ADAMPM: MEMORY-EFFICIENT LLM OPTIMIZATION VIA MOMENT POOLING

Anonymous authors
Paper under double-blind review

ABSTRACT

Adam is widely used for training large language models (LLMs) owing to its strong optimization performance and training stability, but its element-wise estimation for first and second moments incur substantial memory overhead. Although numerous memory-efficient optimizers have been proposed, many incur additional computational overhead or require extra tuning in hyperparameters. Consequently, Adam remains a dominant and reliable choice for LLM pretraining. We propose AdamPM (Adam with Pooled Moments), a simple and memory-efficient variant of Adam that groups adjacent moment entries along the row or column dimension and represents each group by a single shared mean. AdamPM can well inherit Adam's hyperparameter settings and is readily integrated into existing Adam optimizer implementations, with less memory overhead. We systematically investigate pooling of both first and second moment estimates and identify second moment pooling as the effective design. With a group size of s, AdamPM reduces second moment memory to $1/s$ of the original requirement. Extensive experiments on language model pretraining, fine-tuning and multimodal learning demonstrate that AdamPM maintains model performance and training speed while substantially reducing optimizer-state memory.

1 INTRODUCTION

Adam (Kingma & Ba, 2015) and its decoupled weight-decay variant AdamW (Loshchilov & Hutter, 2019), which combine momentum with adaptive learning rates, have become the de facto standard for LLM training due to their strong convergence and stability. However, these advantages come at a substantial memory cost: Adam maintains first and second moment estimates for each parameter, requiring additional memory usage up to twice of the model size.

As LLMs scale to trillions of parameters, this overhead becomes increasingly prohibitive. Against this backdrop, memory-efficient optimizers have emerged as an important research direction, aiming to reduce training memory consumption through optimizer design while preserving the performance of conventional adaptive methods. Representative approaches achieve this goal by projecting gradients into low-rank subspaces (Zhu et al., 2025; Robert et al., 2025), factorizing the second moment state (Luo et al., 2023), sharing the moment statistics (Zhang et al., 2025; Wen et al., 2026b), and employing orthogonalization (Liu et al., 2025) or sign-based update rules (Chen et al., 2023) to eliminate the second moment state.

However, a recent survey from Stanford University (Wen et al., 2026a) notes that, despite numerous optimizers claiming faster training or lower memory consumption, Adam remains the robust default for LLM pre-training, with its primary limitation being the memory overhead of optimizer states. This raises a natural question:

Before an optimizer consistently surpasses Adam in both performance and robustness, would a memory-efficient variant with only minimal modifications to Adam be a better practical choice?

In this work, we systematically analyze the correlations and redundancy in Adam's optimizer states, and propose AdamPM, a minimally modified Adam variant that groups adjacent second moment entries and shares one statistic within each group. Our main contributions are summarized as follows:

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

- We propose AdamPM, a memory-efficient optimizer for LLM training that reduces optimizer-state memory without sacrificing model performance or increasing training time. AdamPM makes only a minimal modification to vanilla Adam by pooling the second moment state. It can directly reuse Adam's hyperparameters, including the learning rate and decay rates, and is fully model-agnostic. These characteristics of AdamPM are validated by experiments on LLM pre-training, language-model and vision-language model fine-tuning.
- We systematically investigate moment pooling of both the first and second moment by experiments. We find that the first moment contains substantially more information and is highly sensitive to pooling, explaining why most memory-efficient optimizers primarily compress the second moment. We introduce cosine similarity and relative error to quantify deviations from vanilla Adam, and show that AdamPM's preconditioners and parameter updates become more similar as model scale grows, suggesting that AdamPM will have more benefits in memory-efficiency over Adam for larger models.

2 Related Work

As LLMs scale to trillions of parameters, memory-efficient training has become a research priority. Existing memory-efficient optimizers mainly address to reduce memory overhead by matrix factorization, low-rank projection, constraint-based transformations, and parameter sharing.

Matrix factorization. These methods address reduction of memory by storing the factorized matrices, and the mainly differences involve to the optimization targets. For example, Adafactor (Shazeer & Stern, 2018) and SMMF (Park & Lee, 2025) factorize optimizer states, the LoRA (Hu et al., 2022) family factorizes weight updates, and LORO (Mo et al., 2025) factorizes weight matrices. Although these methods improve memory efficiency, they are not suitable for pre-training and only confined to fine-tuning (e.g. LoRA), or have not yet been validated as extensively as Adam on LLMs with billions of parameters for pre-training (e.g. Adafactor and SMMF). Our AdamPM, like Adam, works well for both pre-training and fine-tuning, and is validated by experiments on large-scale LLMs and vision-language models.

Low-rank projection. These methods aim at reducing memory by maintaining optimizer states in lower-dimensional spaces. They achieve this goal by random projections in APOLLO (Zhu et al., 2025), or by gradient projection based on either SVD in GaLore (Zhao et al., 2024) and Fira (Chen et al., 2025), or online PCA (Liang et al., 2024). Although these methods reduce memory consumption, they also incur additional computational overhead, such as periodic SVD computations in GaLore (Zhao et al., 2024), which increase training time. Besides, they often introduce new hyperparameters that are difficult to tune, such as the frequency of periodic subspace updates in Fira (Chen et al., 2025). Our proposed AdamPM makes only minimal changes to Adam and preserves its desirable optimization properties, while its parameter updates become increasingly similar to Adam's as model size grows, as demonstrated in Figure 4 (d).

Constraint-based transformations. These method aim to eliminate the need to store certain optimizer states by introducing constraint-based transformations, e.g., imposing orthogonality constraints on optimizer states in Muon (Liu et al., 2025) and SUMO (Refael et al., 2025), normalizing gradients along the output feature dimension in SCALE (Glentis et al., 2025), and using sign-based momentum updates in Lion (Chen et al., 2023) and OLion (Wang et al., 2026). These optimizers substantially modify Adam, preventing direct reuse of its hyperparameters and requiring additional tuning. AdamPM makes only minimal changes to Adam by using parameter sharing, and can directly reuse Adam's hyperparameters, including the learning rate and decay rates.

Parameter sharing. Adam-mini (Zhang et al., 2025) and FOAM (Wen et al., 2026b) also reduce optimizer-state memory by sharing moment estimates. However, compared with Adam-mini (Zhang et al., 2025), AdamPM offers a more general grouping scheme, pooling adjacent second moment entries along rows or columns with a configurable group size s. Moreover, AdamPM is model-agnostic, whereas Adam-mini relies on model-specific information such as the hidden dimension and attention-head count. AdamPM also differs from FOAM (Wen et al., 2026b), which restricts sharing to power-of-two-sized groups of adjacent elements within each row, applies the same compression to first moments, and introduces residual correction. These additional operations make it more complex to invoke the optimizer, and FOAM also cannot well reuse the hyperparameters of Adam. Our AdamPM is fully model-agnostic, and can directly reuse Adam's hyperparameters.

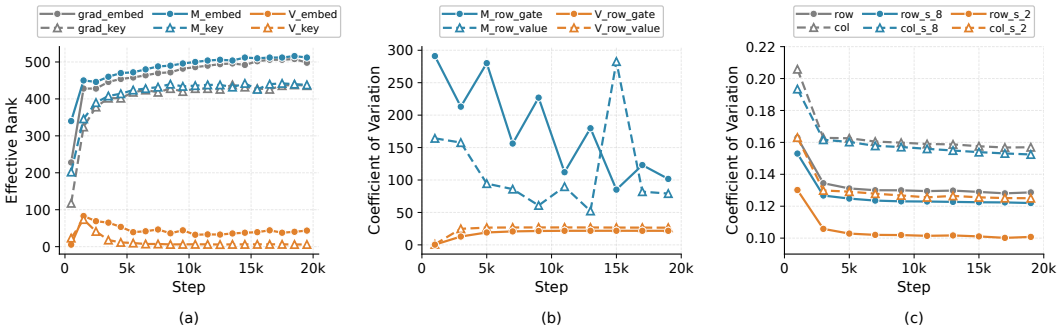


Figure 1: Evolution of effective ranks and coefficients of variation during LLaMA 130M pretraining. (a) Effective ranks of the gradients, first and second moments for `embed` and `key`. (b) Coefficients of variation of the first and second moments for `value` and `gate`. (c) Coefficients of variation of the preconditioner for `gate`.

3 Methodology

We first revisit Adam optimizer in Section 3.1 and examine the local similarity of second moment statistics in Section 3.2. Motivated by these observations, we introduce AdamPM in Section 3.3 and analyze its convergence in Section 3.4.

3.1 Vanilla Adam

Given the stochastic gradients G_t of a mini-batch, Adam maintains the estimates of the first momentum M_t and the second momentum V_t by exponential moving averages (EMA). In particular, it updates the weight W_t as follows:

$$\begin{aligned} M_t &= \beta_1 M_{t-1} + (1-\beta_1) G_t, \\ V_t &= \beta_2 V_{t-1} + (1-\beta_2) G_t \odot G_t, \\ \hat{M}_t &= \tfrac{M_t}{1-\beta_1^t}, \quad \hat{V}_t = \tfrac{V_t}{1-\beta_2^t}, \\ W_t &= W_{t-1} - \eta_t \cdot \frac{\widehat{M}_t}{\sqrt{\widehat{V}_t + \epsilon}}. \end{aligned} \tag{1}$$

where $\hat{M}_t$ and $\hat{V}_t$ are bias-corrections of the first and second moment respectively, $\beta_1, \beta_2 \in [0, 1)$ are decay rates, $\eta_t > 0$ is the learning rate, and $\epsilon > 0$ ensures numerical stability. All operations are performed elementwisely.

3.2 Local Similarity of Second Moment Statistics

Modern Transformer-based large language models widely adopt LayerNorm (Ba et al., 2016) and RMSNorm (Zhang & Sennrich, 2019). By reducing scale discrepancies across different weights and rescaling the loss landscape, normalization partially plays a role similar to adaptive learning-rate adjustment, which is achieved in Adam through second moment estimation. Consequently, the effective learning rates of different parameters tend to become more similar.

We examine the structural redundancy in Adam's optimizer states during the pretraining of LLaMA 130M model (Touvron et al., 2023). For illustration, Figure 1 (a) reports the effective ranks of the gradient, first and second moments for `embed` and the fifth Transformer block's `key`. We define the effective rank of A as $\mathrm{erank}(A) = \exp(-\sum_i p_i \log p_i)$, where $p_i = \sigma_i / \sum_j \sigma_j$ and σ_i denotes its i-th singular value. Throughout training, the second moment consistently exhibits a substantially lower effective rank, indicating that its information is concentrated in only a few dominant directions. In contrast, the gradient and first moment contain richer directional information, with the first moment exhibiting an even higher effective rank than the gradient. This observation is consistent with their roles in Adam: the first moment primarily determines the update direction, whereas the second moment controls the coordinate-wise update magnitude through the diagonal preconditioner. The second moment can tolerate moderate approximation as long as the resulting update scale remains sufficiently accurate. Adam-mini (Zhang et al., 2025) also supports this view

by sharing a single second moment statistic within each parameter block while retaining competitive performance.

To assess local similarity during LLaMA 130M pretraining, we compute the coefficient of variation (CV) for the first moment, second moment, and diagonal preconditioner. For entries $\mathcal{X}$, CV is defined as $\mathrm{CV}(\mathcal{X}) = \sigma_{\mathcal{X}}/|\mu_{\mathcal{X}}|$, where $\mu_{\mathcal{X}}$ and $\sigma_{\mathcal{X}}$ are their mean and standard deviation, respectively.

Figure 1 (b) compares the CVs of the first and second moments for `value` and the MLP gate projection matrix `gate` in the fifth Transformer block, while Figure 1 (c) reports the preconditioner CVs for `gate`. The labels `row` and `col` denote mean row-wise and column-wise CVs, respectively, while `s_`n denotes the mean CV across non-overlapping groups of n adjacent elements along the corresponding dimension. We observe that the first moment exhibits substantially higher CVs because it contains both positive and negative values and carries richer directional information, whereas the second moment has much lower CVs. Moreover, parameter updates depend directly on the diagonal preconditioner $\frac{1}{\sqrt{\widehat{V}_t}+\epsilon}$, whose inverse-square-root transformation further reduces this variation. The preconditioner exhibits consistently low CVs throughout training, indicating strong local similarity among neighboring elements, with even lower variation in smaller group size.

3.3 AdamPM Optimizer

These observations motivate AdamPM, which reduces optimizer-state memory by applying **group-wise moment pooling** to Adam's second moment. AdamPM pools every s adjacent second moment entries within rows or columns, where s denotes the group size. By default, AdamPM performs pooling within each row, with rows corresponding to output features. In particular, given a weight matrix $W \in \mathbb{R}^{m\times n}$, each row is partitioned into $q = n/s$ groups if n is divisible by s. When n is not divisible by s, the final group of squared gradients is padded with its mean.

Algorithm 1 AdamPM Optimizer

Input: Initialized parameters $W_0 \in \mathbb{R}^{m\times n}$, total steps T, group size s, learning rate η_t, decay rates $\{\beta_1, \beta_2\}$, and numerical stabilization constant ϵ.

Initialize $t \leftarrow 0$, $q \leftarrow n/s$ {q is number of groups, assuming $s \mid n$}
Initialize $M_0 \leftarrow \mathbf{0}_{m\times n}$, $\widetilde{V}_0 \leftarrow \mathbf{0}_{m\times q}$ {First moment and compressed second moment}
repeat
  $t \leftarrow t+1$
  $G_t \leftarrow \nabla_W \mathcal{L}(W_{t-1})$ {Compute the gradient G_t}
  $M_t \leftarrow \beta_1 M_{t-1} + (1-\beta_1) G_t$ {Update the first moment M_t}
  $S_t = G_t \odot G_t \in \mathbb{R}^{m\times n}$ {Compute the squared gradient}
  $R_t \leftarrow S_t.\mathrm{reshape}(m, q, s) \in \mathbb{R}^{m\times q\times s}$ {Reshape the squared gradient}
  $\overline{S}_t \leftarrow \frac{1}{s}\sum_{k=1}^{s} (R_t)_{:,:,k} \in \mathbb{R}^{m\times q}$ {Mean within each group}
  $\widetilde{V}_t \leftarrow \beta_2 \widetilde{V}_{t-1} + (1-\beta_2)\overline{S}_t$ {Update the compressed second moment $\widetilde{V}_t$}
  $\widehat{M}_t \leftarrow \frac{M_t}{1-\beta_1^t}$ {Bias-Correction of the first moment}
  $\widehat{\widetilde{V}}_t \leftarrow \frac{\widetilde{V}_t}{1-\beta_2^t}$ {Bias-correct the compressed second moment}
  $\widehat{V}_t \leftarrow \widehat{\widetilde{V}}_t \otimes \mathbf{1}_{1\times s} \in \mathbb{R}^{m\times n}$ {Broadcast each group, $\otimes$ denotes Kronecker product}
  $W_t \leftarrow W_{t-1} - \eta_t \cdot \frac{\widehat{M}_t}{\sqrt{\widehat{V}_t}+\epsilon}$ {Update weights}
until $t = T$ or convergence
return W_t

The complete procedure is presented in Algorithm 1. AdamPM performs moment pooling as follows. It first computes the element-wise squared gradient and reshapes it into $m \times q$ groups of s adjacent elements. Then AdamPM computes the group means $\overline{S}_t$ and applies an EMA to obtain the compressed second moment $\widetilde{V}_t$. After bias correction, each group statistic is temporarily broadcast to its corresponding s elements before the parameter update.

AdamPM pools second moments for all weight matrices, including `embed`, `lm_head`, attention, and MLP weights. One-dimensional parameters, such as RMSNorm weights, are pooled as single-row or single-column matrices. Note that, unlike FOAM (Wen et al., 2026b), AdamPM doesnot pool the first moment and follows Adam's element-wise update, since pooling the first moment naively will lead to significantly degenerated performance (Section 4.2).

The order of squared-gradient computation, pooling, and EMA accumulation is crucial for memory efficiency, as it is the EMA that incurs persistent optimizer-state storage overhead. By averaging squared gradients within each group before applying the EMA, AdamPM maintains only the compressed second-moment state $\widetilde{V}_t \in \mathbb{R}^{m\times q}$ throughout training, reducing second-moment memory usage to $1/s$ of that required by Adam. It therefore stores mn first moment and mn/s second moment elements, totaling $mn(1+1/s)$ optimizer-state elements, compared with $2mn$ for Adam.

3.4 Convergence Analysis

This section presents a finite-time stationarity analysis of AdamPM. We assume that the objective is lower bounded and L-smooth. We further assume that stochastic gradients are conditionally unbiased with bounded noise variance and that the true gradient at each iterate is almost surely bounded in norm. Complete assumptions and proofs are provided in Appendix A. Here, C and σ^2 are upper bounds on the norm of the true gradient and the noise variance, respectively, while s denotes the row-wise group size.

Theorem 3.1 (Stationarity bound)**.** *Under the above conditions and with $\eta_t = \frac{\eta_0}{\sqrt{t}}$, we have*

$$\min_{1\le t\le T} \mathbb{E}\|\nabla f(W_t)\|_2^2 \le K\frac{1+\log T}{\sqrt{T}} + \left(3+\frac{4C}{\epsilon}\right)\sigma^2, \tag{2}$$

where K is finite and independent of T and s.

This bound, including its noise residual, is independent of group size s. Consequently, AdamPM and Adam ($s=1$) share the same stationarity upper bound in our analysis.

Corollary 3.2 (Vanishing stationarity bound)**.** *Under Theorem 3.1, suppose additionally that*

$$\mathbb{E}[\|\xi_t\|_2^2 \mid W_t] \le \kappa\|\nabla f(W_t)\|_2^2, \quad 0\le\kappa<\frac{\epsilon}{16C+12\epsilon}, \tag{3}$$

where $\xi_t = G_t - \nabla f(W_t)$. Then

$$\min_{1\le t\le T} \mathbb{E}\|\nabla f(W_t)\|_2^2 \le \frac{K(1+\log T)}{[1-4\kappa(3+4C/\epsilon)]\sqrt{T}}. \tag{4}$$

Thus, under the additional relative-noise condition, AdamPM achieves the same $O(\log T/\sqrt{T})$ convergence rate as Adam. Proofs and explicit constants are provided in Appendix A.

4 Experiments

We conduct comprehensive experiments to evaluate AdamPM across diverse training scenarios. Our evaluation includes LLaMA (Touvron et al., 2023) pre-training, language-model fine-tuning on the GLUE benchmark (Wang et al., 2019), and multimodal fine-tuning using TinyLLaVA (Zhou et al., 2024), a compact vision-language model. These experiments span different model architectures, parameter scales, and learning tasks, providing a systematic assessment of AdamPM in both pre-training and downstream adaptation. Throughout these experiments, AdamPM-s denotes AdamPM with a group size of s.

4.1 Pre-Training of LLaMA

Experimental setup. We pre-train LLaMA models with 60M, 130M, 350M, and 1.3B parameters on the English subset of C4 (Raffel et al., 2020). Results are summarized in Table 1. We evaluate performance using validation perplexity (PPL), where lower is better. We also report the combined memory footprint of model parameters and optimizer states, both stored in BF16.

Table 1: Comparison of AdamPM, Adam, and representative memory-efficient optimizers for pre-training LLaMA models of various sizes on the C4 dataset. Validation perplexity is reported, with the estimated BF16 memory cost of model parameters and optimizer states. * denotes the results are taken directly from the original papers, as they use the same pre-training setup, model scales, and token budgets.

Model Size	**60M**	**130M**	**350M**	**1.3B**
Training Tokens	1.1B	2.2B	6.4B	13.1B
Adam	30.30 (0.325G)	23.23 (0.749G)	17.79 (2.05G)	14.83 (7.48G)
AdamPM-2	30.31 (0.270G)	23.47 (0.624G)	17.69 (1.71G)	14.70 (6.24G)
AdamPM-4	30.25 (0.243G)	22.99 (0.562G)	17.64 (1.54G)	14.66 (5.61G)
AdamPM-8	30.61 (0.230G)	22.87 (0.531G)	17.75 (1.46G)	14.73 (5.30G)
AdamPM-16	30.57 (0.223G)	23.17 (0.515G)	17.86 (1.41G)	14.93 (5.14G)
AdamPM-inf	30.55 (0.217G)	23.36 (0.500G)	18.27 (1.37G)	15.54 (4.99G)
Adam-mini	30.82 (0.217G)	23.57 (0.500G)	18.59 (1.37G)	15.72 (4.99G)
GaLore*	34.88 (0.24G)	25.36 (0.52G)	18.95 (1.22G)	15.64 (4.38G)
APOLLO*	31.55 (0.24G)	22.94 (0.52G)	16.85 (1.22G)	14.20 (4.38G)
SUMO*	34.26 (0.23G)	24.87 (0.51G)	18.69 (1.16G)	14.68 (3.84G)
LORO*	33.96 (0.24G)	24.59 (0.57G)	18.84 (1.11G)	15.19 (3.66G)

For all model scales, we set $\beta_1 = 0.9$, $\beta_2 = 0.95$, and $\epsilon = 10^{-8}$. For the 60M, 130M, 350M, and 1.3B models, both AdamPM and Adam use learning rates of 4×10^{-3}, 2×10^{-3}, 6×10^{-4}, and 3×10^{-4}, respectively. These values are selected for their robustness based on learning rate sweeps for Adam, such as those shown in Figure 2. Detailed training configurations are provided in Appendix C.1

Baselines. We compare AdamPM with vanilla Adam, Adam-mini (Zhang et al., 2025), and several other representative memory-efficient optimizers, including GaLore (Zhao et al., 2024), which compresses optimizer states by projecting gradients into low-rank subspaces through SVD; APOLLO (Zhu et al., 2025), which uses random projection to construct subspaces; SUMO (Refael et al., 2025), which orthogonalizes the projected first moment; and LORO (Mo et al., 2025), which pre-trains low-rank-parameterized weight matrices through Riemannian optimization. We evaluate Adam-mini in our own experiments, using the same learning rates, moment decay rates, and other hyperparameters as Adam and AdamPM.

We carefully tune the learning rate of Adam for each model scale. As shown in Table 1, the tuned Adam baseline achieves substantially stronger performance than the Adam baselines reported in several previous studies, such as SUMO (Refael et al., 2025) and LORO (Mo et al., 2025). This observation highlights the importance of adequately tuning Adam when evaluating memory-efficient optimizers.

Main results. As shown in Table 1, AdamPM achieves perplexity comparable to that of carefully tuned vanilla Adam and frequently yields lower perplexity when the group size is not too large, while substantially reducing optimizer-state memory. In particular, AdamPM-8 and AdamPM-16 reduce second moment memory by 87.5% and 93.75%, respectively, relative to Adam. We further report the actual memory usage per GPU during LLaMA-1.3B pre-training in the Appendix Table A2. These results demonstrate that Adam's second moment statistics can be pooled to reduce memory usage without noticeably degrading pre-training performance. We additionally evaluate an extreme pooling strategy, denoted as AdamPM-inf, in which all second moment entries within each row share a single scalar. As shown in Table 1, this aggressive pooling leads to modest performance degradation on the 350M and 1.3B models, while Adam-mini, which uses more aggressive second moment sharing, performs even worse on these two models. Overall, group sizes between 2 and 8 generally yield robust performance.

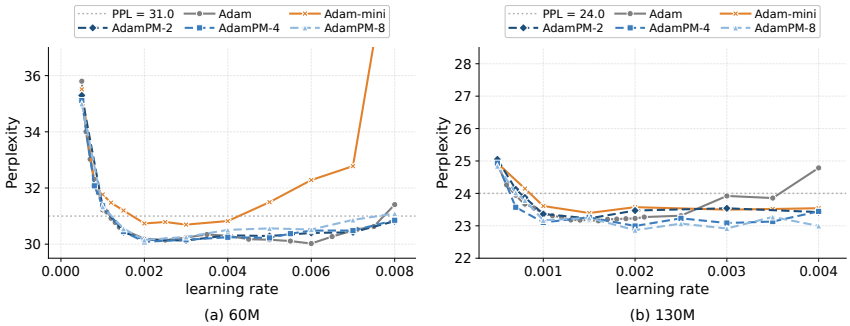


Figure 2: Learning rate sweeps for LLaMA 60M and 130M pre-training on C4

4.2 Analysis of Pre-Training Experiments

Hyperparameter transferability. Figure 2 presents learning rate sweeps for LLaMA 60M and 130M pre-training, with $\beta_1 = 0.9$ and $\beta_2 = 0.95$ fixed. We evaluate AdamPM with group sizes of 2, 4, and 8, and observe that it maintains competitive perplexity over the learning rate ranges where Adam performs well. These results demonstrate that AdamPM can inherit the well-tuned hyperparameters of Adam, making it easy to integrate into existing Adam optimizer implementations. Furthermore, the LLaMA 60M results in Figure 2 (a) show that AdamPM inherits Adam's learning rates more effectively than Adam-mini across all tested group sizes.

Flexibility of the pooling strategy. Table 2 provides additional analyses of the group size and pooling direction. AdamPM-3 and AdamPM-5 achieve performance comparable to, or even better than Adam. This result highlights AdamPM's flexibility: padding the final incomplete group with the mean of its existing entries allows any positive integer group size, rather than only powers or multiples of two. We also evaluate column-wise second moment pooling, denoted as AdamPM-col. This variant achieves performance comparable to that of Adam, demonstrating that pooling is effective in either direction.

Table 2: Ablation study of pooling strategy.

Model Size	**60M**	**130M**	**350M**
Adam	30.30	23.23	17.79
AdamPM-3	30.58	23.36	17.63
AdamPM-5	30.54	23.09	17.59
AdamPM-col-2	30.45	23.40	17.51
AdamPM-col-3	30.36	23.34	17.53
AdamPM-M-2	59.99	206.35	49.26
AdamPM-M†-2	42.35	30.12	23.71

First- versus second-order moment pooling. We further evaluate the first moment pooling variant, denoted as AdamPM-M. Notably, directly pooling first moment entries leads to substantial performance degradation. As shown in Table 2, AdamPM-M-2, which pools every two adjacent entries within each row of the first moment matrices, performs considerably worse than vanilla Adam.

Following the layer-wise strategy adopted by GaLore (Zhao et al., 2024) and FRUGAL (Zmushko et al., 2025), we revert the `embed`, `lm_head` and `normalization` layers to vanilla Adam while applying first moment pooling to the remaining parameters; we denote this variant as AdamPM-M†. The results in Table 2 show that although this modification improves performance over pooling the first moments of all weight parameters, a substantial performance gap relative to vanilla Adam remains.

Together with the higher effective ranks and larger coefficients of variation observed in first moment matrices, these results suggest that the first moment carries substantial information about parameter-update directions. Consequently, the first and second moments should not be considered equally compressible: the second moment exhibits considerable redundancy and can be effectively pooled, whereas directly pooling the first moment discards important optimization information and substantially degrades performance.

Training dynamics. Figure 3 presents the validation-perplexity curves of AdamPM-4, AdamPM-8, and Adam during the pre-training of the 1.3B LLaMA model.

The curves of the three optimizers nearly overlap throughout training, indicating that moment pooling has little effect on the convergence trajectory of Adam. AdamPM therefore preserves both the final perplexity and the overall optimization dynamics of vanilla Adam.

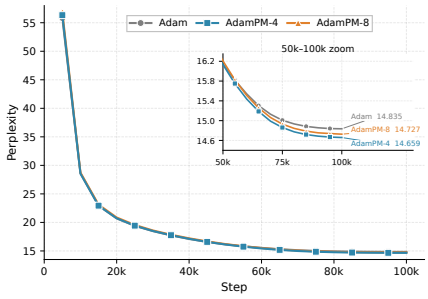


Figure 3: Perplexity comparison between AdamPM and vanilla Adam during LLaMA 1.3B pre-training.

Figure 4 reports the cosine similarity and relative error of the preconditioners and preconditioned updates before and after second moment pooling. Figures 4 (a) and (b) show that, throughout training, the cosine similarities remain consistently high, exceeding 0.993 for most steps, while the relative errors remain low. This indicates that AdamPM closely preserves both the directions and magnitudes of the updates produced by vanilla Adam, explaining their nearly identical convergence curves.

Figure 4 (c) compares the cosine similarities of the preconditioned updates for group sizes $s \in \{2, 4, 8, 16\}$. The similarity gradually decreases as s increases, explaining the slight performance degradation observed with larger group sizes. Figure 4 (d) shows that the cosine similarities increase with model size, suggesting that AdamPM is particularly suitable for larger models. Moreover, Figures 4 (a), (b), and (d) consistently show that the preconditioned updates have higher cosine similarities and lower relative errors than the preconditioners themselves.

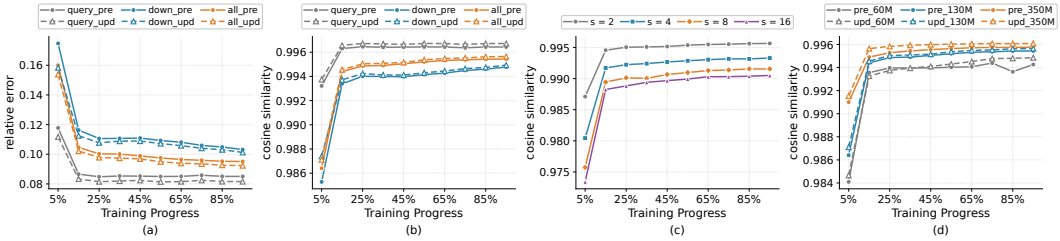


Figure 4: Comparison of the preconditioners $\frac{1}{\sqrt{\widehat{V}_t}+\epsilon}$ (denoted by *pre*) and preconditioned updates $\frac{\widehat{M}_t}{\sqrt{\widehat{V}_t}+\epsilon}$ (denoted by *upd*) before and after applying second moment pooling during LLaMA 130M pre-training. (a) Relative errors of the preconditioners and preconditioned updates for `query` and the MLP down projection matrix `down`, together with the averages across all weight matrices in the network. (b) Corresponding cosine similarities. (c) Average cosine similarity of preconditioned updates across all layers under different group sizes s. (d) Average cosine similarity across all layers for different model sizes.

4.3 Fine-Tuning on GLUE Benchmark

Table 3: Evaluating AdamPM for memory-efficient fine-tuning on GLUE benchmark using pretrained RoBERTa-base. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, F1-score for MRPC, and accuracy for the remaining tasks.

	Memory	CoLA	STS-B	MRPC	RTE	SST2	MNLI	QNLI	QQP	Avg
Adam	748M	64.09	91.18	93.21	79.78	94.49	87.30	92.87	92.06	86.87
Adam-mini	499M	54.68	90.46	86.32	52.70	94.49	86.01	92.34	91.52	81.06
AdamPM-2	623M	**64.29**	91.16	92.28	80.14	94.83	87.28	93.02	91.97	86.87
AdamPM-4	561M	63.26	91.16	**93.52**	79.06	94.83	87.32	**93.15**	**91.99**	86.79
AdamPM-8	529M	63.87	**91.20**	93.28	**82.31**	95.06	**87.53**	92.87	91.86	**87.25**
AdamPM-16	514M	62.92	**91.20**	93.05	79.78	**95.29**	87.36	92.75	91.87	86.78

Experimental setup. We evaluate AdamPM by fine-tuning a pretrained RoBERTa-base (Liu et al., 2019) model on the GLUE benchmark (Wang et al., 2019), using full fine-tuning with Adam and

Adam-mini as baselines. For each task, we select Adam's learning rate by grid search and reuse it for AdamPM and Adam-mini without further search. We vary the group size $s \in \{2, 4, 8, 16\}$. All methods adopt the same hyperparameter strategy to ensure fair comparisons. Detailed experimental settings are provided in Appendix C.2

Result analysis. As shown in the Table 3, AdamPM maintained average scores close to full fine-tuning with Adam across all evaluated group sizes, while memory usage decreased as group size s increased. At $s = 8$, its average score even exceeded the baseline, indicating that more extensive second-moment sharing can offer a favorable memory–performance trade-off. Although Adam-mini used less memory, it performed worse when directly reusing Adam's task-specific learning rates, particularly on smaller-data tasks such as CoLA and RTE. These results suggest that AdamPM can reuse Adam's task-specific learning rates while maintaining competitive performance.

4.4 Experiments on Large Multimodal Models

Table 4: Performance comparison of AdamPM with different group sizes on TinyLLaVA-1B.

	Memory	**VQAv2**	**MM-Vet**	**GQA**	**SQA**	**TextVQA**	**POPE**	**MME**	**MMMU**
Adam	4.38G	68.95	20.6	51.38	54.02	**42.88**	85.22	1330.60	29.80
AdamPM-V-2	3.91G	69.27	**21.6**	51.66	53.88	42.34	85.21	1329.66	30.30
AdamPM-V-4	3.68G	69.55	19.5	52.06	**54.14**	42.41	85.58	**1336.84**	31.10
AdamPM-V-8	3.57G	69.70	19.6	52.23	53.95	42.75	85.61	1320.59	**31.30**
AdamPM-V-16	3.51G	**69.84**	19.2	**52.61**	54.07	42.41	**85.82**	1313.27	31.10

Experimental setup. We evaluate AdamPM on TinyLLaVA-1B Zhou et al. (2024), comprising a SigLIP vision encoder, an MLP2x-GELU projector, and Qwen2-0.5B. We perform supervised fine-tuning on the vision-language instruction-tuning data released by LLaVA Liu et al. (2023). During SFT, the language model and projector are fully fine-tuned, while the vision encoder remains frozen. We compare Adam and AdamPM on VQAv2 Goyal et al. (2017b), MM-Vet Yu et al. (2024), GQA Hudson & Manning (2019), ScienceQA Lu et al. (2022), TextVQA Singh et al. (2019), POPE Li et al. (2023), MME Fu et al. (2025), and MMMU Yue et al. (2024) under identical training settings. Detailed training configurations are provided in Appendix C.3.

Result analysis. As shown in Table 4, AdamPM substantially reduces static memory while maintaining comparable overall performance; AdamPM-16 saves 19.9% memory over Adam and obtains the highest VQAv2, GQA, and POPE scores among the AdamPM variants. Different grouping sizes perform best on different benchmarks, demonstrating that AdamPM provides a flexible trade-off between memory efficiency and multimodal performance.

5 Conclusion

We introduced AdamPM, a simple memory-efficient Adam variant that only pools second moment estimates. AdamPM reuses Adam's hyperparameters, remains model-agnostic, and incurs negligible training overhead. Our analysis shows that its preconditioners and parameter updates remain highly similar to Adam's, particularly at larger model scales. Experiments on LLM pre-training, fine-tuning, and multimodal models demonstrate substantial memory savings while preserving Adam-level performance and training efficiency. These results establish second moment pooling used in AdamPM as a simple and practical approach to memory-efficient LLM training.

Limitations. Due to computational constraints, we have not conducted full training runs on models with 7B or more parameters. Our study considers pooling only along a single direction, either row-wise or column-wise. Pooling over two-dimensional groups and selective pooling that retains elementwise second moments in certain layers or modules, along with their effects on optimization, remain unexplored.

AI USE STATEMENT

In this work, we used generative AI tools solely for language polishing to improve the clarity and readability of the manuscript. We have not used generative AI tools for any other research tasks. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Our experiments use open-source model implementations and publicly available datasets for training and evaluation. To facilitate reproducibility, we provide the AdamPM implementation and experimental code in the supplementary materials.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. URL https://arxiv.org/abs/1607.06450.

Xi Chen, Kaituo Feng, Changsheng Li, Xunhao Lai, Xiangyu Yue, Ye Yuan, and Guoren Wang. Fira: Can we achieve full-rank training of LLMs under low-rank constraint? In *Advances in Neural Information Processing Systems*, volume 38, pp. 120680–120712, 2025.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. Symbolic discovery of optimization algorithms. In *Advances in Neural Information Processing Systems*, volume 36, pp. 49205–49233, 2023.

Chaoyou Fu, Peixian Chen, Yunhang Shen, Yulei Qin, Mengdan Zhang, Xu Lin, Jinrui Yang, Xiawu Zheng, Ke Li, Xing Sun, Yunsheng Wu, Rongrong Ji, Caifeng Shan, and Ran He. MME: A comprehensive evaluation benchmark for multimodal large language models. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Athanasios Glentis, Jiaxiang Li, Andi Han, and Mingyi Hong. A minimalist optimizer design for LLM pretraining. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*, 2025.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017a. URL https://arxiv.org/abs/1706.02677.

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the V in VQA matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6325–6334. IEEE Computer Society, 2017b.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Drew A. Hudson and Christopher D. Manning. GQA: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6700–6709, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. URL https://arxiv.org/abs/1412.6980.

Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Wayne Xin Zhao, and Ji-Rong Wen. Evaluating object hallucination in large vision-language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 292–305. Association for Computational Linguistics, 2023.

Kaizhao Liang, Bo Liu, Lizhang Chen, and Qiang Liu. Memory-efficient LLM training with online subspace descent. In *Advances in Neural Information Processing Systems*, volume 37, pp. 64412–64432, 2024.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *NeurIPS*, 2023.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for LLM training. *arXiv preprint arXiv:2502.16982*, 2025. URL https://arxiv.org/abs/2502.16982.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019. URL https://arxiv.org/abs/1907.11692.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. In *Advances in Neural Information Processing Systems*, 2022.

Yang Luo, Xiaozhe Ren, Zangwei Zheng, Zhuo Jiang, Xin Jiang, and Yang You. CAME: Confidence-guided adaptive memory efficient optimization. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4442–4453, 2023.

Zhanfeng Mo, Long-Kai Huang, and Sinno Jialin Pan. Parameter and memory efficient pretraining via low-rank Riemannian optimization. In *International Conference on Learning Representations*, 2025.

Kwangryeol Park and Seulki Lee. SMMF: Square-matricized momentum factorization for memory-efficient optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 19848–19856, 2025.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Yehonathan Refael, Guy Smorodinsky, Tom Tirer, and Ofir Lindenbaum. SUMO: Subspace-aware moment-orthogonalization for accelerating memory-efficient LLM training. In *Advances in Neural Information Processing Systems*, volume 38, pp. 147250–147281, 2025.

Thomas Robert, Mher Safaryan, Ionut-Vlad Modoranu, and Dan Alistarh. Ldadam: Adaptive optimization from low-dimensional gradient statistics. In *International Conference on Learning Representations*, 2025.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4596–4604. PMLR, 2018.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. Towards VQA models that can read. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8317–8326. IEEE, 2019.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. URL https://arxiv.org/abs/2302.13971.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJ4km2R5t7.

Zixiao Wang, Yifei Shen, and Huishuai Zhang. OLion: Approaching the hadamard ideal by intersecting spectral and ℓ_∞ implicit biases. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Kaiyue Wen, David Hall, Tengyu Ma, and Percy Liang. Fantastic pretraining optimizers and where to find them. In *International Conference on Learning Representations*, 2026a.

Ziqing Wen, Jiahuan Wang, Ping Luo, Dongsheng Li, and Tao Sun. FOAM: Blocked state folding for memory-efficient LLM training. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026b. URL https://arxiv.org/abs/2512.07112.

Weihao Yu, Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Zicheng Liu, Xinchao Wang, and Lijuan Wang. MM-Vet: Evaluating large multimodal models for integrated capabilities. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 57730–57754. PMLR, 2024. URL https://proceedings.mlr.press/v235/yu24o.html.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. MMMU: A massive multi-discipline multimodal understanding and reasoning benchmark for expert AGI. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9556–9567, 2024.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Diederik P. Kingma, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. Adam-mini: Use fewer learning rates to gain more. In *International Conference on Learning Representations*, 2025.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. GaLore: Memory-efficient LLM training by gradient low-rank projection. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 61121–61143. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhao24s.html.

Baichuan Zhou, Ying Hu, Xi Weng, Junlong Jia, Jie Luo, Xien Liu, Ji Wu, and Lei Huang. TinyLLaVA: A framework of small-scale large multimodal models. *arXiv preprint arXiv:2402.14289*, 2024. URL https://arxiv.org/abs/2402.14289.

Hanqing Zhu, Zhenyu Zhang, Wenyan Cong, Xi Liu, Sem Park, Vikas Chandra, Bo Long, David Z. Pan, Zhangyang Wang, and Jinwon Lee. APOLLO: SGD-like memory, AdamW-level performance. In *Proceedings of Machine Learning and Systems*, volume 7, 2025.

Philip Zmushko, Aleksandr Beznosikov, Martin Takáč, and Samuel Horváth. FRUGAL: Memory-efficient optimization by reducing state overhead for scalable training. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 80708–80739. PMLR, 2025.

A LEMMAS AND PROOFS

Our analysis draws on the momentum-error decomposition and descent-analysis framework used in FOAM (Wen et al., 2026b). We derive bounds tailored to AdamPM's grouped second moments, explicitly accounting for the random preconditioner and accumulated gradient noise.

A.1 SETUP AND ASSUMPTIONS

We identify all parameters with $W \in \mathbb{R}^d$ and use the Euclidean norm; matrix parameters are equivalently equipped with the Frobenius norm. We state the assumptions used throughout the analysis.

1. *L-Smoothness and lower boundedness.* The differentiable objective $f : \mathbb{R}^d \to \mathbb{R}$ satisfies
$$f(W) \geq f_* > -\infty,$$
and
$$\|\nabla f(W) - \nabla f(W')\|_2 \leq L\|W - W'\|_2, \quad W, W' \in \mathbb{R}^d.$$
2. *Conditional unbiasedness and bounded noise variance.* For every t,
$$G_t = \nabla f(W_t) + \xi_t, \quad \mathbb{E}[\xi_t \mid W_t] = 0, \quad \mathbb{E}\|\xi_t\|_2^2 \leq \sigma^2.$$
3. *Bounded true gradients.* For every t, $\|\nabla f(W_t)\|_2 \leq C$ almost surely.

Let P_s average and broadcast coordinates within each fixed row-wise group of at most s entries. For any such group I,
$$(P_s z)_i = \frac{1}{|I|} \sum_{j \in I} z_j, \quad i \in I.$$
A final partial group is averaged over its actual entries. Padding that group with its own mean and discarding the padding gives the same operator. Ungrouped coordinates are represented by singleton groups. Starting from deterministic W_1 and $M_0 = V_0 = 0$, AdamPM performs
$$\begin{aligned} M_t &= \beta_1 M_{t-1} + (1-\beta_1) G_t, \\ V_t &= \beta_2 V_{t-1} + (1-\beta_2) P_s(G_t^{\odot 2}), \\ \widehat{M}_t &= \frac{M_t}{1-\beta_1^t}, \quad \widehat{V}_t = \frac{V_t}{1-\beta_2^t}, \\ W_{t+1} &= W_t - \eta_t \frac{\widehat{M}_t}{\sqrt{\widehat{V}_t} + \epsilon}. \end{aligned} \tag{5}$$
All vector operations are elementwise, and $\odot 2$ denotes elementwise squaring. The parameters $\beta_1, \beta_2 \in [0, 1)$ and $\epsilon > 0$ are fixed independently of T.

Define
$$H_t = \operatorname{diag}\left(\frac{1}{\sqrt{\widehat{V}_{t,i}} + \epsilon} \right)_{i=1}^{d}. \tag{6}$$
Then $W_{t+1} = W_t - \eta_t H_t \widehat{M}_t$ and $0 \prec H_t \preceq \epsilon^{-1} I$.

A.2 MOMENT AND PRECONDITIONER BOUNDS

Lemma A.1. *For every $t \geq 1$,*
$$\mathbb{E}\|G_t\|_2^2 \leq C^2 + \sigma^2, \quad \mathbb{E}\|\widehat{M}_t\|_2^2 \leq C^2 + \sigma^2, \quad \mathbb{E}\|H_t \widehat{M}_t\|_2^2 \leq \frac{C^2 + \sigma^2}{\epsilon^2}. \tag{7}$$
Define the accumulated noise energy
$$Z_t = \frac{1-\beta_2}{1-\beta_2^t} \sum_{j=1}^{t} \beta_2^{t-j} \|\xi_j\|_2^2. \tag{8}$$

Then $\mathbb{E}Z_t \le \sigma^2$*, and, almost surely,*

$$\nabla f(W_t)^\top H_t \nabla f(W_t) \ge \frac{\|\nabla f(W_t)\|_2^2 - Z_t}{2C + \epsilon}. \tag{9}$$

Proof. Conditional unbiasedness gives $\mathbb{E}\langle \nabla f(W_t), \xi_t \rangle = 0$. Consequently,

$$\mathbb{E}\|G_t\|_2^2 = \mathbb{E}\|\nabla f(W_t)\|_2^2 + \mathbb{E}\|\xi_t\|_2^2 \le C^2 + \sigma^2.$$

The bias-corrected first moment satisfies

$$\widehat{M}_t = \sum_{j=1}^{t} w_{t,j} G_j, \quad w_{t,j} = \frac{(1-\beta_1)\beta_1^{t-j}}{1-\beta_1^t}. \tag{10}$$

Jensen's inequality and $H_t \preceq \epsilon^{-1} I$ prove Eq. 7. The same convex-combination argument gives $\mathbb{E}Z_t \le \sigma^2$.

The inequality $(a+b)^2 \le 2a^2 + 2b^2$ and group averaging yield

$$\begin{aligned} \big(P_s(G_j^{\odot 2})\big)_i &\le 2\big(P_s(\nabla f(W_j)^{\odot 2})\big)_i + 2\big(P_s(\xi_j^{\odot 2})\big)_i \\ &\le 2C^2 + 2\|\xi_j\|_2^2. \end{aligned}$$

Here, each group average is bounded by the squared norm of the full vector. Applying the bias-corrected second-moment weights gives

$$\widehat{V}_{t,i} \le 2C^2 + 2Z_t. \tag{11}$$

On $\{Z_t \le C^2\}$, every denominator is at most $2C + \epsilon$, which implies Eq. 9. On $\{Z_t > C^2\}$, its right-hand side is nonpositive, whereas its left-hand side is nonnegative. □

A.3 Momentum error

Decompose the first-moment error as

$$\begin{aligned} e_t &= \widehat{M}_t - \nabla f(W_t) = a_t + n_t, \\ a_t &= \sum_{j=1}^{t} w_{t,j}\big(\nabla f(W_j) - \nabla f(W_t)\big), \\ n_t &= \sum_{j=1}^{t} w_{t,j}\xi_j. \end{aligned} \tag{12}$$

Lemma A.2. *For* $\eta_t = \frac{\eta_0}{\sqrt{t}}$*, define*

$$C_{\mathrm{m}} = \frac{L\eta_0\sqrt{C^2+\sigma^2}}{\epsilon}\,\frac{\beta_1\sqrt{2-\beta_1}}{(1-\beta_1)^{3/2}}. \tag{13}$$

Then, for every $t \ge 1$*,*

$$\begin{aligned} \mathbb{E}\|a_t\|_2^2 &\le \frac{C_{\mathrm{m}}^2}{t}, \\ \mathbb{E}\|n_t\|_2^2 &\le \sum_{j=1}^{t} w_{t,j}\mathbb{E}\|\xi_j\|_2^2 \le \sigma^2. \end{aligned} \tag{14}$$

In particular,

$$\mathbb{E}\|e_t\|_2^2 \le \frac{2C_{\mathrm{m}}^2}{t} + 2\sigma^2. \tag{15}$$

Proof. Expanding the gradient differences and exchanging finite sums gives

$$a_t = \sum_{r=1}^{t-1}\left(\sum_{j=1}^{r} w_{t,j}\right)\big(\nabla f(W_r) - \nabla f(W_{r+1})\big).$$

Moreover, for $r < t$,

$$\sum_{j=1}^{r} w_{t,j} = \beta_1^{t-r} \frac{1-\beta_1^r}{1-\beta_1^t} \le \beta_1^{t-r}.$$

Smoothness, the L^2 triangle inequality, and Eq. 7 therefore imply

$$\begin{aligned} \left(\mathbb{E}\|a_t\|_2^2\right)^{1/2} &\le L \sum_{r=1}^{t-1} \eta_r \left(\sum_{j=1}^{r} w_{t,j} \right) \left(\mathbb{E}\|H_r \widehat{M}_r\|_2^2\right)^{1/2} \\ &\le \frac{L\sqrt{C^2+\sigma^2}}{\epsilon} \sum_{r=1}^{t-1} \eta_r \beta_1^{t-r}. \end{aligned} \tag{16}$$

For $1 \le k < t$, the inequality $t/(t-k) \le k+1$ gives

$$\sum_{r=1}^{t-1} \frac{\beta_1^{t-r}}{\sqrt{r}} \le \frac{1}{\sqrt{t}} \sum_{k=1}^{\infty} \beta_1^k \sqrt{k+1}.$$

By Cauchy–Schwarz and geometric-series identities,

$$\begin{aligned} \sum_{k=1}^{\infty} \beta_1^k \sqrt{k+1} &\le \left(\sum_{k=1}^{\infty} \beta_1^k \right)^{1/2} \left(\sum_{k=1}^{\infty} (k+1) \beta_1^k \right)^{1/2} \\ &= \frac{\beta_1 \sqrt{2-\beta_1}}{(1-\beta_1)^{3/2}}. \end{aligned}$$

This proves the drift bound. Jensen's inequality gives

$$\mathbb{E}\|n_t\|_2^2 \le \sum_{j=1}^{t} w_{t,j} \mathbb{E}\|\xi_j\|_2^2.$$

Finally, use $\|a_t + n_t\|_2^2 \le 2\|a_t\|_2^2 + 2\|n_t\|_2^2$. □

A.4 Proof of Theorem 3.1

Proof. Young's inequality in the inner product induced by H_t yields

$$\begin{aligned} \langle \nabla f(W_t), H_t \widehat{M}_t \rangle &\ge \frac{1}{2} \nabla f(W_t)^\top H_t \nabla f(W_t) - \frac{1}{2} e_t^\top H_t e_t \\ &\ge \frac{\|\nabla f(W_t)\|_2^2 - Z_t}{2(2C+\epsilon)} - \frac{\|e_t\|_2^2}{2\epsilon}. \end{aligned} \tag{17}$$

Applying smoothness to Eq. 5 and taking expectations therefore gives

$$\begin{aligned} \frac{\eta_t}{2(2C+\epsilon)} \mathbb{E}\|\nabla f(W_t)\|_2^2 \le \mathbb{E}[f(W_t) - f(W_{t+1})] + \frac{\eta_t \mathbb{E} Z_t}{2(2C+\epsilon)} \\ + \frac{\eta_t}{2\epsilon} \mathbb{E}\|e_t\|_2^2 + \frac{L(C^2+\sigma^2)}{2\epsilon^2} \eta_t^2. \end{aligned} \tag{18}$$

Summing and using lower boundedness gives the common starting point for both results:

$$\begin{aligned} \sum_{t=1}^{T} \eta_t \mathbb{E}\|\nabla f(W_t)\|_2^2 \le 2(2C+\epsilon)\big(f(W_1) - f_*\big) + \sum_{t=1}^{T} \eta_t \mathbb{E} Z_t \\ + \frac{2C+\epsilon}{\epsilon} \sum_{t=1}^{T} \eta_t \mathbb{E}\|e_t\|_2^2 \\ + \frac{L(2C+\epsilon)(C^2+\sigma^2)}{\epsilon^2} \sum_{t=1}^{T} \eta_t^2. \end{aligned} \tag{19}$$

Substituting $\mathbb{E}Z_t \le \sigma^2$ and Eq. 15, we obtain

$$\sum_{t=1}^{T} \eta_t \mathbb{E}\|\nabla f(W_t)\|_2^2 \le 2(2C+\epsilon)\big(f(W_1)-f_*\big) + \left(3+\frac{4C}{\epsilon}\right)\sigma^2 \sum_{t=1}^{T}\eta_t + \frac{2(2C+\epsilon)C_{\mathrm{m}}^2}{\epsilon}\sum_{t=1}^{T}\frac{\eta_t}{t} + \frac{L(2C+\epsilon)(C^2+\sigma^2)}{\epsilon^2}\sum_{t=1}^{T}\eta_t^2. \tag{20}$$

For $\eta_t = \frac{\eta_0}{\sqrt{t}}$, elementary sum bounds give

$$\sum_{t=1}^{T}\eta_t \ge \eta_0\sqrt{T}, \quad \sum_{t=1}^{T}\frac{\eta_t}{t} \le 3\eta_0, \quad \sum_{t=1}^{T}\eta_t^2 \le \eta_0^2(1+\log T). \tag{21}$$

Indeed, $t^{-1/2} \ge T^{-1/2}$ for $t \le T$, $\sum_{t\ge 1} t^{-3/2} \le 1+\int_1^\infty x^{-3/2}\,dx = 3$, and $\sum_{t=1}^{T} t^{-1} \le 1+\log T$. Divide Eq. 20 by $\sum_{t=1}^{T}\eta_t$ and use

$$\min_{1\le t\le T}\mathbb{E}\|\nabla f(W_t)\|_2^2 \le \frac{\sum_{t=1}^{T}\eta_t\mathbb{E}\|\nabla f(W_t)\|_2^2}{\sum_{t=1}^{T}\eta_t}.$$

Equation 2 follows with

$$K = \frac{2(2C+\epsilon)\big(f(W_1)-f_*\big)}{\eta_0} + \frac{6(2C+\epsilon)C_{\mathrm{m}}^2}{\epsilon} + \frac{L\eta_0(2C+\epsilon)(C^2+\sigma^2)}{\epsilon^2}. \tag{22}$$

For common assumption constants and initial objective gap, this expression is independent of T, β_2, and the fixed grouping. □

A.5 A WEIGHTED-HISTORY BOUND

Lemma A.3. *For $\beta \in [0,1)$, set*

$$w_{t,j}^{(\beta)} = \frac{(1-\beta)\beta^{t-j}}{1-\beta^t}.$$

For $\eta_t = \frac{\eta_0}{\sqrt{t}}$ and any nonnegative sequence (u_j),

$$\sum_{t=1}^{T}\eta_t\sum_{j=1}^{t} w_{t,j}^{(\beta)}u_j \le 4\sum_{j=1}^{T}\eta_j u_j. \tag{23}$$

The constant is uniform over $\beta \in [0,1)$.

Proof. For $0<\beta<1$,

$$w_{t,j}^{(\beta)} = (1-\beta)\beta^{t-j} + \beta^{t-j}\frac{(1-\beta)\beta^t}{1-\beta^t}.$$

Since

$$\frac{(1-\beta)\beta^t}{1-\beta^t} = \frac{\beta^t}{\sum_{r=0}^{t-1}\beta^r} \le \frac{\beta}{t} \le \frac{1}{t},$$

we have $w_{t,j}^{(\beta)} \le (1-\beta)\beta^{t-j} + 1/t$. Using the monotonicity of η_t and an integral bound,

$$\sum_{t=j}^{T}\eta_t w_{t,j}^{(\beta)} \le \eta_j(1-\beta)\sum_{r=0}^{\infty}\beta^r + \eta_0\sum_{t=j}^{\infty}t^{-3/2} \le \eta_j + \eta_0\big(j^{-3/2}+2j^{-1/2}\big) \le 4\eta_j.$$

Exchanging the finite sums proves Eq. 23. □

A.6 Proof of Corollary 3.2

Proof. Let

$$S_T = \sum_{t=1}^{T} \eta_t \mathbb{E}\|\nabla f(W_t)\|_2^2.$$

Taking expectations in Eq. 3 gives $\mathbb{E}\|\xi_t\|_2^2 \le \kappa \mathbb{E}\|\nabla f(W_t)\|_2^2$. Apply Lemma A.3 to the second-moment weights and then to the first-moment weights in Eq. 14:

$$\sum_{t=1}^{T} \eta_t \mathbb{E} Z_t \le 4\kappa S_T, \quad \sum_{t=1}^{T} \eta_t \mathbb{E}\|n_t\|_2^2 \le 4\kappa S_T. \tag{24}$$

Combining these bounds with the drift estimate yields

$$\begin{aligned} \sum_{t=1}^{T} \eta_t \mathbb{E}\|e_t\|_2^2 &\le 2C_{\mathrm{m}}^2 \sum_{t=1}^{T} \frac{\eta_t}{t} + 8\kappa S_T \\ &\le 6\eta_0 C_{\mathrm{m}}^2 + 8\kappa S_T. \end{aligned} \tag{25}$$

Substitution into Eq. 19, before replacing $\mathbb{E} Z_t$ by a uniform variance bound, gives

$$\begin{aligned} &\left[1 - 4\kappa\left(3 + \frac{4C}{\epsilon}\right)\right] S_T \\ &\quad \le 2(2C+\epsilon)\big(f(W_1) - f_*\big) + \frac{6\eta_0(2C+\epsilon)C_{\mathrm{m}}^2}{\epsilon} \\ &\qquad + \frac{L(2C+\epsilon)(C^2+\sigma^2)}{\epsilon^2} \sum_{t=1}^{T} \eta_t^2. \end{aligned} \tag{26}$$

The coefficient on the left is positive by Eq. 3. The right-hand side is at most $\eta_0 K(1 + \log T)$ by Eqs. 21 and 22. Dividing by this coefficient and by $\sum_{t=1}^{T} \eta_t$ proves Eq. 4. □

B Optimizer Instantiation

The following code illustrates how to instantiate the optimizers used in our experiments, highlighting AdamPM's ease of use.

```
# Adam
optimizer = Adam(
  trainable_params,
  lr=args.lr,
  betas=(args.beta1, args.beta2),
)

# AdamPM
optimizer = AdamPM(
  trainable_params,
  lr=args.lr,
  betas=(args.beta1, args.beta2),
)

# Adam-mini
optimizer = Adam_mini(
  model.named_parameters(),
  lr=args.lr,
  betas=(args.beta1, args.beta2),
  dim=model_config.hidden_size,
  n_heads=model_config.num_attention_heads,
)
```

```
# FOAM
optimizer = FOAM(
  param_groups,
  lr=args.lr,
  betas=(args.beta1, args.beta2),
  res_scale=args.res_scale,
  no_norm_limit=args.no_norm_limit,
)

# GaLore
optimizer = GaLoreAdamW(
  param_groups,
  lr=args.lr,
  betas=(args.beta1, args.beta2),
)
```

C Details of the experiments

C.1 Details of LLaMA pre-training

Hyperparameters. Table A1 summarizes the architectural and optimization hyperparameters for LLaMA pre-training of different sizes. For each model size, Adam, Adam-mini, and AdamPM use identical hyperparameter settings, regardless of the AdamPM variant or group size. The learning rates for the 60M, 130M, 350M, and 1.3B models are 4×10^{-3}, 2×10^{-3}, 6×10^{-4}, and 3×10^{-4}, respectively. These values are selected for their robustness based on learning-rate sweeps for Adam, such as those shown in Figure 2. Across all model sizes, we set $\beta_1 = 0.9$, $\beta_2 = 0.95$, and $\epsilon = 10^{-8}$.

We employ a cosine learning rate decay schedule (Loshchilov & Hutter, 2017) with learning rate warmup (Goyal et al., 2017a). The first 10% of the total training steps are used for warmup, followed by cosine decay for the remaining steps. Weight decay is disabled in all experiments.

Table A1: The hyperparameters for LLaMA pre-training.

Model Size	**60M**	**130M**	**350M**	**1.3B**
Hidden dimension	512	768	1024	2048
Intermediate dimension	1376	2048	2736	5461
Attention Heads	8	12	16	32
Decoder layers	8	12	24	24
Learning Rate	4×10^{-3}	2×10^{-3}	6×10^{-4}	3×10^{-4}
β_1	0.9			
β_2	0.95			
ϵ	10^{-8}			
Weight Decay	0			
Learning Rate Scheduler	*Cosine*			
Warmup	10%			
Max seq. len.	256			

Training Efficiency. The implementation of AdamPM performs group-wise second moment pooling entirely through matrix-level tensor operations, including reshaping, averaging, and broadcasting, enabling efficient vectorized computation without element-wise Python `for` loops. Its computational cost is therefore nearly identical to that of vanilla Adam. On the pre-training of LLaMA 1.3B with a batch size of 32 and 4× NVIDIA A100 40GB GPUs, vanilla Adam and AdamPM require 116.2 and 116.4 hours, respectively, confirming that AdamPM introduces negligible training-time overhead. Moreover, the reduced memory footprint may allow a larger batch size under the same hardware constraints, improving hardware utilization and further accelerating training.

Actual training memory usage. We measure the actual training memory consumption per GPU for Adam and AdamPM during LLaMA-1.3B pre-training on NVIDIA A100 40GB GPUs with a batch size of 32. The results are reported in Table A2.

Table A2: Measured memory usage per GPU during LLaMA-1.3B pre-training.

Optimizer	Memory Cost
Adam	38.11 G
AdamPM-2	36.73 G
AdamPM-4	35.96 G
AdamPM-8	35.43 G
AdamPM-16	35.19 G
AdamPM-inf	35.14 G

Memory evaluation on LLaMA 7B. Due to limited computational resources, we do not perform a complete pre-training run on the LLaMA 7B model. Instead, we evaluate the static memory footprint of AdamPM at the 7B scale, including the model parameters and optimizer states under BF16 precision. Adam requires approximately 39.12GB, whereas AdamPM-2 and AdamPM-inf require only approximately 32.60GB and 26.08GB, respectively.

C.2 Details of Fine-Tuning on GLUE

For each GLUE task, we select the best-performing learning rate for Adam when fine-tuning RoBERTa-base by searching over $\{1\times10^{-5}, 2\times10^{-5}, 3.5\times10^{-5}, 5\times10^{-5}, 7.5\times10^{-5}, 1\times10^{-4}\}$. We use the selected task-specific learning rate for AdamPM and Adam-mini without further tuning and report the performance of all three optimizers. The fine-tuning hyperparameters are summarized in Table A3

Table A3: Hyperparameters for fine-tuning RoBERTa-base on GLUE.

Hyperparameters	**MNLI**	**SST-2**	**MRPC**	**CoLA**	**QNLI**	**QQP**	**RTE**	**STS-B**
Batch size	64	32	16	32	64	64	16	32
Epochs	30	30	30	30	30	30	30	30
lr	2e-5	1e-5	2e-5	2e-5	1e-5	2e-5	2e-5	2e-5
β_1	0.9							
β_2	0.95							
Max seq. len.	512							

C.3 Details of TinyLLaVA-1B Fine-Tuning

Following the setup described in the main text, all methods are initialized from the same pretrained checkpoint and differ only in the SFT optimizer. We perform SFT for one epoch with a global batch size of 128, a learning rate of 2×10^{-5}, $\beta_1 = 0.9$, $\beta_2 = 0.95$, and a maximum sequence length of 2048. Both the training seed and data-order seed are fixed to 42.