-off: GA cannot interleave parameter updates within accumulation periods without losing memory efficiency, while small-batch training suffers from high variance and unstable parameter updates. Our central insight is that, during post-training, LLM parameters evolve gradually due to low learning rates, leading to consecutive gradients with highly similar expectations. This motivates rethinking the exponential moving average (EMA): while EMA discounts historical gradients exponentially and thus rapidly forgets useful information, a




(a) AdamW vs. AdamW−PMA on SFT.

(b) Lion vs. Lion−PMA on SFT.

(c) AdamW vs. AdamW−PMA on DPO.

(d) Lion vs. Lion−PMA on DPO.

Figure 1: Optimizers with PMA achieve approximately 2× speedup compared to EMA-based optimizers. (1a, 1b) show SFT validation loss on Phi-2 2.7B with Alpaca; (1c, 1d) show DPO validation loss on Phi-2 2.7B with HH-RLHF.

properly designed moving average can better utilize recent history for stabilization.

PMA partitions training into discrete periods of $K$ steps each. Within each period, it replaces EMA with a simple moving average in the momentum update, uniformly weighting recent gradients to reduce variance. Between periods, it resets to standard EMA, thus preserving rapid convergence and optimizer stability. This hybrid mechanism achieves a favorable trade-off between memory efficiency and convergence speed.

A notable challenge with PMA is *trajectory deviation*, wherein uniform averaging within a period may cause parameter updates to diverge from the ideal optimization path, due to lack of access to the true gradient expectation. To mitigate this, we employ a *periodic learning rate decay*: the learning rate linearly decreases within each period and resets at period boundaries, ensuring that parameter updates remain stable and aligned with the underlying optimization trajectory.

We instantiate PMA within AdamW [Loshchilov and Hutter, 2017] and Lion [Chen et al., 2023], resulting in AdamW−PMA and Lion−PMA, respectively. We thoroughly evaluate these variants across SFT and Direct Preference Optimization (DPO) [Rafailov et al., 2023] tasks on a range of models, including GPT-2 [Brown et al., 2020], Phi-2 [Javaheripi et al., 2023], Qwen1.5 [Team, 2024a], Qwen2 [Yang et al., 2024], and Llama2 [Touvron et al., 2023b]. Our experiments show that AdamW−PMA and Lion−PMA achieve approximately 2× speedup over traditional EMA-based optimizers, while attaining better downstream task performance. We further provide theoretical analysis of the learning rate strategy and regret bound, establishing that AdamW−PMA retains the convergence guarantees of Adam.

**Our key contributions are:**

- We propose Periodical Moving Average (PMA), a momentum update mechanism that accelerates large-batch emulation for LLM post-training on memory-limited devices. When applied to AdamW and Lion (AdamW−PMA and Lion−PMA), our method stabilizes training and reduces training time and data requirements, without incurring extra per-step memory or computation.
- We provide comprehensive empirical validation across models ranging from 0.1B to 7B parameters and for both SFT and DPO settings. PMA-based optimizers realize up to 2× speedup over GA, consistently outperforming baselines in downstream evaluation [1].
- We theoretically analyze the convergence of AdamW−PMA, highlighting the effectiveness of the dynamic learning rate strategy and establishing regret guarantees on par with standard Adam.

## 2 PRELIMINARIES

### 2.1 BACKGROUND: FIRST-ORDER OPTIMIZATION

Adam [Kingma and Ba, 2014, Reddi et al., 2019] and AdamW [Loshchilov and Hutter, 2017] are among the most widely used optimizers for large language model (LLM) training, integrating adaptive learning rates and momentum-based techniques. Given an objective function $f : \mathbb{R}^d \to \mathbb{R}$, at iteration $t$, let the stochastic gradient be $g_t = \nabla f(x_t)$. Adam maintains exponential moving averages of the first and second moments: $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$ and $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$, where $\beta_1$ and $\beta_2$ are hyperparameters, and all operations are element-wise. A small constant $\epsilon > 0$ is typically added to $v_t$ for numerical stability. To correct the bias introduced at initialization, the moment estimates are debiased: $\hat{m}_t = m_t/(1-\beta_1^t)$ and $\hat{v}_t = v_t/(1-\beta_2^t)$. Since historical gradients receive exponentially decaying weights, these averages are referred to as exponentially weighted moving averages (EMAs). The parameter update is then

$$x_{t+1} \leftarrow x_t - \gamma \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{1}$$

[1] Implementation available at https://github.com/liuyumou/periodical-moving-average.git

where $\gamma$ is the learning rate. All operations are performed element-wise. AdamW decouples weight decay from the gradient update. More recently, Lion [Chen et al., 2023] has been proposed, which relies solely on the first moment and updates parameters using EMA.

Despite their effectiveness, Adam-based methods encounter difficulties in high-variance settings, particularly under memory constraints. Training LLMs is intrinsically a high-variance optimization problem [McCandlish et al., 2018]. To mitigate variance, practitioners commonly increase the batch size using high-performance clusters [Touvron et al., 2023a]. Conversely, reducing batch size further amplifies the stochasticity of gradient estimates, slowing convergence and degrading optimization, especially in memory-limited environments [Yuan et al., 2016, Bottou et al., 2018, Kunstner et al., 2023, Fu et al., 2023].

### 2.2 THEORETICAL SOLUTION: VARIANCE REDUCTION IN SGD

Here we review representative approaches for reducing gradient variance in stochastic optimization [Bottou et al., 2018], alongside their limitations. *Variance reduction* techniques typically reuse historical information to construct lower-variance gradient estimates. For instance, SVRG [Johnson and Zhang, 2013] maintains a snapshot parameter $\theta_k$ (with $k < t$) and leverages it for gradient correction. The iterate averaging method [Polyak, 1991] averages the iterates across steps to yield a final estimate. Recent advances such as SARAH [Nguyen et al., 2017] and STORM [Cutkosky and Orabona, 2019] adopt recursive update rules that avoid explicit storage of past gradients.

However, these variance reduction methods exhibit practical deficiencies in the LLM context. Approaches like SAGA incur prohibitive memory costs by requiring storage of a gradient for every data sample, with memory usage scaling with dataset size. Iterate averaging demands storing all historical parameter vectors, incurring memory overhead proportional to the number of steps. SVRG relies on large-batch computations at each snapshot, increasing memory requirements. Although SARAH and STORM mitigate storage needs by not retaining past gradients explicitly, they require multiple backpropagation passes per parameter update, which substantially increases computational cost during LLM training.

### 2.3 PRACTICAL SOLUTION: GRADIENT ACCUMULATION

To address high gradient variance under memory constraints, *gradient accumulation* (GA)[2] provides a straightforward solution. GA divides a large batch into $K$ smaller micro-batches processed sequentially, accumulating gradients computed on each without exceeding the device memory limit. After accumulating gradients over the $K$ micro-batches, the optimizer averages them to approximate the gradient over the full batch and performs a parameter update. Notably, the gradient accumulated via GA is mathematically equivalent to that obtained from a large batch.

Despite its statistical soundness, GA presents substantial practical drawbacks, primarily in terms of increased wall-clock time. That is, GA achieves memory efficiency by trading off parallelism for serial computation: on resource-limited devices, each parameter update requires $K$ successive forward and backward passes, leading to lower computational efficiency than if the large batch could be processed in parallel. Recent variants such as Pham et al. [2023] further reduce memory usage in GA, but do not alleviate the increased training time on memory-constrained hardware.

### 2.4 THE DILEMMA

The discussions in Section 2.2 and Section 2.3 reveal a fundamental dilemma between theoretically grounded and practically feasible approaches for high-variance optimization under GPU memory constraints. On the one hand, state-of-the-art variance reduction methods frequently incur prohibitive memory costs, rendering them unsuitable for resource-limited environments. On the other hand, practical strategies either converge slowly due to elevated gradient variance or require substantially increased training time, as in the case of the sequential computations inherent to gradient accumulation (GA). Crucially, this dilemma stems from the persistence of high variance in stochastic gradients. Therefore, there is a pressing need for novel methods that can reduce the variance of parameter updates without incurring additional memory or computational overhead.

## 3 METHODOLOGY: PERIODICAL MOVING AVERAGE

To address the dilemma identified in Section 2.4, we propose `PMA` as an enhancement of the exponential moving average (EMA) process to more effectively reduce gradient variance. Section 3.1 presents the high-level design and intuition underlying `PMA`, including its connections and distinctions to existing work. Section 3.2 provides detailed implementation, with an emphasis on dynamics of $\beta$ and learning rate scheduling, while Section 3.3 describes the integration of `PMA` with AdamW and Lion.

[2]While memory-efficient optimizers such as those proposed in [Shazeer and Stern, 2018, Luo et al., 2023, Zhao et al., 2024, Zhang et al., 2024] are also viable, we argue that GA remains more practical for our scenario. Detailed discussion appears in § 6.

### 3.1 HIGH-LEVEL IDEA

**Mimicking GA in Momentum Updates.** At a high level, `PMA` simulates the variance reduction of gradient accumulation (GA) within momentum-based optimizers. Unlike standard EMA, which exponentially discounts past gradients, our method maintains uniform weighting for recent gradients within fixed periods. By partitioning training iterations into periods and employing a vanilla moving average for momentum updates within each period, `PMA` mimics the effect of GA, providing moment estimates of lower variance analogous to those obtained by EMA-based optimizers using GA (§3.2.1).

**From Pure Accumulation to Progressive Updates.** Whereas standard GA does not update parameters until the end of each accumulation period, `PMA` interleaves updates within each period. Specifically, `PMA` alternates between steps with large and small learning rates: we designate the former as *large update steps*, typically at the culmination of a period, and the latter as *small update steps*. Each large update step, taken after $K$ small update steps, emulates the behavior of EMA-based optimizers with GA, while the intervening small update steps facilitate faster convergence. By judiciously choosing a reduced learning rate for these small steps, we seek to accelerate optimization without destabilizing the variance reduction effect (§3.2.2).

### 3.2 DETAILED DESIGN

We now describe the update rules governing the first moment (momentum) as an illustrative example.

#### 3.2.1 Momentum Update: Dynamics of $\beta$

Unlike conventional EMA, where $\beta$ is fixed, `PMA` employs a dynamically adjusting $\beta$ to achieve a uniform moving average within each accumulation period.[3] Uniform weighting requires systematically decaying $\beta$ during each period so that each historical gradient within the period contributes equally.

We describe momentum updates for both the large and small update steps. See Fig. 2a for a visualization.

**Large Update Step: Low Gradient Weight for Variance Control.** At the first small update step following a large update step ($\tau = 0$), the momentum update is

$$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)\frac{g_t}{K},$$

where $K$ denotes the accumulation length. Unlike EMA, here the current gradient is scaled down by $1/K$ and, after the update, the first and second momenta are scaled by $K$. This design both reduces the variance of the momentum estimates and, after $K$ steps, ensures the accumulated gradients receive equal weighting, consistent with the behavior of GA.

**Small Update Steps: Uniform Moving Average via Dynamic Weights.** For subsequent small steps within the period ($\tau = 1, \ldots, K-1$), the momentum is updated as

$$m_t \leftarrow \frac{\tau}{\tau+1} m_{t-1} + \frac{1-\beta_1}{\tau+1} g_t.$$

This procedure, in essence, replaces the EMA with a vanilla moving average, ensuring that the gradients from each small step contribute equally in $m_{t,K-1}$. Notably, in $m_{t,K-1}$, $m_{t-1,K-1}$ receives weight $1-\beta_1$, and each $g_{t,\tau}$ (for all $\tau$) is weighted by $\frac{1-\beta_1}{K}$, conforming with the GA weighting scheme.

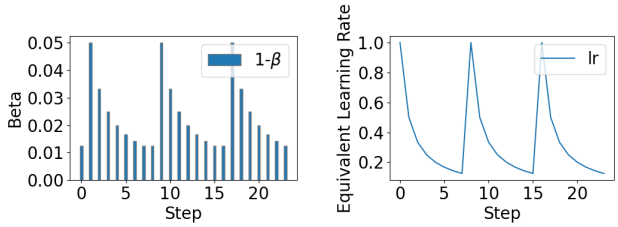
**Algorithm 1:** AdamW with Periodical Moving Average (AdamW-PMA)

**Input**: Learning rate $\gamma$; coefficients $(\beta_1, \beta_2)$;
Initial params $\theta_0$; objective $f(\theta)$;
Epsilon $\epsilon$; weight decay $\lambda$;
Period length $K$ (# microsteps per period)

**Data:** $m_0 \leftarrow 0, v_0 \leftarrow 0$

**for** $t = 1, 2, \ldots$ **do**
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$;
  $\tau \leftarrow t \bmod K$;
  **if** $\tau = 0$ ***and*** $t > 0$ **then** `// Large update step every K iterations`
    $\gamma_t \leftarrow \gamma$;
    $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)\frac{g_t}{K}$ `// Divide by K for stability`
    $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2)\frac{g_t^2}{K}$;
  **else**
    $\gamma_t \leftarrow \gamma/\sqrt{K}$ `// Small update: reduced LR`
    $m_t \leftarrow \frac{\tau}{\tau+1} m_{t-1} + \frac{1-\beta_1}{\tau+1} g_t$ `// Moving average update`
    $v_t \leftarrow \frac{\tau}{\tau+1} v_{t-1} + \frac{1-\beta_2}{\tau+1} g_t^2$;
  `// Debiasing (adjust for initialization bias)`
  $\hat{m}_t \leftarrow m_t/(1-\beta_1^{t//K})$;
  $\hat{v}_t \leftarrow v_t/(1-\beta_2^{t//K})$;
  `// Apply weight decay`
  $\bar{\theta}_t \leftarrow (1-\gamma_t\lambda)\theta_{t-1}$;
  `// Update parameter`
  $\theta_t \leftarrow \bar{\theta}_t - \gamma_t \hat{m}_t/(\sqrt{\hat{v}_t}+\epsilon)$;
  **if** $\tau = 0$ ***and*** $t > 0$ **then** `// Rescale after large update step`
    $\hat{m}_t \leftarrow K\hat{m}_t$;
    $\hat{v}_t \leftarrow K\hat{v}_t$;
**return** $\theta_t$

[3] Here, $\beta$ denotes the weight of the previous momentum; $1-\beta$ is the weight of the current gradient. For clarity, we retain the notation $\beta$ throughout.




(a) Dynamic $\beta$ ($K = 8$, $\beta = 0.9$)

(b) Learning rate schedule ($K = 8$, $\gamma = 1$)

Figure 2: Illustrations of dynamic $\beta$ and learning rate scheduling.

#### 3.2.2 Learning Rate Schedule

Complementing the momentum update, we design a learning rate schedule that distinguishes between small and large update steps. Specifically, small update steps employ a linearly decaying learning rate rather than a fixed schedule. As shown in Fig. 2b, this mitigates the risk of excessive parameter advancement that could undermine the variance reduction effect intended by GA-mimicking. For step $\tau$, the effective learning rate is given by

$$\eta_\tau = \frac{\eta}{\tau + 1},$$

where $\eta$ is the base learning rate. Such linear decay ensures that later small steps cause less drift from the virtual GA reference point, preserving the intended statistical behavior.

### 3.3 CASE STUDY

#### 3.3.1 From AdamW to `AdamW-PMA`

To obtain `AdamW-PMA`, we substitute the EMA used in AdamWs first and second moment estimators with the periodical moving average scheme detailed above. The pseudocode is provided in Algorithm 1. Both momenta are scaled at large update steps, while at small steps, the learning rate is effectively scaled by $1/\sqrt{K}$, owing to the combination of momentum and learning rate scaling. All remaining components of AdamW remain unchanged.

#### 3.3.2 From Lion to `Lion-PMA`

A similar modification applies to Lion. The vanilla moving average with dynamic scheduling replaces the EMA rule. Because Lion lacks a second moment estimate, the small step learning rate is decayed by $1/K$, not $1/\sqrt{K}$ as in AdamW. This aligns with the rescaled momentum at large update steps, ensuring a linearly decreasing learning rate as described above. All other Lion components are retained unchanged. Pseudocode for `Lion-PMA` is presented in Appendix A.

## 4 THEORETICAL ANALYSIS

In this section, we provide theoretical guarantees for `AdamW-PMA`, demonstrating that it achieves a convergence rate comparable to Adam, with similar resource overhead.

### 4.1 CONVERGENCE ANALYSIS

We analyze the convergence properties of `AdamW-PMA` under the same assumptions as Kingma and Ba [2014]. As a performance metric, we consider the *regret*:

$$R_\tau(T) = \sum_{t=1}^{T} \left[ f(x_{t,\tau}) - f(x^*) \right], \tag{2}$$

where $\tau$ denotes the index of the small update steps. This quantity measures the cumulative sub-optimality of the iterates $x_{t,\tau}$ compared to the global optimum $x^*$ over $T$ periods.

For the analysis, we modify the notational convention so that the index $\tau$ of the small update steps ranges from 1 to $K$. The large update step, corresponding to $\tau = K$, transitions from $x_{t,K}$ to $x_{t+1,1}$; the remaining cases $\tau \in [K-1]$ correspond to the small update steps. We define $g_{t,\tau} = \nabla f(x_{t,\tau})$, with $g_{t,\tau,i}$ denoting its $i$th component.

**Theorem 1.** *Suppose the objective $f$ is convex, with $\|\nabla f(x)\|_2 \le G$ and $\|\nabla f(x)\|_\infty \le G_\infty$ for all $x$. Assume that, for any $(t_1, \tau_1), (t_2, \tau_2) \in [T] \times [K]$, the parameter differences satisfy $\|x_{t_1,\tau_1} - x_{t_2,\tau_2}\|_2 \le D$ and $\|x_{t_1,\tau_1} - x_{t_2,\tau_2}\|_\infty \le D_\infty$. Further suppose the hyperparameters satisfy $\frac{\sqrt{1-\beta_2}}{1-\beta_1} \le 1$. Then, for any $T \ge 1$, `AdamW-PMA` achieves:*

$$\begin{aligned} R_K(T) \le & \frac{\sqrt{K} D^2}{2\gamma(1-\beta_1)} \sum_{i=1}^{d} \sqrt{T\, \hat{v}_{T,K,i}} \\ & + \frac{(1+\gamma) K^{3/2} G_\infty}{2(1-\beta_1)} \sum_{i=1}^{d} \|g_{1:KT,i}\|_2 \\ & + \frac{D_\infty^2 G_\infty (K-1)}{2(1-\beta_1)}. \end{aligned} \tag{3}$$

The proof of Theorem 1 is provided in Appendix C. This result establishes that `AdamW-PMA` attains a regret bound of $O(\sqrt{T})$, matching the order obtained for Adam [Kingma and Ba, 2014], and confirming its convergence guarantee in the convex setting.

Additionally, Theorem 1 reveals that, for fixed $T$, the cumulative regret increases as $K$ (the number of small steps) increases. This reflects a fundamental trade-off: larger $K$ can result in faster initial convergence, but overly large $K$ may cause the update trajectory to deviate from that of Adam, potentially leading to larger regret in the long term.

### 4.2 RESOURCE OVERHEAD ANALYSIS

We now discuss the computational and memory overhead of `AdamW-PMA`. Since `AdamW-PMA` is based on AdamW, we focus on a comparison with AdamW employing gradient accumulation.

`AdamW-PMA` does not incur any additional memory overhead compared to AdamW or AdamW with gradient accumulation. The memory required by `AdamW-PMA` consists of the storage for model parameters, gradients, and the running estimates of the first and second moments—identical to Adam and AdamW. When the same micro-batch size is used, the memory footprint of `AdamW-PMA` and AdamW with GA are equivalent.

## 5 EVALUATION

### 5.1 EXPERIMENTAL SETUP

**Tasks.** We evaluate `AdamW-PMA` and `Lion-PMA` on language modeling tasks, specifically supervised fine-tuning (SFT) and direct preference optimization (DPO) [Rafailov et al., 2023]. Additional experiments, including pre-training and learning rate scheduler ablation, are presented in Appendix E due to space constraints. Table 1 summarizes the experimental settings.

| Dataset | Model | Results |
|---|---|---|
| Alpaca | Phi-2-2.7B | Tab. 2 |
| DuReader_Robust | Llama2-7B-base | Fig. 8 |
| HH-RLHF-harmless | Phi2-2.7B | Fig. 3,5,6 |
| | Qwen1.5-0.5B | Fig. 6 |
| | Qwen2-0.5B | Fig. 4 |
| | GPT2-medium | Fig. 7 |

Table 1: Summary of Settings.

**Baselines.** We compare `AdamW-PMA` and `Lion-PMA` to AdamW and Lion, respectively. For each optimizer with period length $K$ and batch size $B$, we denote the settings as `AdamW-PMA`-$K$ and AdamW-$K$ (similarly for Lion). AdamW-$K$ refers to AdamW with $K$-step gradient accumulation, resulting in effective batch size $KB$. All optimizers within each comparison group share identical hyperparameters. For the `AdamW-PMA` and `Lion-PMA` groups, we set lr $= 2\text{e}{-6}$ and betas to $(0.9, 0.95)$ and $(0.95, 0.98)$, respectively, following Chen et al. [2023].

**Implementation.** Experiments are conducted on a server equipped with $8\times$ NVIDIA A40 GPUs (each with $48$GB memory). We use the Swift framework [Team, 2024b] and PyTorch [Paszke et al., 2019]. SFT experiments use $K = 8$, $B = 32$; DPO tasks use $K = 16$, $B = 16$.

**Metrics.** For SFT, we report validation loss and model performance on the MMLU benchmark [Hendrycks et al., 2020]. For DPO, we report validation loss and classification accuracy (distinguishing accepted from rejected responses). For classification, we regard a prediction as correct if the model assigns higher probability to the accepted response.

**Methodology of Comparison.** To compare optimizer efficiency, we adopt two metrics: (1) *data efficiency*, measured by the amount of training data processed to reach a target metric value, accounting for batch size differences; and (2) *FLOPs*, representing the computation required to reach a target metric value and reflecting training speed.

### 5.2 SUPERVISED FINE-TUNING

| Algorithm | Val Loss | MMLU(Zero-Shot) | | | | |
|---|---|---|---|---|---|---|
| | | Hums. | STEM | Social | Other | Avg. |
| AdamW-4 | 0.9212 | 15.4 | 28.3 | 26.7 | 24.3 | 24.4 |
| AdamW-8 | 0.9408 | **19.2** | 22.8 | 26.7 | 25.0 | 23.3 |
| AdamW-PMA-4 | 0.9352 | 16.9 | 22.8 | 25.0 | 22.7 | 21.9 |
| AdamW-PMA-8 | **0.9078** | 16.2 | **28.3** | **30.1** | **35.0** | **27.7** |
| Lion-4 | 0.9227 | 13.1 | 23.3 | 24.2 | 25.7 | 21.8 |
| Lion-8 | 0.9486 | **20.8** | 22.2 | 24.2 | 25.0 | **23.0** |
| Lion-PMA-4 | **0.9136** | 13.1 | **23.3** | **24.2** | 25.7 | 21.8 |
| Lion-PMA-8 | 0.9373 | 17.7 | 22.2 | 22.5 | **26.4** | 22.3 |

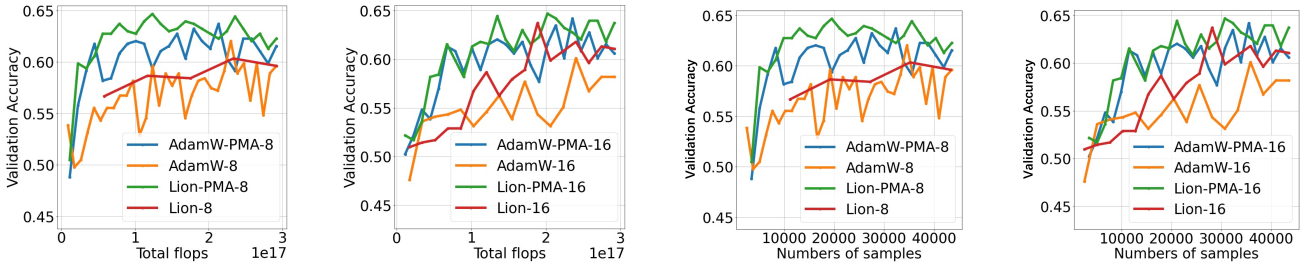
Table 2: Comparison of the validation loss and the performance after one epoch training on zero-shot MMLU for various algorithms. With limited space, we only demonstrate four representative categories.

Table 2 shows that `AdamW-PMA` and `Lion-PMA` improve SFT performance compared to their baseline counterparts. For validation loss, AdamW-PMA-8 achieves the best result among AdamW variants, while Lion-PMA-4 is best in the Lion family.

For MMLU zero-shot tasks, PMA variants outperform their EMA counterparts in all categories except Humanities, attributable to input length limitations on Phi-2. PMA-based optimizers consistently achieve higher average scores; e.g., AdamW-PMA-8 obtains an average of 27.7, outperforming AdamW.

### 5.3 DIRECT PREFERENCE OPTIMIZATION

**PMA improves accuracy and reduces overfitting.** Figure 3 compares the validation accuracy across flops and training samples for $K = 8$ and $K = 16$. AdamW is consistently the slowest and least accurate, while PMA-based optimizers yield marked improvements in both convergence speed and final accuracy. With larger $K$, AdamW-PMA achieves convergence speeds that surpass AdamW and rival Lion, while Lion-PMA delivers the strongest overall performance.




(a) Validation accuracy on flops (K = 8)

(b) Validation accuracy on flops (K = 16)

(c) Validation accuracy on number of samples (K=8)

(d) Validation accuracy on number of samples (K=16)

Figure 3: The accuracy of classifying the accepted and rejected responses on the validation dataset for DPO task. Compared to AdamW and Lion, AdamW-PMA and Lion-PMA exhibit faster convergence rates and higher accuracy.

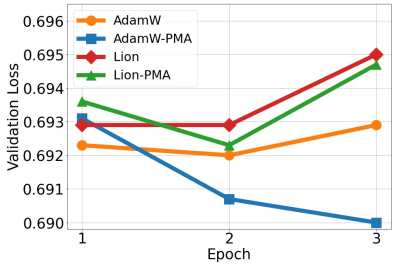



Figure 4: Validation loss of training more epochs on DPO task.

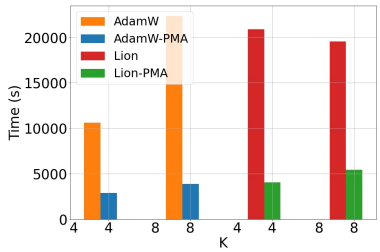



Figure 5: Runtime to achieve the same loss on DPO task..

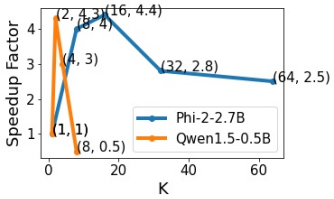



Figure 6: The speedup factor of AdamW-PMA compared to AdamW under different K.

Figure 4 presents validation loss across multiple epochs for Qwen2-0.5B on DPO. PMA-based optimizers sustain lower validation loss and are less prone to overfitting compared to EMA-based counterparts.

**PMA reduces runtime.** Figure 5 demonstrates that PMA-based optimizers reach a given loss target in less wall-clock time than EMA-based optimizers. Furthermore, PMA can utilize remaining data to achieve even higher accuracy once that target is met.

**PMA is sensitive to $K$.** We study the effect of $K$ on speedup in DPO experiments with both Phi-2 and Qwen1.5 models (see Fig. 6). For small $K$, PMA approximates standard AdamW, and variance reduction is minimal. For excessively large $K$, aggressive learning rate reduction can diminish acceleration. In practice, an intermediate $K$ achieves the best trade-off between speed and stability.

### 5.4 ADDITIONAL PROPERTIES

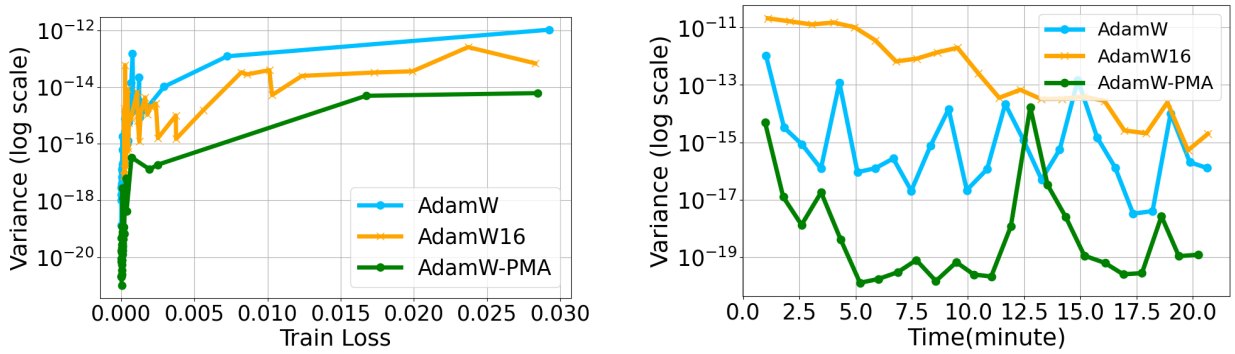
**PMA reduces update variance.** We measure update variance using GPT-2 medium on the Alpaca dataset with $K = 16$, comparing algorithms with identical configurations. Following [Ash et al., 2019, Mirzasoleiman et al., 2020, Killamsetty et al., 2021], we use the last layer gradient as a proxy for model gradient variance. Figure 7 demonstrates that PMA yields consistently lower update variance for the same loss and over training time, compared to EMA.

**PMA scales to large models.** On Llama2-7B trained with SFT on DuReader_Robust [Tang et al., 2020], AdamW-PMA consistently achieves lower validation loss than AdamW throughout training (see Fig. 8), demonstrating effectiveness at scale.

**PMA incurs minimal extra per-step overhead.** Although PMA introduces more frequent updates and slightly more communication in distributed settings, per-iteration time increases are modest: on a 7B model, AdamW-PMA is only about $2\%$ slower per iteration than AdamW.

## 6 RELATED WORK

**First-Order Adaptive Optimizers.** First-order adaptive methods form the backbone of modern deep learning optimization. Algorithms such as AdaGrad [Duchi et al., 2011] adapt learning rates for each parameter based on historical gradient information, assigning larger updates to infrequent features. RMSProp [Hinton et al., 2012] extends AdaGrad by maintaining an exponential moving average of squared gradients. Adam [Kingma and Ba, 2014], which further introduces an EMA of the first moment, and its variant with decoupled weight decay, AdamW [Loshchilov and




(a) Variance vs Training loss

(b) Variance vs Time

Figure 7: Comparison of the magnitude in variance w.r.t the training loss and time.

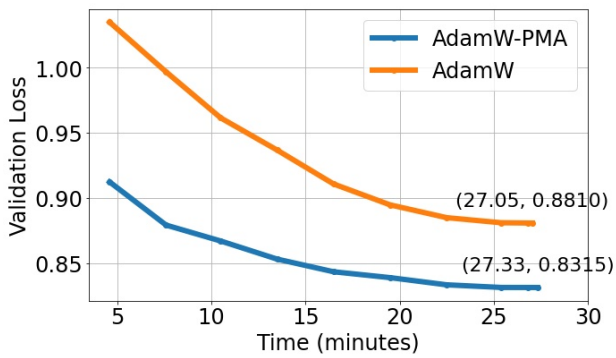



Figure 8: Validation loss of SFT on Llama2-7B.AdamW-PMA consistently attains a much lower loss than AdamW.

Hutter, 2017], are now the predominant optimizers for large-scale neural networks, especially Transformers [Vaswani et al., 2017]. Numerous subsequent methods build on this foundation, including variants such as AdaFactor [Shazeer and Stern, 2018], Adam with Nesterov momentum [Dozat, 2016], Adabelief [Zhuang et al., 2020], Adan [Xie et al., 2022], Lion [Chen et al., 2023], and GrAMS [Cao et al., 2024]. Despite their effectiveness, these algorithms often incur high memory costs due to storing additional first and second moment estimates, posing significant challenges for training large models on memory-limited devices.

**Memory-Efficient Optimizers.** To address memory bottlenecks, several optimizers have been proposed. AdaFactor [Shazeer and Stern, 2018] approximates the second-moment matrix using row and column factors. LOMO [Lv et al., 2023] streamlines the update and gradient computation to reduce transient storage. CAME [Luo et al., 2023] employs residual-based adaptive updating, and GaLore [Zhao et al., 2024] applies low-rank gradient projections to save memory. Adam-mini [Zhang et al., 2024] further trims learning rate-related state in Adam. However, most of these methods entail a trade-off, sacrificing convergence rates or providing only moderate memory reduction (e.g., CAME saves $12.1\%$ over Adam per Luo et al. [2023]), which remains inadequate for training LLMs under stringent memory constraints. Thus, techniques like gradient accumulation (GA) remain indispensable in memory-limited settings, further motivating acceleration of these approaches.

**Variance Reduction.** Variance reduction techniques in SGD are essential for accelerating convergence. Among these, dynamic sampling, gradient aggregation, and iterate averaging have received particular attention [Bottou et al., 2018]. SVRG [Johnson and Zhang, 2013] and SAGA [Defazio et al., 2014] leverage historical gradient or parameter states to construct lower-variance stochastic gradient estimators, albeit at increased memory or computational costs. Iterate averaging [Polyak, 1991] returns the average of parameters across SGD iterates; Nesterov's accelerated techniques [Nesterov, 2013] further establish $O(1/t)$ convergence for such schemes. However, the practical application of these methods to LLMs is limited by their overhead in storage or per-iteration computation.

## 7 CONCLUSION

We tackled the challenge of high-variance stochastic optimization for post-training large language models (LLMs) on GPU-memory-constrained devices. Our analysis revealed that the slow convergence of existing momentum-based optimizers is largely attributable to the EMA scheme, which inadequately exploits information from historical gradients to stabilize parameter updates. To remedy this, we introduced `PMA`, a novel momentum update framework that partitions training into periods and applies a moving average within each period. We integrated `PMA` into AdamW and Lion to obtain `AdamW-PMA` and `Lion-PMA`, respectively. Extensive experiments on SFT and DPO tasks across various models demonstrate that `PMA` yields at least a $2\times$ speedup in training and consistently improves downstream task performance.

## LIMITATIONS

While PMA offers substantial training acceleration, it can incur increased communication overhead in multi-GPU settings, particularly for large $K$. Specifically, since AdamW-PMA performs $K$ times as many parameter update communications as AdamW with GA, its communication cost can scale linearly with $K$. In addition, we have not yet evaluated PMA in large-scale distributed experiments; further investigation in such settings is warranted.

## ACKNOWLEDGEMENT

This work was supported by the Shenzhen Science and Technology Program (JCYJ20220818103001002), Shenzhen Doctoral Startup Funding (RCBS20221008093330065), Tianyuan Fund for Mathematics of National Natural Science Foundation of China (NSFC) (12326608), Shenzhen Science and Technology Program (Shenzhen Key Laboratory Grant No. ZDSYS20230626091302006), and Shenzhen Stability Science Program 2023, Shenzhen Key Lab of Multi-Modal Cognitive Computing.

### References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Jordan T Ash, Chicheng Zhang, Akshay Krishnamurthy, John Langford, and Alekh Agarwal. Deep batch active learning by diverse, uncertain gradient lower bounds. *arXiv preprint arXiv:1906.03671*, 2019.

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM review*, 60(2):223–311, 2018.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *NeurIPS*, 33:1877–1901, 2020.

Yang Cao, Xiaoyu Li, and Zhao Song. Grams: Gradient descent with adaptive momentum scaling. *arXiv preprint arXiv:2412.17107*, 2024.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, et al. Symbolic discovery of optimization algorithms. *arXiv preprint arXiv:2302.06675*, 2023.

Ashok Cutkosky and Francesco Orabona. Momentum-based variance reduction in non-convex sgd. *NeurIPS*, 32, 2019.

Aaron Defazio, Francis R. Bach, and Simon Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *NIPS*, pages 1646–1654, 2014.

Timothy Dozat. Incorporating nesterov momentum into adam. *ICLR Workshop*, 2016.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Jingwen Fu, Bohan Wang, Huishuai Zhang, Zhizheng Zhang, Wei Chen, and Nanning Zheng. When and why momentum accelerates sgd: An empirical study. *arXiv preprint arXiv:2306.09000*, 2023.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Mojan Javaheripi, Sébastien Bubeck, et al. Phi-2: The surprising power of small language models. *URL https://www. microsoft. com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models*, 2023.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*, pages 315–323, 2013.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Krishnateja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, and Rishabh Iyer. Glister: Generalization based data subset selection for efficient and robust learning. In *AAAI*, volume 35, pages 8110–8118, 2021.

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

Frederik Kunstner, Jacques Chen, Jonathan Wilder Lavington, and Mark 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.

Kevin Lee and Shubho Sengupta. Introducing the ai research supercluster metas cutting-edge ai supercomputer for ai research, 2022. URL https://ai.meta.com/blog/ai-rsc/.

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

Yang Luo, Xiaozhe Ren, Zangwei Zheng, Zhuo Jiang, Xin Jiang, and Yang You. Came: Confidence-guided adaptive memory efficient optimization. *arXiv preprint arXiv:2307.02047*, 2023.

Kai Lv, Yuqing Yang, Tengxiao Liu, Qinghui Gao, Qipeng Guo, and Xipeng Qiu. Full parameter fine-tuning for large language models with limited resources. *arXiv preprint arXiv:2306.09782*, 2023.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *ICML*, pages 6950–6960, 2020.

Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

Lam M Nguyen, Jie Liu, Katya Scheinberg, and Martin Takáč. Sarah: A novel method for machine learning problems using stochastic recursive gradient. In *ICML*, pages 2613–2621, 2017.

OpenAI. Learning to reason with llms. *https://openai.com/index/learning-to-reason-with-llms/*, 2024.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 32, 2019.

Hieu Pham, Zihang Dai, Golnaz Ghiasi, Kenji Kawaguchi, Hanxiao Liu, Adams Wei Yu, Jiahui Yu, Yi-Ting Chen, Minh-Thang Luong, Yonghui Wu, et al. Combined scaling for zero-shot transfer learning. *Neurocomputing*, 555: 126658, 2023.

Boris Polyak. New method of stochastic approximation type. *Autom. Remote Control*, 7:937–946, 01 1991.

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

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *ICML*, pages 4596–4604, 2018.

Hongxuan Tang, Hongyu Li, Jing Liu, Yu Hong, Hua Wu, and Haifeng Wang. Dureader_robust: A chinese dataset towards evaluating robustness and generalization of machine reading comprehension in real-world applications. *arXiv preprint arXiv:2004.11142*, 2020.

Qwen Team. Introducing qwen1.5, February 2024a. URL https://qwenlm.github.io/blog/qwen1.5/.

The ModelScope Team. Swift:scalable lightweight infrastructure for fine-tuning. https://github.com/modelscope/swift, 2024b.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

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

Xingyu Xie, Pan Zhou, Huan Li, Zhouchen Lin, and Shuicheng Yan. Adan: Adaptive nesterov momentum algorithm for faster optimizing deep models. *arXiv preprint arXiv:2208.06677*, 2022.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

Kun Yuan, Bicheng Ying, and Ali H Sayed. On the influence of momentum acceleration on online learning. *Journal of Machine Learning Research*, 17(192):1–66, 2016.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. Adam-mini: Use fewer learning rates to gain more. *arXiv preprint arXiv:2406.16793*, 2024.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024.

Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar C Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *NeurIPS*, 33: 18795–18806, 2020.

# Perodical Moving Average Accelerates Gradient Accumulation for Post-Training (Supplementary Material)

**Yumou Liu**[§1] **An Li**[1] **Chaojie Li**[1] **Fei Yu**[1] **Benyou Wang**[¶1]

[1]School of Data Science
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China

## A EXTRA PSEUDO-CODE

**Algorithm 2:** Lion-PMA

1 **for** $t = 1 \rightarrow \ldots$ **do**
2 $\quad g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$;
3 $\quad \tau \leftarrow t\%K$;
4 $\quad$ **if** $\tau = 0$ *and* $t > 0$ **then**
5 $\quad\quad \gamma_t \leftarrow \gamma$;
6 $\quad\quad u_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t / K$;
7 $\quad\quad u_t \leftarrow \mathrm{sign}(u_t)$;
8 $\quad$ **else**
9 $\quad\quad \gamma_t \leftarrow \gamma / K$;
10 $\quad\quad u_t \leftarrow \frac{\tau}{\tau+1} m_t + \frac{1-\beta_1}{\tau+1} g_t$;
11 $\quad\quad u_t \leftarrow \mathrm{sign}(u_t)$;
12 $\quad\quad m_t \leftarrow \frac{\tau}{\tau+1} v_t + \frac{1-\beta_2}{\tau+1} g_t^2$;
13 $\quad \hat{\theta}_t \leftarrow (1-\gamma_t \lambda)\theta_{t-1}$;
14 $\quad \theta_t = \hat{\theta}_t - \gamma_t u_t$;
15 $\quad$ **if** $\tau = 0$ *and* $t > 0$ **then**
16 $\quad\quad \hat{m}_t \leftarrow K \hat{m}_t$;
17 **return** $\theta_t$;

Algorithm 2 presents the pseudo-code of `Lion-PMA`. Lines 4-8 illustrate the large update steps, while lines 9-13 demonstrate the small update steps. We incorporate `Lion-PMA` as an adaptation of `AdamW-PMA` to AdamW. The main difference between the adaptation of AdamW and Lion lies in the implementation of the learning rate strategy. In `Lion-PMA`, we decay the learning rate by $1/K$ at the small update steps instead of $1/\sqrt{K}$ in `AdamW-PMA`.

## B ADDITIONAL THEORETICAL ANALYSIS

In this section, we provide a theoretical analysis on the convergence property of `AdamW-PMA`. Specifically, we focus on the convergence properties concerning the number of large update steps. This focus is due to the time cost between two large steps being approximately equal to the time between two updates of Adam with GA. During the analysis, we slightly modify the notations for ease of analysis. Unlike Algorithm 1, where the index of small update steps ranges from 0 to $K-1$, in the subsequent analysis, this index ranges from 1 to $K$. Specifically, when $\tau = K$, the update step from $x_{t,K}$ to $x_{t+1,1}$ is considered a large update step for all $t$. For the other $\tau \in [K-1]$, the subsequent update step is a small step.

[*]yumouliu@link.cuhk.edu.cn
[†]Benyou is the corresponding author
[§]yumouliu@link.cuhk.edu.cn
[¶]Benyou is the corresponding author

Firstly, we can show the average regret of `AdamW-PMA` converges based on Theorem 1,

**Corollary 1.** *Assume that the optimization objective $f$ is convex and has bounded gradients, $\|\nabla f(x)\|_2 \leq G$, $\|\nabla f(x)\|_\infty \leq G_\infty$, and the distance between any parameter generated by `AdamW-PMA` is bounded, $\|x_{t_1,\tau_1} - x_{t_2,\tau_2}\|_2 \leq D$, $\|x_{t_1,\tau_1} - x_{t_2,\tau_2}\|_\infty \leq D_\infty$ for any $t_1, t_2 \in [T]$ and $\tau_1, \tau_2 \in [K]$, and $\beta_1$, $\beta_2$ satisfy $\frac{\sqrt{1-\beta_2}}{1-\beta_1} \leq 1$. `AdamW-PMA` achieves the following regret guarantee, for all $T \geq 1$.*

$$\frac{R_K(T)}{T} = O\left(\frac{1}{\sqrt{T}}\right).$$

Then, we provide the update size between two large update steps in general non-convex settings.

**Theorem 2.** *Assume that the objective function $f$ is $L$-smooth, the step size between two large update steps is bounded by*

$$\|x_{t+1,1} - x_{t,1}\|^2 \leq \frac{2}{L}\left(1 + \frac{\gamma^2 L}{\sqrt{K}} \frac{(1-\beta_1)^2}{1-\beta_2}(K+1)\right) \cdot \bar{\zeta}(2a)^{2t-2}, \tag{4}$$

*where $\bar{\zeta}$ and $a$ are constants, and $\bar{\zeta}(2a)^{2t-2} \geq \zeta(2a)^{2t-2} + \frac{c}{1-4a^2} + K^2$, and $a = \frac{\beta_1(1-\beta_1)}{\sqrt{\beta_2(1-\beta_2)}} \cdot \frac{1}{\sqrt{K}}$.*

Theorem 2 indicates that the distance between two large update steps is bounded and converges to 0. Despite having $K$ small updates with varying momentum averaging weights, the step sizes still converge rapidly , suggesting the validity of setting the learning rate of the small steps to be $\gamma/\sqrt{K}$. Furthermore, the exponential term decreases with $K$, aligning with the intuition that more small update steps lead to faster convergence.

## C PROOF OF THEOREM 2

Before the analysis, we slightly modify the notations to simplify the analysis. The large step update takes the $x_{t,K}$ as input and outputs $x_{t+1,1}$. Then, `AdamW-PMA`uses small step update to obtain $x_{t+1,2}, \ldots, x_{t+1,K}$. It is noteworthy that the indexes of small step updates in Algorithm 1 range from 0 to $K-1$, while in the following analysis, they will range from 1 to $K$.

Before the analysis, we start with some important lemmas. Firstly, we consider the size of small step updates between two large step updates. To start with, we bound the size of every small step update using Lemma 1.

**Lemma 1.** *When $\tau \geq 2$,*

$$\left\|\frac{m_{t,\tau}}{\sqrt{v_{t,\tau}}}\right\| \leq \frac{1-\beta_1}{\sqrt{1-\beta_2}} \cdot \frac{1}{\sqrt{\tau}}\left(\left\|\frac{m_{t,1}}{\sqrt{v_{t,1}}}\right\| + \tau - 1\right). \tag{5}$$

*Proof.*

$$\begin{aligned}
\left\|\frac{m_{t,\tau}}{\sqrt{v_{t,\tau}}}\right\| &= \left\|\frac{\frac{\tau-1}{\tau} m_{t,\tau-1} + \frac{1-\beta_1}{\tau} g_{t,\tau-1}}{\sqrt{\frac{\tau-1}{\tau} v_{t,\tau-1} + \frac{1-\beta_2}{\tau} g_{t,\tau-1}^2}}\right\| \\
&\leq \frac{1-\beta_1}{\sqrt{1-\beta_2}} \cdot \frac{1}{\sqrt{\tau}} \cdot \left\|\frac{m_{t,1} + \sum_{\sigma=2}^{\tau} g_{t,\sigma}}{\sqrt{v_{t,1} + \sum_{\sigma=2}^{\tau} g_{t,\sigma}^2}}\right\| \\
&\leq \frac{1-\beta_1}{\sqrt{1-\beta_2}} \cdot \frac{1}{\sqrt{\tau}} \left(\sum_{\sigma=2}^{\tau}\left\|\frac{g_{t,\sigma}}{\sqrt{v_{t,1} + \sum_{\rho=2}^{\tau} g_{t,\rho}^2}}\right\| + \left\|\frac{m_{t,1}}{\sqrt{v_{t,1} + \sum_{\sigma=2}^{\tau} g_{t,\sigma}^2}}\right\|\right) \\
&\leq \frac{1-\beta_1}{\sqrt{1-\beta_2}} \cdot \frac{1}{\sqrt{\tau}} \left(\left\|\frac{m_{t,1}}{\sqrt{v_{t,1}}}\right\| + \sum_{\sigma=2}^{\tau}\left\|\frac{g_{t,\sigma}}{\sqrt{g_{t,\sigma}^2}}\right\|\right) \\
&= \frac{1-\beta_1}{\sqrt{1-\beta_2}} \cdot \frac{1}{\sqrt{\tau}} \left(\left\|\frac{m_{t,1}}{\sqrt{v_{t,1}}}\right\| + \tau - 1\right)
\end{aligned}$$

□

Then, we bound the squared size of the small step update.

**Corollary 2.**

$$\left\| \frac{m_{t,\tau}}{\sqrt{v_{t,\tau}}} \right\|^2 \leq \frac{(1-\beta_1)^2}{1-\beta_2} \cdot \frac{2}{\tau} \left( \left\| \frac{m_{t,1}}{\sqrt{v_{t,1}}} \right\|^2 + \tau^2 \right). \tag{6}$$

*Proof.* Since $(a+b)^2 = a^2 + b^2 + 2ab \leq 2(a^2+b^2)$,

$$\begin{aligned} \left\| \frac{m_{t,\tau}}{\sqrt{v_{t,\tau}}} \right\| &\leq \frac{1-\beta_1}{\sqrt{1-\beta_2}} \cdot \frac{1}{\sqrt{\tau}} \left( \left\| \frac{m_{t,1}}{\sqrt{v_{t,1}}} \right\| + \tau - 1 \right) \\ &\leq \frac{(1-\beta_1)^2}{1-\beta_2} \cdot \frac{2}{\tau} \left( \left\| \frac{m_{t,1}}{\sqrt{v_{t,1}}} \right\|^2 + \tau^2 \right). \end{aligned}$$

□

Then, we bound the sum of the squared size of small step updates between two large step updates.

**Corollary 3.**

$$\sum_{\sigma=1}^{\tau-1} \left\| \frac{m_{t,\sigma}}{\sqrt{v_{t,\sigma}}} \right\|^2 \leq \frac{(1-\beta_1)^2}{1-\beta_2} \sum_{\sigma=1}^{\tau-2} \frac{2}{\sigma} \left( \left\| \frac{m_{t,1}}{\sqrt{v_{t,1}}} \right\|^2 + \sigma^2 \right) + \left\| \frac{m_{t,1}}{\sqrt{v_{t,1}}} \right\|^2. \tag{7}$$

*Proof.*

$$\begin{aligned} \sum_{\sigma=1}^{\tau-1} \left\| \frac{m_{t,\sigma}}{\sqrt{v_{t,\sigma}}} \right\|^2 &= \sum_{\sigma=1}^{\tau-2} \left\| \frac{m_{t,\sigma}}{\sqrt{v_{t,\sigma}}} \right\|^2 + \left\| \frac{m_{t,\tau-1}}{\sqrt{v_{t,\tau-1}}} \right\|^2 \\ &\overset{\text{Corollary 2}}{\leq} \sum_{\sigma=1}^{\tau-2} \left\| \frac{m_{t,\sigma}}{\sqrt{v_{t,\sigma}}} \right\|^2 + \frac{(1-\beta_1)^2}{1-\beta_2} \cdot \frac{2}{\tau-1} \cdot \left( \left\| \frac{m_{t_1}}{\sqrt{v_{t,1}}} \right\|^2 + (\tau-1)^2 \right) \\ &\leq \frac{(1-\beta_1)^2}{1-\beta_2} \sum_{\sigma=1}^{\tau-2} \frac{2}{\sigma} \left( \left\| \frac{m_{t,1}}{\sqrt{v_{t,1}}} \right\|^2 + \sigma^2 \right) + \left\| \frac{m_{t,1}}{\sqrt{v_{t,1}}} \right\|^2. \end{aligned}$$

□

After bounding the small steps between two large update steps, we consider the size of a large update step and $K$ following small steps. To start with, we assume that the update size of the first large step is bounded.

**Assumption 1.** *Let* $a = \frac{\beta_1(1-\beta_1)}{\sqrt{\beta_2(1-\beta_2)}} \cdot \frac{1}{\sqrt{K}}$ *and* $b = \frac{1-\beta_1}{\sqrt{1-\beta_2}} \cdot \frac{1}{\sqrt{K}} \cdot \left(1 + \frac{\beta_1 K}{\sqrt{\beta_2}}\right)$.

$$\left\| \frac{m_{1,1}}{\sqrt{v_{1,1}}} \right\| \leq \frac{b}{1-a} + \alpha. \tag{8}$$

Then, we make some assumptions on the weight of the momentum.

**Assumption 2.** *For all t,*

$$\frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} \leq 1.$$

If we take $\beta_1 = 0.9$, $\beta_2 = 0.99$ as the default configuration of Adam, this assumption holds.

Then, we bound the size of a large update step.

**Lemma 2.** *By tuning the hyper-parameters* $\beta_1$ *and* $\beta_2$*, let* $a \leq 1/2$*. Then*

$$\left\| \frac{m_{t,1}}{\sqrt{v_{t,1}}} \right\| \leq \bar{\alpha} \cdot a^{t-1}, \tag{9}$$

*where* $\bar{\alpha} > \alpha$ *is a constant to make* $\bar{\alpha} \cdot a^{t-1} \geq \alpha \cdot a^{t-1} + \frac{b}{1-a} + K$.

*Proof.*

$$\begin{aligned}
\left\|\frac{m_{t,1}}{\sqrt{v_{t,1}}}\right\| &= \left\|\frac{\beta_1 m_{t-1,K} + \frac{1-\beta_1}{K} g_{t,1}}{\sqrt{\beta_2 v_{t-1,K} + \frac{1-\beta_2}{K} g_{t,1}^2}}\right\| \\
&\leq \left\|\frac{\beta_1 m_{t-1,K}}{\sqrt{\beta_2 v_{t-1,K}}}\right\| + \frac{1-\beta_1}{\sqrt{1-\beta_2}} \cdot \frac{1}{\sqrt{K}} \cdot \left\|\frac{g_{t,1}}{\sqrt{g_{t,1}^2}}\right\| \\
&= \left\|\frac{\beta_1 m_{t-1,K}}{\sqrt{\beta_2 v_{t-1,K}}}\right\| + \frac{1-\beta_1}{\sqrt{1-\beta_2}} \cdot \frac{1}{\sqrt{K}} \\
&\overset{\text{Lem. 1}}{\leq} \frac{1-\beta_1}{\sqrt{1-\beta_2}} \cdot \frac{1}{\sqrt{K}} + \frac{\beta_1}{\sqrt{\beta_2}} \cdot \frac{1-\beta_1}{\sqrt{1-\beta_2}} \cdot \frac{1}{\sqrt{K}} \cdot \left(\left\|\frac{m_{t-1,1}}{\sqrt{v_{t-1,1}}}\right\| + K - 1\right) \\
&\leq \underbrace{\frac{\beta_1(1-\beta_1)}{\sqrt{\beta_2(1-\beta_2)}} \cdot \frac{1}{\sqrt{K}}}_{:=a} \cdot \left\|\frac{m_{t-1,1}}{\sqrt{v_{t-1,1}}}\right\| + \underbrace{\frac{1-\beta_1}{\sqrt{1-\beta_2}} \cdot \frac{1}{\sqrt{K}} \cdot \left(1 + \frac{\beta_1 K}{\sqrt{\beta_2}}\right)}_{:=b}
\end{aligned}$$

Let $x_t = \left\|\frac{m_{t,1}}{\sqrt{v_{t,1}}}\right\|$, then

$$x_t \leq a^{t-1}\left(x_1 - \frac{b}{1-a}\right) + \frac{b}{1-a}$$

Then, by Assumption 1

$$\left\|\frac{m_{t,1}}{\sqrt{v_{t,\tau}}}\right\| \leq \alpha \cdot a^{t-1} + \frac{b}{1-a} \leq \bar{\alpha} \cdot a^{t-1}.$$

□

Similar to the above approach, we assume the bounded squared first large step and prove the bounded squared large steps.

**Assumption 3.** *Let* $c = \frac{(1-\beta_1)^2}{1-\beta_2} \cdot \frac{2}{K}\left(1 + \frac{2\beta_1^2 K^2}{\beta_2^2}\right)$

$$\left\|\frac{m_{1,1}}{\sqrt{v_{t,1}}}\right\|^2 \leq \frac{c}{1-4a^2} + \zeta \tag{10}$$

**Corollary 4.**

$$\left\|\frac{m_{t,1}}{\sqrt{v_{t,1}}}\right\|^2 \leq \bar{\zeta}(2a)^{2t-2}, \tag{11}$$

*where* $\bar{\zeta} > \zeta$ *is a constant to make* $\bar{\zeta}(2a)^{2t-2} \geq \zeta(2a)^{2t-2} + \frac{c}{1-4a^2} + K^2$.

*Proof.*

$$\begin{aligned}
\left\|\frac{m_{t,1}}{\sqrt{v_{t,1}}}\right\|^2 &\leq 2\left\|\frac{\beta_1 m_{t-1,K}}{\sqrt{\beta_2 v_{t-1,K}}}\right\|^2 + \frac{(1-\beta_1)^2}{1-\beta_2} \cdot \frac{2}{K} \\
&\leq \frac{(1-\beta_1)^2}{1-\beta_2} \cdot \frac{2}{K} + \frac{4}{K} \cdot \frac{(1-\beta_1)^2\beta_1^2}{(1-\beta_2)\beta_2} \cdot \left(\left\|\frac{m_{t-1,1}}{\sqrt{v_{t-1,1}}}\right\|^2 + K^2\right) \\
&= 4a^2\left\|\frac{m_{t-1,1}}{\sqrt{v_{t-1,1}}}\right\|^2 + c.
\end{aligned}$$

Then,

$$\begin{aligned}
\left\|\frac{m_{t,1}}{\sqrt{v_{t,1}}}\right\|^2 &\leq (2a)^{2t-2}\left(\left\|\frac{m_{1,1}}{\sqrt{v_{1,1}}}\right\|^2 - \frac{c}{1-4a^2}\right) + \frac{c}{1-4a^2} \\
&\leq \zeta(2a)^{2t-2} + \frac{c}{1-4a^2} \\
&\leq \bar{\zeta}(2a)^{2t-2}.
\end{aligned}$$

□

Before proving Theorem 2, we need more assumptions on the objective function and the initial point. First, we assume that $f$ has Lipschitz continuous gradient.

**Assumption 4** ($L$-smoothness)**.** *A function $f\colon \mathbb{R}^d \to \mathbb{R}$ is differentiable and for any $x_1, x_2 \in \mathbb{R}^d$,*

$$\|\nabla f(x_1) - \nabla f(x_2)\| \le L\|x_1 - x_2\|,$$

*where $L$ is a constant.*

Now, by putting everything together, we are ready to prove Theorem 2.

*Proof of Theorem 2.* Let $\mathcal{T} = \frac{L}{2}\|x_{t+1,1} - x_t\|^2$. At the beginning, we assume that the `AdamW-PMA` does not employ the bias correction shown in Line 12-13 in Algorithm 1.

$$\begin{aligned}
\mathcal{T} &= \frac{L}{2}\|x_{t+1,1} - x_{t.1}\|^2 \\
&\le \frac{L}{2}\sum_{\tau=1}^{K-1}\|x_{t,\tau+1} - x_{t,\tau}\|^2 + \frac{L}{2}\|x_{t+1,1} - x_{t,K}\|^2 \\
&= \frac{L}{2}\sum_{\tau=1}^{K-1}\left\|\frac{\gamma}{\sqrt{K}}\frac{\hat{m}_{t,\tau}}{\sqrt{\hat{v}_{t,\tau}}}\right\|^2 + \frac{L}{2}\left\|\gamma\cdot\frac{\hat{m}_{t,K}}{\sqrt{\hat{v}_{t,K}}}\right\|^2 \\
&= \frac{\gamma^2 L}{2\sqrt{K}}\sum_{\tau=1}^{K-1}\left\|\frac{m_{t,\tau}}{\sqrt{v_{t,\tau}}}\right\|^2 + \frac{\gamma^2 L\sqrt{K}}{2}\left\|\frac{m_{t,K}}{\sqrt{v_{t,K}}}\right\|^2.
\end{aligned}$$

$$\begin{aligned}
\mathcal{T} &\le \frac{\gamma^2 L}{2\sqrt{K}}\sum_{\tau=1}^{K-1}\left\|\frac{m_{t,\tau}}{\sqrt{v_{t,\tau}}}\right\|^2 + \frac{\gamma^2 L\sqrt{K}}{2}\left\|\frac{m_{t,K}}{\sqrt{v_{t,K}}}\right\|^2 \\
&\overset{\text{Corollary 2}}{\le} \frac{\gamma^2 L}{2\sqrt{K}}\sum_{\tau=1}^{K-1}\left\|\frac{m_{t,\tau}}{\sqrt{v_{t,\tau}}}\right\|^2 + \frac{\gamma^2 L}{2}\frac{(1-\beta_1)^2}{1-\beta_2}\cdot\frac{2}{\sqrt{K}}\left(\left\|\frac{m_{t,1}}{\sqrt{v_{t,1}}}\right\|^2 + K^2\right) \\
&= \frac{\gamma^2 L}{2\sqrt{K}}\sum_{\tau=1}^{K-1}\left\|\frac{m_{t,\tau}}{\sqrt{v_{t,\tau}}}\right\|^2 + \frac{\gamma^2 L}{\sqrt{K}}\cdot\frac{(1-\beta_1)^2}{1-\beta_2}\cdot\left(\left\|\frac{m_{t,1}}{\sqrt{v_{t,1}}}\right\|^2 + K^2\right) \\
&\overset{\text{Corollary 3}}{\le} \frac{\gamma^2 L}{2\sqrt{K}}\frac{(1-\beta_1)^2}{1-\beta_2}\sum_{\sigma=1}^{K-2}\frac{2}{\sigma}\left(\left\|\frac{m_{t,1}}{\sqrt{v_{t,1}}}\right\|^2 + \sigma^2\right) + \left\|\frac{m_{t,1}}{\sqrt{v_{t,1}}}\right\|^2 + \frac{\gamma^2 L}{\sqrt{K}}\cdot\frac{(1-\beta_1)^2}{1-\beta_2}\cdot\left(\left\|\frac{m_{t,1}}{\sqrt{v_{t,1}}}\right\|^2 + K^2\right) \\
&\le \frac{\gamma^2 L}{\sqrt{K}}\frac{(1-\beta_1)^2}{1-\beta_2}\left(K\cdot\left\|\frac{m_{t,1}}{\sqrt{v_{t,1}}}\right\|^2 + K^2\right) + \left(1 + \frac{\gamma^2 L}{\sqrt{K}}\frac{(1-\beta_1)^2}{1-\beta_2}\right)\left\|\frac{m_{t,1}}{\sqrt{v_{t,1}}}\right\|^2 + \gamma^2 L K^{\frac{3}{2}}\frac{(1-\beta_1)^2}{1-\beta_2} \\
&= \left(1 + \frac{\gamma^2 L}{\sqrt{K}}\frac{(1-\beta_1)^2}{1-\beta_2}(K+1)\right)\cdot\left\|\frac{m_{t,1}}{\sqrt{v_{t,1}}}\right\|^2 + 2\gamma^2 L K^{\frac{3}{2}}\frac{(1-\beta_1)^2}{1-\beta_2} \\
&\overset{\text{Corollary 4}}{\le} \left(1 + \frac{\gamma^2 L}{\sqrt{K}}\frac{(1-\beta_1)^2}{1-\beta_2}(K+1)\right)\cdot\bar{\zeta}(2a)^{2t-2} + 2\gamma^2 L K^{\frac{3}{2}}\frac{(1-\beta_1)^2}{1-\beta_2} \\
&\overset{\text{Larger }\bar{\zeta}}{\le} \left(1 + \frac{\gamma^2 L}{\sqrt{K}}\frac{(1-\beta_1)^2}{1-\beta_2}(K+1)\right)\cdot\bar{\zeta}(2a)^{2t-2}.
\end{aligned}$$

Thus,

$$\|x_{t+1,1} - x_{t,1}\|^2 \le \frac{2}{L}\left(1 + \frac{\gamma^2 L}{\sqrt{K}}\frac{(1-\beta_1)^2}{1-\beta_2}(K+1)\right)\cdot\bar{\zeta}(2a)^{2t-2}.$$

Then, we consider the bias correction shown in Linw 12-13 in Algorithm 1. By the bias correction, the learning rate at time step $t$ can be viewed as $\gamma_t = \frac{\sqrt{1-\beta_2^t}}{1-\beta_1}\gamma \le \gamma$ by Assumption 2. Then, with the bias correction operation, this bound still holds.

□

## D PROOF OF THEOREM 1

Before the analysis, we assume that the variable is bounded, as assumed in Kingma and Ba [2014].

**Assumption 5.** *We assume that the distance between the variable and the optimal point is bounded during the optimization process, such that* $\|x_{t,\tau} - x^*\|_2 \leq D$, $\|x_{i,j} - x_{k,l}\|_\infty \leq D_\infty$.

*Proof of Theorem 1.* Since the objective function $f$ is convex,

$$f(x_{t,K}) - f(x^*) \leq \langle \nabla f(x_{t,K}), x_{t,K} - \theta^* \rangle = \sum_{i=1}^{d} g_{t,K,i}(x_{t,K,i} - x_i^*).$$

Using the update method defined in Algorithm 1, we can get

$$\begin{aligned} x_{t+1,1,i} &= x_{t,K,i} - \gamma \frac{\hat{m}_{t,K,i}}{\sqrt{\hat{v}_{t,K,i}}} \\ &= x_{t,K,i} - \frac{\gamma}{1-\beta_1^t} \left( \frac{K-1}{\sqrt{K}\sqrt{v_{t,K,i}}} m_{t,K-1,i} + \frac{1-\beta_1}{\sqrt{K}\sqrt{v_{t,K,i}}} g_{t,K,i} \right). \end{aligned}$$

$$\begin{aligned} (x_{t+1,1,i} - x^*)^2 = (x_{t,K,i} - x_i^*)^2 &- \frac{2\gamma}{1-\beta_1^t} (x_{t,K,i} - x_i^*) \left( \frac{K-1}{\sqrt{K}\sqrt{v_{t,K,i}}} m_{t,K-1,i} + \frac{1-\beta_1}{\sqrt{K}\sqrt{v_{t,K,i}}} g_{t,K,i} \right) \\ &+ \gamma^2 K \left( \frac{m_{t,K,i}}{\sqrt{v_{t,K,i}}} \right)^2. \end{aligned}$$

Rearrange the equation above,

$$\begin{aligned} g_{t,K,i}(x_{t,K,i} - x_i^*) &= \frac{(1-\beta_1^t)\sqrt{K}\sqrt{v_{t,K,i}}}{2\gamma(1-\beta_1)} \left( (x_{t+1,K,i} - x_i^*)^2 - (x_{t,K,i} - x_i^*)^2 \right) \\ &+ \frac{K-1}{1-\beta_1} m_{t,K-1,i}(x_{t,K,i} - x_i^*) + \frac{(1-\beta_1^t)\gamma K^{\frac{3}{2}}\sqrt{v_{t,K,i}}}{2(1-\beta_1)} \left( \frac{m_{t,K,i}}{\sqrt{v_{t,K,i}}} \right)^2 \\ &\leq \frac{\sqrt{K}\sqrt{v_{t,K,i}}}{2\gamma(1-\beta_1)} \left( (x_{t+1,K,i} - x_i^*)^2 - (x_{t,K,i} - x_i^*)^2 \right) \\ &+ \frac{K-1}{1-\beta_1} (x_{t,K,i} - x_i^*)\sqrt{v_{t,K-1}} \left( \frac{m_{t,K-1,i}}{\sqrt{v_{t,K-1,i}}} \right) + \frac{\gamma K^{\frac{3}{2}}}{2(1-\beta_1)} \frac{m_{t,K,i}^2}{\sqrt{v_{t,K,i}}} \\ &\leq \frac{\sqrt{K}\sqrt{v_{t,K,i}}}{2\gamma(1-\beta_1)} \left( (x_{t+1,K,i} - x_i^*)^2 - (x_{t,K,i} - x_i^*)^2 \right) \\ &+ \frac{K-1}{2(1-\beta_1)} (x_{t,K,i} - x^*)^2 \cdot \sqrt{v_{t,K,i}} + \frac{K-1}{2(1-\beta_1)} \frac{m_{t,K-1,i}^2}{\sqrt{v_{t,K-1,i}}} + \frac{\gamma K^{\frac{3}{2}}}{2(1-\beta_1)} \frac{m_{t,K,i}^2}{\sqrt{v_{t,K,i}}}. \end{aligned}$$

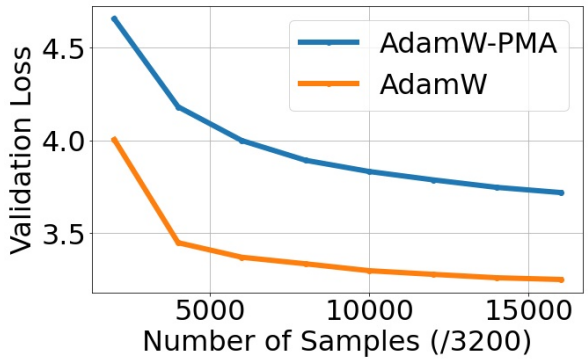



Figure 9: Runtime to achieve the same loss on DPO task. PMA can reduce the training time cost than EMA.

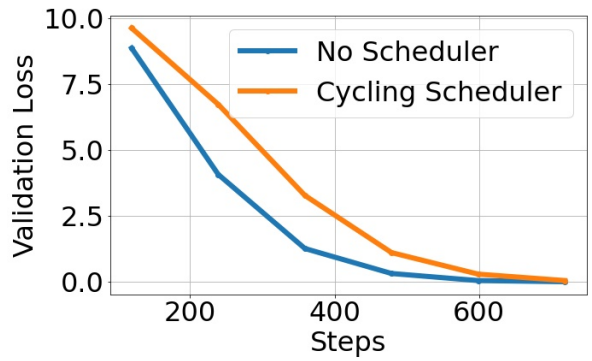



Figure 10: Validation loss of AdamW without learning rate scheduler and AdamW with a PMA-like lr scheduler.

$$
\begin{aligned}
R_K(T) &\le \sum_{t=1}^{T}\sum_{i=1}^{d} g_{t,K,i}(x_{t,K,i}-x_i^*) \\
&\le \sum_{i=1}^{d}\sum_{t=1}^{T} \frac{\sqrt{K}\sqrt{v_{t,K,i}}}{2\gamma(1-\beta_1)}(x_{t+1,K,i}-x_i^*)^2 - \frac{\sqrt{K}\sqrt{v_{t,K,i}}}{2\gamma(1-\beta_1)}(x_{t,K,i}-x_i^*)^2 \\
&+ \frac{K-1}{2(1-\beta_1)}(x_{t,K,i}-x^*)^2\cdot\sqrt{v_{t,K,i}} + \frac{K-1}{2(1-\beta_1)}\frac{m_{t,K-1,i}^2}{\sqrt{v_{t,K-1,i}}} + \frac{\gamma K^{\frac{3}{2}}}{2(1-\beta_1)}\frac{m_{t,K,i}^2}{\sqrt{v_{t,K,i}}} \\
&\overset{\text{Lemma 2}}{\le} \frac{\sqrt{K}D^2}{2\gamma(1-\beta_1)}\sum_{i=1}^{d}\sqrt{T\hat{v}_{T,K,i}} + \frac{D_\infty^2(K-1)}{2(1-\beta_1)}\cdot\sum_{i=1}^{d}\sum_{t=1}^{T}\sqrt{v_{T,K,i}} \\
&+ \frac{(1+\gamma)K^{\frac{3}{2}}G_\infty}{2(1-\beta_1)}\sum_{i=1}^{d}\|g_{1:KT,i}\|_2 \\
&\le \frac{\sqrt{K}D^2}{2\gamma(1-\beta_1)}\sum_{i=1}^{d}\sqrt{T\hat{v}_{T,K,i}} + \frac{(1+\gamma)K^{\frac{3}{2}}G_\infty}{2(1-\beta_1)}\sum_{i=1}^{d}\|g_{1:KT,i}\|_2 + \frac{D_\infty^2 G_\infty(K-1)}{2(1-\beta_1)}.
\end{aligned}
$$

□

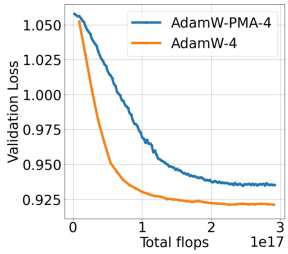
# E ADDITIONAL EXPERIMENTS

## E.1 PRE-TRAINING

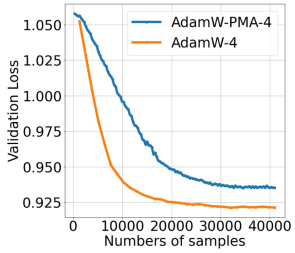
Although `PMA` is designed for post-training, we also evaluate its performance on pre-training task. Specifically, we train a randomly-initialized nanoGPT model on WikiPedia dataset. Figure 9 shows the validation loss of `AdamW-PMA` and AdamW. EMA-based AdamW achieves a lower validation loss than `AdamW-PMA`. This is because `PMA`, especially the small update step, is designed for post-training tasks where the distance between the original and trained parameters is small. Large distance of updates, such as pre-training, can make the update direction deviate too much from the direction of AdamW, leading to a slow training.

## E.2 ABLATION ON LEARNING RATE SCHEDULER

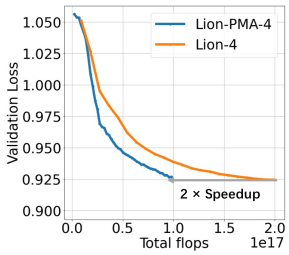
To evaluate how the learning rate scheduler introduced in Sec. 3.2.2, we conduct an experiment on Qwen2-0.5B, comparing AdamW without a scheduler and with a PMA-like scheduler. The other settings are the same as the experiment in Fig. 4. We evaluate the tuned model every 120 steps, and the statistics are shown in Fig. 10. The PMA-like scheduler slows down the



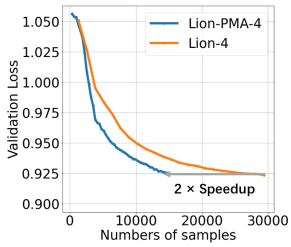

(a) AdamW-PMA v.s. AdamW on flops




(b) AdamW-PMA v.s. AdamW on number of samples

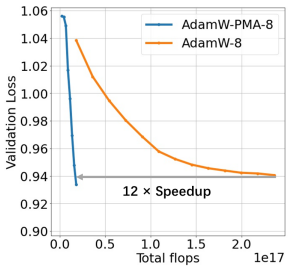



(c) Lion-PMA v.s. Lion on flops

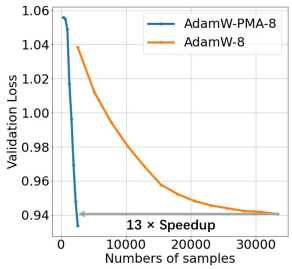



(d) Lion-PMA v.s. Lion on number of samples

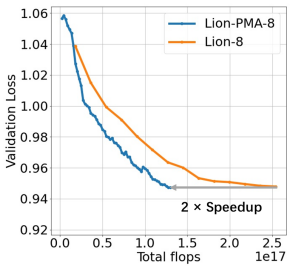
Figure 11: From the perspectives of total flops and number of steps, AdamW-PMA and Lion-PMA achieved speedups of 1.8x and 1.4x respectively, compared to AdamW and Lion when K = 1.



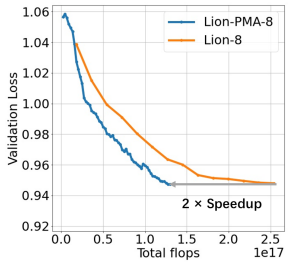

(a) AdamW-PMA v.s. AdamW on Flops




(b) AdamW-PMA v.s. AdamW on Number of Samples




(c) Lion-PMA v.s. Lion on Flops




(d) Lion-PMA v.s. Lion on Number of Samples

Figure 12: We evaluate the optimizers by comparing the total flops and number of samples needed to achieve the same validation loss level. `AdamW-PMA` and `Lion-PMA` achieved approximately 12x and 2x speedup, respectively, relative to AdamW and Lion.

training process if the other components of PMA are not applied. This result indicates the necessity of the joint design of each component in `AdamW-PMA`.

### E.3 SFT

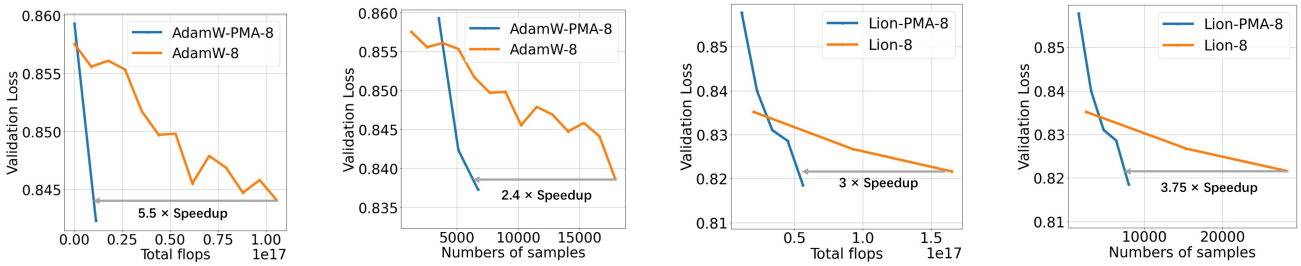
The improvement in validation loss brought by PMA can be translated into a reduction of the number of steps or total compute. In Figure 11, we evaluate the optimizers by comparing the number of steps or total flops needed to achieve the same validation loss level, setting K to 4. As can be observed in Figure 12, AdamW-PMA and Lion-PMA achieve a 12x and 2x speedup compared with AdamW and Lion.

### E.4 DPO

Figure 13 and Figure 14 illustrate the validation loss of the DPO task on Phi-2 and HH-RLHF-harmless dataset, using four different optimizers. We compare the total flops and number of samples needed to achieve the same validation loss across vanilla AdamW and AdamW-PMA, Lion and Lion-PMA. The corresponding accuracy graph for this experiment can be found in Figure 3 of Section 5.3 in the main text.

### E.5 HYPER-PARAMETER SENSITIVITY

We do experiments on DPO task with the Phi-2-2.7B model and Qwen1half-0.5B-chat model to explore the sensitivity of the PMA method's speedup factor with hyper-parameter K on AdamW. In experiment of Phi-2 model, we set K to be 8, 16, 32, 64 to explore the optimal K value. For Qwen1.5-0.5B model, the K is set to be 4, 8, 16, 32, which are relatively smaller



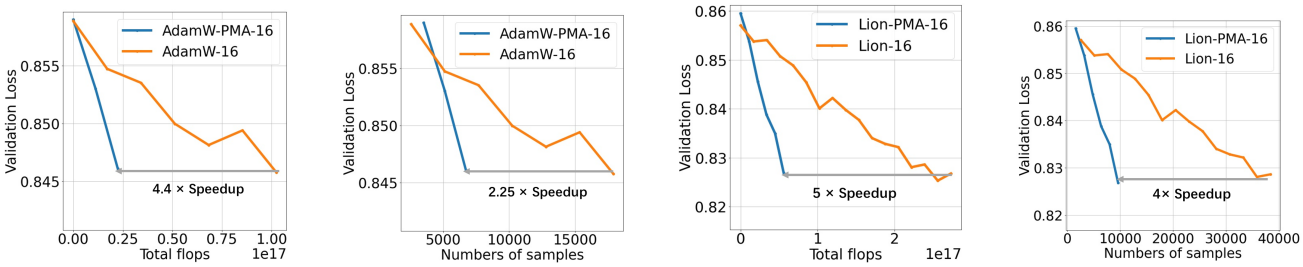

(a) AdamW-PMA v.s. AdamW on flops

(b) AdamW-PMA v.s. AdamW on number of samples

(c) Lion-PMA v.s. Lion on flops

(d) Lion-PMA v.s. Lion on number of samples

Figure 13: Validation loss of the DPO task on Phi-2 and HH-RLHF-harmless dataset.




(a) AdamW-PMA v.s. AdamW on flops

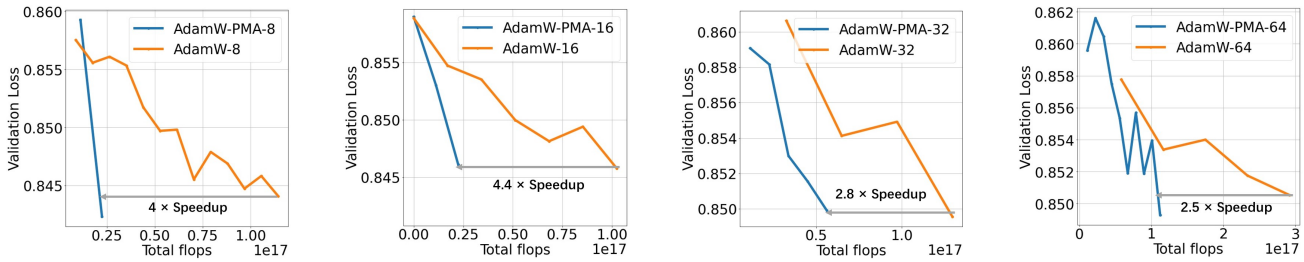
(b) AdamW-PMA v.s. AdamW on number of samples

(c) Lion-PMA v.s. Lion on flops

(d) Lion-PMA v.s. Lion on number of samples

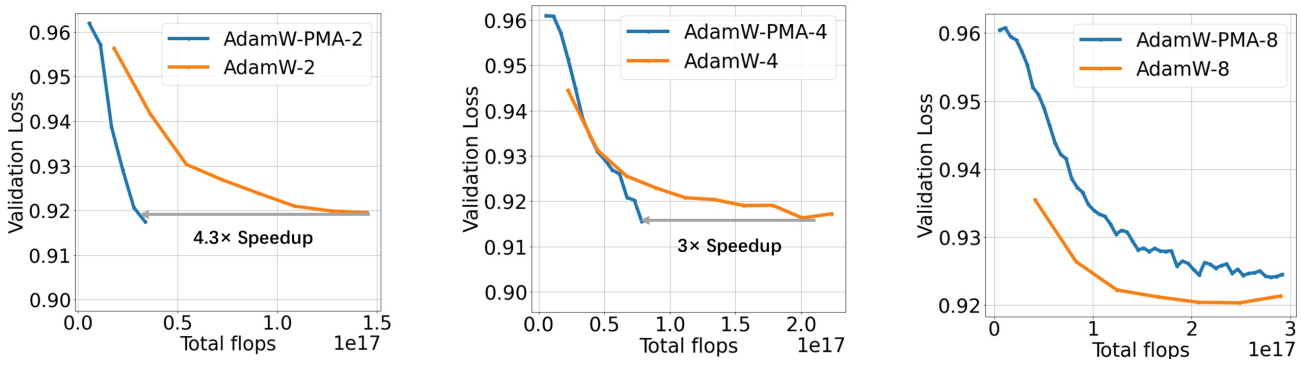
Figure 14: We evaluate the optimizers by comparing the total flops and number of samples needed to achieve the same DPO validation loss level, with K setting to be 16. `AdamW-PMA` and `Lion-PMA` achieved approximately 4x and 5x speedup, respectively, relative to AdamW and Lion.

since the model is smaller. The results of experiments can be seen in Figure 15 and 16. This part is the supplement results of Section 5.4 in the main text.




(a) (b) (c) (d)

Figure 15: The sensitivity of PMA's speedup factor with hyper-parameter K on Phi-2 model using AdamW




(a) (b) (c)

Figure 16: The sensitivity of PMA's speedup factor with hyper-parameter K on Qwenhalf1-0.5B model using AdamW