NOT ALL BLOCKS NEED FULL MOMENTUM: SHARPNESS-AWARE MEMORY-EFFICIENT OPTIMIZATION FOR TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Momentum is a key component of modern optimizers, but maintaining full-rank momentum for every parameter is increasingly costly for large language model training. We show that momentum is not uniformly valuable across Transformer blocks: different blocks exhibit substantially different local optimization sensitivities, and consequently require different levels of momentum capacity. Motivated by this observation, we propose AdaPM (Adaptive Partial Momentum), a sharpness-aware framework that allocates momentum non-uniformly across parameter blocks. Rather than treating momentum as an all-or-nothing resource, AdaPM assigns each block to one of three regimes—no momentum, low-rank momentum, or full momentum—according to its local sharpness and empirical momentum dependency. For blocks where the dominant optimization dynamics are low-dimensional, we further introduce a residual-corrected low-rank momentum estimator that compensates for information lost by rank truncation while retaining a small memory footprint. This yields a simple principle for optimizer-state reduction: match momentum capacity to the local sensitivity and optimization structure of each block, rather than applying full momentum uniformly. Across GPT-2 and LLaMA pretraining and fine-tuning tasks, AdaPM reduces momentum memory by over 90% while maintaining competitive training performance. When combined with memory-efficient second-order statistics, it reduces total optimizer-state memory by up to 95% and substantially improves training efficiency.[1]

1 INTRODUCTION

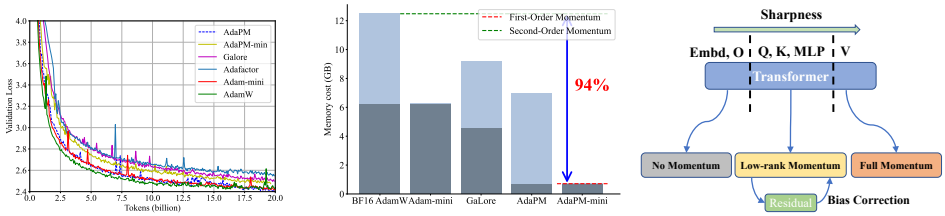
Efficient optimizers, working as engines, are a key factor in the success and boom of modern large language models (LLMs) [34; 1; 33; 22]. However, besides accelerating training, the optimizers also introduce significant memory overhead, posing a major challenge to limited memory resources. A typical example is the widely-used Adam optimizer [16], which requires two additional sets of values—the first- and second-order statistics estimators for every parameter—thereby significantly increasing the demand for device memory. As model sizes increase, model performance continues to improve, yet the memory occupied by optimizer states alone becomes dominant [39], creating substantial implementation and time-consuming challenges.

The challenges have spurred significant interest in designing memory-efficient optimizers. The pursuit of these optimizers is driven by dual benefits [39]: they circumvent the limitations on model size and enable larger batch sizes during parallel training, thereby lowering communication overhead, and directly speeding up optimization.

Current approaches to reducing optimizer states primarily focus on second-order statistics, with comparatively less progress made in improving first-order statistics. This asymmetry stems from their distinct roles. Second-order statistics are non-negative and serve as per-parameter scale estimators. Adam-mini[38] shows that they typically exhibit near-uniform scales within architectural units such as blocks or neurons. This reducibility has motivated methods like Adafactor [31], which significantly compress second-order statistics.

[1] Our implementation for AdaPM will be released at github.

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.



(a) Loss v.s. iteration (b) Memory cost (c) An illustration of AdaPM

Figure 1: AdaPM takes less memory and can reach higher throughput with on par or better performance than AdamW. (a) Results for GPT-2 1.5B pre-training. (b) The memory cost when training GPT-2 1.5B with various optimizers. The experimental details are shown in Section 5.2. (c) AdaPM assigns different momentum designs to different blocks and enhances the partial momentum using a bias- corrected approach.

In contrast, first-order statistics—known as momentum in optimization theory—are widely recognized for accelerating convergence [32]. However, they are signed and highly sensitive in governing function value descent [17; 8]. Attempts to reduce first-order statistics, such as the low-rank update algorithms [6; 39] often lead to non-negligible performance degradation during pretraining. Consequently, whether first-order statistics exhibit similar reducibility as second-order states and can be substantially compressed without compromising performance remains an open and important question.

We propose the method, **Ada**ptive **P**artial **M**omentum, AdaPM, which AdaPM (Adaptive Partial Momentum) cuts first-order statistics memory without hurting performance. Motivated by transformer component heterogeneity [37], we find most blocks don't need full momentum acceleration; since momentum is approximately low-rank with a nontrivial singular-value tail, naive truncation (as in Galore [39]) slows convergence. We address this via residual compensation, rescaling the discarded full-vs-approximate momentum residual to recover lost descent directions (Section 4.2). AdaPM can be combined with second-order compression approaches like Adam-mini [38] for substantial overall memory reduction. On GPT-2 and LLaMA models (124M to 8B), AdaPM matches or exceeds AdamW's performance using only 10% of the momentum memory; combined with Adam-mini it achieves a remarkable 94% memory saving on the optimizer state without sacrificing convergence (Fig. 1). Moreover, due to higher throughput and larger batch sizes, AdaPM saves over 30% GPU hours for pretraining.

To summarize our contribution:

- L **as a predictor of momentum dependency.** Ablations on partial momentum allocation empirically validate the L disparity principle across Transformer blocks: the need for full momentum correlates strongly with the local loss landscape's L.
- **Partition of Transformer blocks.** We study how momentum acceleration needs vary across blocks: (1) embedding and attn.proj need no momentum acceleration; (2) query, key, and MLP need only low-rank momentum; (3) value needs full momentum.
- **Bias-corrected low-rank momentum estimation.** A novel debiasing method yields an unbiased estimate of full momentum from low-rank momentum using only 5% of the original dimensions.
- **Memory-efficient optimizer.** Combining the partition principle with debiased low-rank estimation, we propose a framework that substantially cuts Adam's memory overhead; paired with methods like Adam-mini, it saves over 95% of optimizer-state memory without performance loss.

2 Related Works

Lightweight optimizers focusing on second-order statistics. Several lightweight optimization algorithms have been developed to reduce computational and memory costs by leveraging second-order statistical information. Adafactor [31] and CAME [24] are memory-efficient variants of Adam employing nonnegative low-rank factorization over Adam's second-order statistics. SM3 [2]

employs a candidate set, which is derived from the maximum squared gradient within a specific group of parameters defined by a cover of the parameters, and determines the learning rate for the i-th parameter by selecting the smallest value from the candidate set. Adam-mini [38] partitions the parameters into blocks, a proposed principle on Hessian structure, and assigns a uniform learning rate for each block using the average of Adam's second-order statistics in that block.

Lightweight optimizers focusing on subspace learning. Recent research has shown that the learning process primarily takes place within a lower-dimensional subspace of the parameter space [11]. Research, such as Gooneratne et al. [10]; Yang et al. [36]; Chen et al. [5]; Zhu et al. [40], has applied the low-rank property of gradients during the training of neural networks to reduce the memory footprint during training. A similar approach has been widely used in meta-learning and continual learning [18; 4]. GaLore [39] and a novel variant, Golore [12], calculate a low-rank gradient estimator and then calculate the first- and second-order statistics on this low-rank gradient estimator. Fira[5], APOLLO [40] and SWAN [25] claim that they achieving AdamW-level performance while nearly eliminating AdamW's optimizer state and reducing memory overhead to that of SGD. However, these methods treat each blocks in Transformer equally, overlooking the fact that the difficulty of optimizing different blocks varies, thus leading to suboptimal performance.

3 Starting Point of Momentum Reduction

In this section, we present a detailed discussion on the potential reducibility of the momentum. We present the theoretical justification and empirical evidence in Section 3.1 and Section 3.2, respectively.

3.1 Theoretical Justification

Our first illustration of reducibility in momentum is posed by the following question: Does adding momentum in optimization consistently lower the validation loss? To investigate, we consider the standard problem of minimizing the validation risk $\mathcal{R}(\mathbf{W}) = \mathbb{E}_{\mathbf{x},y\sim\mathcal{D}}\ell(\mathbf{W};(\mathbf{x},y))$. Let $\mathbf{W}^* \in \arg\min_{\mathbf{W}} \mathcal{R}(\mathbf{W})$ denote the minimizer, and let $\hat{\mathbf{W}}_{\text{opt}}$ denote the output of the optimization algorithm with stochastic gradients.

The output of a stochastic optimization algorithm can be decomposed into a mean behavior and a stochastic deviation around it:

$$\mathcal{R}(\hat{\mathbf{W}}_{\text{opt}}) - \mathcal{R}(\mathbf{W}^*) = \underbrace{\mathcal{R}(\hat{\mathbf{W}}_{\text{opt}}) - \mathcal{R}(\bar{\mathbf{W}}_{\text{opt}})}_{\text{stochastic deviation } \mathcal{A}} + \underbrace{\mathcal{R}(\bar{\mathbf{W}}_{\text{opt}}) - \mathcal{R}(\mathbf{W}^*)}_{\text{optimization bias } \mathcal{B}}, \tag{1}$$

where $\bar{\mathbf{W}}_{\text{opt}} = \mathbb{E}[\hat{\mathbf{W}}_{\text{opt}}]$. Momentum accelerates the deterministic term $\mathcal{B}$ [28; 26], but does not necessarily suppress $\mathcal{A}$: temporal accumulation of stochastic gradients can amplify its steady-state variance.

Under a local quadratic approximation $\mathcal{R}(\mathbf{W}) \approx \mathcal{R}(\mathbf{W}^*) + \frac{1}{2}(\mathbf{W}-\mathbf{W}^*)^\top \mathbf{H}(\mathbf{W}-\mathbf{W}^*)$ with curvature λ_i and gradient-noise variance σ_i^2 along the i-th eigendirection, a second-order expansion around the mean iterate gives

$$\mathcal{A} \approx \frac{1}{2}\sum_i \lambda_i \operatorname{Var}(\hat{W}^i_{\text{opt}}), \tag{2}$$

so the stochastic error is explicitly governed by the local curvature spectrum.

Theorem 3.1 (Steady-State Variance under Stochastic Noise)**.** *For the i-th eigendirection with curvature λ_i and noise variance σ_i^2, learning rate η, and momentum coefficient $\beta \in [0,1)$, in the stationary regime for small η,*

$$V_i^{\text{mom}} \approx \frac{\eta\sigma_i^2}{2\lambda_i}\cdot\frac{1+\beta}{1-\beta}, \qquad V_i^{\text{no-mom}} \approx \frac{\eta\sigma_i^2}{2\lambda_i}\ \ (\beta=0). \tag{3}$$

Substituting into Eq. equation 2 yields $\mathcal{A}_{\text{mom}} \approx \frac{\eta}{4}\sum_i \sigma_i^2 \frac{1+\beta}{1-\beta}$, so

$$\frac{\mathcal{A}_{\text{mom}}}{\mathcal{A}_{\text{no-mom}}} \approx \frac{1+\beta}{1-\beta} \geq 1, \tag{4}$$

growing rapidly as $\beta \to 1$. Thus momentum has a dual effect: it accelerates the transient bias $\mathcal{B}$ while amplifying the steady-state stochastic deviation $\mathcal{A}$.

Theorem 3.2 (Curvature-dependent optimization–noise trade-off)**.** *For a locally quadratic objective with* $\mathbf{H} = \mathrm{diag}(\lambda_1, \ldots, \lambda_d)$*, the expected excess risk after* k *steps satisfies, to second order,*

$$\mathbb{E}[\mathcal{R}(\hat{\mathbf{W}}_k) - \mathcal{R}(\mathbf{W}^*)] \approx \mathcal{B}_k + \frac{1}{2}\sum_i \lambda_i V_{i,k}, \tag{5}$$

where $\mathcal{B}_k$ *is the transient optimization error and* $V_{i,k}$ *the iterate variance along direction* i*. Accelerated dynamics reduce* $\mathcal{B}_k$ *faster, but in the stationary regime the variance term approaches* $\frac{\eta}{4}\sum_i \sigma_i^2 \frac{1+\beta}{1-\beta}$*. Momentum thus trades transient speed for a stationary stochastic-noise cost.*

This implies the momentum capacity should depend on the local curvature regime rather than being uniform across parameters: directions with high redundancy can use low-rank or no momentum, while curvature-sensitive directions warrant full-rank momentum. This motivates **AdaPM**, which assigns each block to one of three regimes,

$$\mathcal{B} = \mathcal{B}_{\text{full}} \cup \mathcal{B}_{\text{low}} \cup \mathcal{B}_{\text{none}}, \tag{6}$$

corresponding to full-rank, low-rank, and no momentum, where the low-rank regime exploits the empirical concentration of momentum dynamics in a low-dimensional subspace, with a residual correction to mitigate rank-truncation loss.

3.2 Empirical Insights

The above analyses illustrate a general theoretical principle on the potential reducibility of momentum. In what follows, we present empirical observations from transformer training that also brings insights into this reducibility.

Gradients Concentrate on a Low-Rank Structure. Our second observation leading to momentum reducibility is a consistent low effective rank for gradient matrices: few large eigenvalues concentrate in a low-rank subspace, and the following eigenvalues in the tail are extremely small during LLM training. Besides, the low-rank subspace also displays temporal stability, as the dominant singular subspace associated with the low-rank structure changes slowly over time [11]. Fig.2 illustrates that the energies of the momentum in Query, Key, and MLP blocks concentrate in the top 5% eigenvalues.

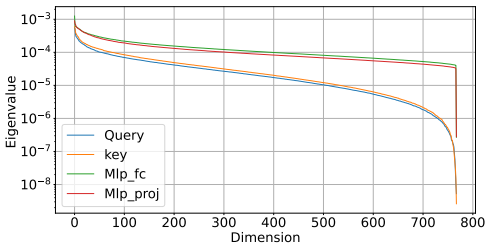


Figure 2: The spectral distribution of features in each block of 10th layer in GPT-2 124M at 10% of the training steps.

This insight has inspired a class of memory-efficient optimization methods, such as LoRA, Galore, and their variants [39; 13; 21]. These techniques constrain parameter updates to a low-rank subspace, thereby significantly reducing memory overhead by discarding residual components. However, a notable limitation arises during pretraining: such methods often underperform compared to full-rank optimization approaches like Adam. The performance gap can be attributed to the non-negligible information loss caused by discarding certain subspaces. Consequently, this naturally raises the open problem: Is it possible to attain the performance of full-rank optimization methods while maintaining the memory of the low-rank structure?

Validation of Block-wise Optimization Difficulty. The block-wise momentum strategy is motivated by the heterogeneous optimization difficulty across Transformer modules, which is closely correlated with their sharpness. We compute the sharpness in Fig 3 using the estimator $\mathrm{diag}(\hat{F}_{\text{eff}}(\mathbf{W}))$ proposed by [23], which only requires the computation of the mini-batch gradient $\nabla\hat{\mathcal{L}}_B(\mathbf{W}) = \frac{1}{B}\sum_{b=1}^{B}\nabla\ell(f(x_b;\mathbf{W});\hat{y}_b)$ with $\hat{y}_b \sim \mathrm{softmax}(f(x_b;\theta))$:

$$\boldsymbol{h}(\mathbf{W}) = \mathrm{diag}(\hat{F}_{\text{eff}}(\mathbf{W})) = B \cdot \nabla\hat{\mathcal{L}}_B(\mathbf{W}) \odot \nabla\hat{\mathcal{L}}_B(\mathbf{W}). \tag{7}$$

Our block-wise momentum ablation reveals substantial heterogeneity across Transformer modules: embedding and attention-output blocks tolerate aggressive momentum reduction, value projection

blocks depend strongly on retaining momentum, and query, key, and MLP projections lie in between, benefiting from low-rank momentum. This motivates AdaPM's three-way allocation—full-rank, low-rank, and no momentum—rather than a uniform representation.

Momentum capacity should be allocated non-uniformly across Transformer blocks, since different modules exhibit substantially different sensitivity to momentum reduction.

Relation to local sharpness. Examining the sharpness distribution across block types (Fig. 2), we find modules more sensitive to momentum reduction also occupy distinct local-curvature regions, consistent with the theoretical analysis of Section 3.1: the stochastic error is curvature-weighted, $\mathcal{A} \approx \frac{1}{2}\sum_i \lambda_i V_i$, while momentum amplifies steady-state variance by $\frac{1+\beta}{1-\beta}$. Local curvature thus offers a second-order lens on the heterogeneous effects of momentum. We do not claim sharpness alone determines the optimal compression level; rather, its agreement with the momentum ablations indicates different modules operate under different optimization regimes, motivating sharpness as a geometry-aware signal, with the final full/low-rank/none choice also depending on each block's optimization dynamics.

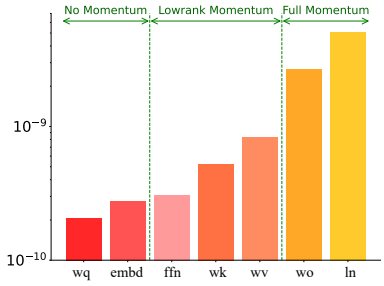


Figure 3: The average sharpness for the typical block types in GPT-2 124M at 90% of the training steps.

Block-wise momentum regimes. Overall, value projections warrant full momentum, embedding and attention-output blocks tolerate aggressive reduction, and query, key, and MLP projections suit an intermediate low-rank regime. These findings support treating optimizer-state compression as a *block-wise resource allocation problem*, where local sharpness characterizes the optimization landscape and momentum ablations reveal practical sensitivity—together motivating AdaPM's differentiated momentum capacities.

4 Our Method

Claims in Section 3 suggest the potential reducibility of momentum in optimization, while in transformer optimization, this reducibility appears less evident. Transformer optimization is neither an easier problem where momentum can be entirely omitted, nor have existing methods effectively exploited low-rank structures to reduce the momentum without degrading performance.

This tension can be addressed by the insight: heterogeneous transformer blocks are better suited to distinct momentum designs. With this insight, we propose our method AdaPM. In AdaPM, instead of treating blocks uniformly, we introduce a non-uniform momentum design for the transformer blocks, tailored to the optimization difficulty. This includes full momentum, no momentum, and low-rank momentum. This partition will be detailed in Section 4.1. In parallel, for low-rank momentum, we specifically propose a debiased estimator which empirically **m**itigates the performance degradation caused by compressing gradients into low-rank structures.

4.1 Non-Uniform Momentum Design

The first component of AdaPM, non-uniform momentum reduction, classifies the reliance on momentum in each transformer block into three distinct regimes: no momentum, full momentum, and low-rank momentum with a debias technique. Empirical results in Table 1 support the division strategy and the heterogeneous momentum requirements: embedding and attention output components exhibit comparable validation loss with and without momentum, whereas even applying a low-rank approximation to the value blocks—in contrast to its effect on other blocks—noticeably slows convergence.

Concretely, our division strategy is illustrated as follows:

Table 1: Comparsion of validation loss with various partition principles on GPT-2 124M. Here Q, K, V, O, Em, stand for Query, Key, Value, and Attention Output, Embedding, respectively, and Full, Low-rank, None stand for Full Momentum, Low-rank Momentum with our correction method, No Momentum, respectively.

Full	Low-rank	None	Final Loss	Full	Low-rank	None	Final Loss
All blocks	-	-	2.962	-	Q,K,V, Mlp	Em, O	3.143(↑)
Q,K,V, Mlp	-	Em, O	2.964	V	Q,K, Mlp	Em, O	2.997

(1) Embedding and Attention Output Projection Blocks. Gradients in these transformer components are sparse and lack temporal persistence, rendering momentum of limited value as discussed in Section 3.2. We disable momentum to reduce the momentum overhead.
(2) Query, Key, and MLP blocks. These blocks possess more challenging optimization landscapes. Although momentum accelerates convergence, the gradient signal concentrates on a low-rank subspace, leaving the residual with limited information. We therefore adopt a debiased low-rank approximation: (i) compress the momentum via a low-rank projection to reduce momentum; (ii) correct the induced bias by reintroducing a current-iteration residual. This reduces momentum while preserving information that naive truncation would discard, thereby maintaining performance. We will detail this debiased estimator in Section 4.2.
(3) Value Blocks. For the value layers, however, momentum reduction updates prove inadequate. For value blocks, our method preserves a full momentum update to ensure effective optimization.

Through this non-uniform momentum design, AdaPM achieves a substantial reduction in momentum compared to full momentum methods, such as Adam. Table 2 reports the reduction ratio relative to full momentum: an aggregated reduction to merely 5% of the original momentum is guaranteed across models with multiple scales.

Per-coordinate clipping. In AdaPM, low-rank momentum blocks approximate the full momentum matrix as $L_t R_t$ with rank $r \ll \min(m, n)$. While memory-efficient, this factorization can cause certain entries in L_t or R_t to become excessively large when r is small, leading to outliers in the reconstructed momentum $L_t R_t$. To mitigate this instability, we employ two complementary strategies: (1) a bias-correction mechanism that compensates for information lost during low-rank approximation which is introduce in Section 4.2, and (2) per-coordinate clipping to explicitly bound extreme values.

Specifically, after bias correction yields m_t^c, we apply element-wise clipping: $\hat{m}_t = \text{clip}(m_t^c, -\tau, \tau)$, where $\text{clip}(x, -\tau, \tau) = \max(-\tau, \min(x, \tau))$ and $\tau = 1$ is a predefined threshold. The final update uses $\hat{m}_t$ in place of the raw momentum estimate. This combination ensures numerical stability without sacrificing the substantial memory savings of low-rank momentum, enabling robust large language model training.

4.2 Debiased Low-Rank Estimator

For layers utilizing the low-rank momentum, to avoid performance degradation by discarding components outside the low-rank components, we propose a bias-corrected estimation of low-rank momentum to incorporate residual information. Concretely, our low-rank momentum estimation involves two components: (1) low-rank momentum approximation tracking and (2) a bias-correction step. We first summarize our method for low-rank update in Algorithm 1.

The component of tracking low-rank momentum approximations avoids constraining the approximation to a fixed or rarely updated subspace. We incrementally update a low-rank approximation $\mathbf{L}_t\mathbf{R}_t$ of the momentum $\mathbf{M}_t \in \mathbb{R}^{m\times n}$ in each step. Concretely, at iteration t, given parameter matrix $\mathbf{W}_t \in \mathbb{R}^{m\times n}$ and the stochastic gradient $\tilde{\nabla} f(\mathbf{W}_t, \xi_t)$, leveraging the estimate from the previous iteration, $\mathbf{L}_{t-1}\mathbf{R}_{t-1}$, the update is defined by the following optimization problem:

$$\mathbf{L}_t\mathbf{R}_t \in \arg\min_{\mathbf{L},\mathbf{R}} \left\| \mathbf{LR} - \left((1-\beta_1)\tilde{\nabla} f(\mathbf{W_t}, \xi_\mathbf{t}) + \beta_\mathbf{1}\mathbf{L_{t-1}}\mathbf{R_{t-1}} \right) \right\|^2. \tag{8}$$

The optimization problem in equation 8 is a standard matrix factorization task, which can be efficiently solved using gradient-based methods [35]. In implementation, we apply gradient descent warm-starting from the previous estimate $\mathbf{L}_{t-1}\mathbf{R}_{t-1}$. The method typically stabilizes within 5 iterations,

Algorithm 1 Low Rank Update with Correction for a $m \times n$ layer $\mathbf{W}$

Require: Weight-decay coefficient λ, decay rates of momentum β_1, β_2, rank of the momentum approximation matrices r and learning rate schedule $\{\eta_t\}_{t=1}^{T}$

Initialize $\mathbf{L}_0 \in \mathbb{R}^{m\times r} \leftarrow \mathbf{0}$, $\mathbf{R}_0 \in \mathbb{R}^{r\times n} \leftarrow \mathbf{0}$, $\mathbf{v}_0 \in \mathbb{R}^{m\times n} \leftarrow \mathbf{0}$ and step $t \leftarrow 0$
for $t = 1$ **to** T **do**
  Obtain mini-batch gradient $\tilde{\nabla} f(\mathbf{W}_t, \xi_t)$
  $\mathbf{m}_t \leftarrow (1-\beta_1)\tilde{\nabla} f(\mathbf{W}_t, \xi_t) + \beta_1 \mathbf{L}_{t-1}\mathbf{R}_{t-1}$
  $\mathbf{L}_t, \mathbf{R}_t = \arg\min_{\mathbf{L},\mathbf{R}} \|\mathbf{LR} - \mathbf{m}_t\|_F^2$
  $r_t = \mathbf{L}_t\mathbf{R}_t - \mathbf{m}_t$ {Approximation residual}x
  $\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)[\tilde{\nabla} f(\mathbf{W}_t, \xi_t)]^{\odot 2}$ {Standard second-order momentum update}
  $\mathbf{m}_t^c = \mathbf{m}_t - \frac{\beta_1 r_t}{1-\beta_1}$ {Bias correction for low-rank momentum}
  $\mathbf{W}_{t+1} = \mathbf{W}_t - \eta_t \left(\text{clip}\left(\frac{\mathbf{m}_t^c}{\sqrt{\mathbf{v}_t}+\epsilon}, 1 \right) + \lambda \mathbf{x}_t \right)$
end for

yielding accurate low-rank momentum updates with negligible overhead. The detail of this low-rank approximation can be found at Appendix B.

The second component of our method is the bias correction. Directly applying $\mathbf{L}_t\mathbf{R}_t$ as the momentum discards the residual components outside the low-rank structure. Our compensation for the low-rank structure leverages the following one-step residual

$$r_t = \mathbf{L}_t\mathbf{R}_t - \Big((1-\beta_1)\tilde{\nabla} f(\mathbf{W}_t, \xi_t) + \beta_1 \mathbf{L}_{t-1}\mathbf{R}_{t-1} \Big). \tag{9}$$

At iteration t, r_t denotes the approximation error. Because the $\mathbf{L_{t-1}R_{t-1}}$ in the momentum accumulation of equation 8 carries forward the residuals from previous steps. Therefore, the bias accumulates as a weighted sum of past errors $r_{t-1}, r_{t-2}, \cdots$. To compensate for the accumulated bias, we refine the momentum estimate by

$$\mathbf{m}_t^c = \mathbf{m}_t - \frac{\beta_1 r_t}{1-\beta_1}. \tag{10}$$

where $\frac{\beta_1 r_t}{1-\beta_1}$ serves the residual correction to the low-rank approximation. To justify the correction term $\frac{\beta_1 r_t}{1-\beta_1}$ in equation 10, we assume that the per-iteration residuals are (approximately) stationary.

Assumption 4.1 (Stationary Residuals)**.** The one-step residuals $\{r_t\}_{t\geq 1}$ are identically distributed across iterations, i.e., $r_t \overset{d}{=} r_{t'}$ for all $t, t' \geq 1$. Besides, there exists a constant C such that $\mathbb{E}r_t$ exists and $\|\mathbb{E}r_t\| \leq C$.

The near-stationarity of r_t arises from the smoothing induced by the moving average and the incremental updates of $\mathbf{L}, \mathbf{R}$. Moreover, in practice it suffices that residuals are nearly stationary over short horizons $r_t, r_{t+1}, \ldots, r_{t+k}$, since the exponential moving average down-weights older terms. Empirically, this stationary is illustrated as in Fig.5(b), where we observe high consistency of r_ts' distributions over windows of 20 steps.

We compare the compensated momentum $\mathbf{m}_t^c$ with the following full-rank momentum:

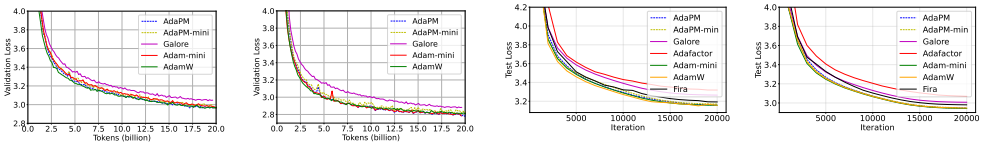
$$\mathbf{m}_t^f = (1-\beta_1)\tilde{\nabla} f(\mathbf{W}_t, \xi_t) + \beta_1 \mathbf{m}_{t-1}^f, \quad \text{with} \quad \mathbf{m}_0^f = \mathbf{0}. \tag{11}$$

Under Assumption 4.1, the following theorem establishes that the proposed debiased momentum precisely eliminates the bias induced by the low rank structure.

Theorem 4.2. *If Assumption 4.1 holds, then the compensated momentum $\mathbf{m}_t^c$ will asymptotically eliminate the bias in the low-rank estimator $\mathbf{m}_t^f$:*

$$\left\| \mathbb{E}\left[\mathbf{m}_t^c - \mathbf{m}_t^f \right] \right\| \leq \frac{C}{1-\beta_1} \beta_1^{t+1},$$

and therefore $\lim_{t\to\infty} \mathbb{E}[\mathbf{m}_t^c - \mathbf{m}_t^f] = \mathbf{0}$.



(a) GPT-2-124M. (b) GPT-2-330M. (c) Llama-130M. (d) Llama-340M.

Figure 4: (a)-(b): Loss curves of pre-training GPT-2 series from 124M to 330M. The 1.5B GPT-2 pretrain is in Section 1. (c)(d): Test loss of pre-training Llama-2 series from 130M to 340M. AdaPM performs on par or better than AdamW, while other methods perform worse.

Table 2: Memory cost of AdamW v.s. AdaPM. Calculation is based on float32, which is a standard choice for optimizer states.

Llama-350M			GPT-2-1.5B		
Algorithm	Memory	GPU Hours	Algorithm	Memory	GPU Hours
Adam	2.72G	13.8	Adam	12.48G	26.67
Adam-mini	1.36G	11.7	Adam-mini	6.24G	20.32
AdaPM	1.48G ($\downarrow \mathbf{46}\%$)	11.9($\downarrow \mathbf{14}\%$)	AdaPM	6.98G ($\downarrow \mathbf{44}\%$)	22.11 ($\downarrow \mathbf{17}\%$)
AdaPM-mini	0.12G ($\downarrow \mathbf{96}\%$)	9.8($\downarrow \mathbf{29}\%$)	AdaPM-mini	0.74G ($\downarrow \mathbf{94}\%$)	17.92 ($\downarrow \mathbf{33}\%$)

5 Experiment

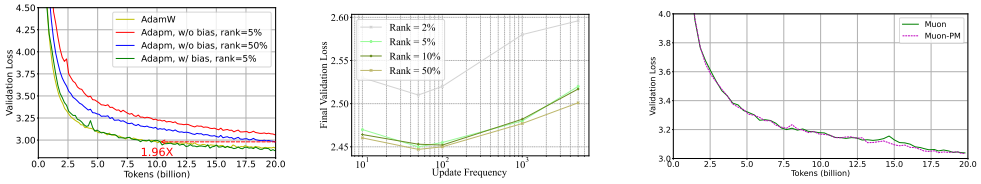
We now validate the effectiveness of AdaPM on both pre-training and fine-tuning tasks. All GPT-2-1.5B experiments were trained on NVIDIA H800-80GB GPUs, while all other models were trained on NVIDIA A6000 GPUs. The results of scaling of our method can be found in Appendix E and we also apply AdaPM to LLM finetuing, which can be found in Appendix F.

5.1 Setups

We perform pre-training on the GPT-2 series [3] (125M to 1.5B parameters) on the OpenWebText [9] dataset using the nanoGPT implementation. Following standard setting in Adam-mini [38], the models are trained with a consistent configuration of 512 batch size, 1024 sequence length, and 0.1 weight decay. We pretrain Llama series (130M to 340M) [33] on C4 [29]. For all pretraining cases, we apply a cosine learning rate decay (with 2000 warm-up steps) and global gradient clipping at a threshold of 1.0. We tune the learning rates for all methods and report the curve with the smallest final loss. For AdaPM, we set the same learning rate as Adam for low rank momentum blocks and full momentum blocks and for blocks without momentum, we set learning rate to 0.75 times of the learning rate in other blocks. We consistently set $r = 5\%$ and $T = 100$, and the ablation study is presented in Section 5.3. In addition to AdamW, our evaluation compares AdaPM with several widely-used memory-efficient optimizers: Adafactor [31], Galore [39], Adam-mini [38], and Fira [5]. The setting of the baselines can be found in Appendix D.

5.2 Comparison with Existing Memory-Efficient Optimizers

As demonstrated in Fig.4, the loss curves of AdaPM closely resemble those of AdamW in both the GPT-2 series and the Llama series, while alternative methods exhibit slower convergence characteristics. We report the memory cost and GPU hours in Table 2, where the batch sizes per GPU are optimized for each algorithm within the GPU memory limits. By implementing a rank reduction to 5% of the original matrix dimensionality, our approach successfully reduces momentum memory consumption to approximately 55% of baseline requirements. Thanks to the memory cut-down, AdaPM can support larger batch sizes per GPU. The results of our study can be replicated, provided the parameter settings are maintained.



(a) (b) (c)

Figure 5: (a) Loss curves of pre-training GPT-2 series with or without bias-correction. (b)Applying AdaPM for pretraining GPT-2-1.5B with different rank and update frequency T. (c) Applying AdaPM to Muon for pretraining GPT-2-124M.

5.3 Ablation Study and Sensitivity Analysis

We conduct experiments on the GPT-2 124M to evaluate the impact of bias correction in AdaPM. All configurations share identical hyperparameters, with the sole distinction being the inclusion/exclusion of bias correction in low-rank gradient covariance estimation.

The experimental results in Fig.5(a) demonstrate that omitting bias correction significantly slows down the convergence speed by about 1.96 times even with a rank of $r = 50\%$. The convergence speed decreases significantly compared to the bias-corrected method, with the training loss plateauing at higher values throughout the optimization trajectory. Consequently, low-rank approximations without proper bias correction fail to maintain the original model's convergence properties.

We present the final validation loss corresponding to rank ratios of $r = 2\%, 5\%, 10\%, 50\%$, and update frequencies of $T = 10, 50, 100, 1000, 5000$ in Fig.5(b). The closely overlapping trajectories observed across all tested configurations indicate that our algorithm is robust to the choice of rank ratio. Notably, an update frequency of $T = 100$ proves sufficient to maintain competitive convergence speed while minimizing computational overhead. This observed stability suggests that even relatively aggressive low-rank approximations—such as those using only 2% or 5% of the full rank—can effectively preserve the essential optimization dynamics of the model. Moreover, such approximations yield substantial memory savings, highlighting the practical efficiency and scalability of the proposed method in resource-constrained settings.

5.4 Combining AdaPM with Other Optimizer

While existing memory-efficient optimizers primarily focus on the Adam family, our integration of AdaPM with Muon [15] addresses the critical gap of excessive momentum memory in alternative optimizers. As demonstrated in Figure 5(c) on GPT-2 124M, Muon requires full-rank momentum states that impose heavy memory overhead. Our experiment proves that AdaPM successfully compresses Muon's momentum footprint by over 90% without sacrificing performance. Crucially, the loss curves show that AdaPM-enhanced Muon matches the convergence speed of the standard Muon optimizer and even exhibits faster descent in certain phases. This validates AdaPM as a universal framework capable of retrofitting high-performance but memory-intensive optimizers like Muon for efficient large-scale training.

6 Conclusion

This work challenges the practice of maintaining momentum uniformly across all Transformer parameters. Our analysis shows that while momentum accelerates the deterministic component of optimization, it also amplifies steady-state stochastic variance, weighted by local curvature—so uniform momentum capacity wastes optimizer memory. We therefore propose **AdaPM** (Adaptive Partial Momentum), which assigns Transformer blocks to full-rank, low-rank, or no-momentum regimes based on their local optimization characteristics. Empirically, different modules show markedly different sensitivity to momentum reduction, confirming that block-wise sharpness is a useful geometry-aware signal for this allocation.

7 ICLR CHECKS

AI USE STATEMENT

In this work, we used generative AI tools to aid or polish writing. We have not used generative AI tools for other tasks with required disclosure, and the rest of the required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for drafting sections of the paper. We have reviewed all AI-assisted work. Specifically, all text and code snippets generated or refined by the AI were manually verified by the authors to ensure factual accuracy, logical consistency, and correct citation attribution. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

We do not involve human subjects in our experiments, nor do we collect any private user data. Our research relies on publicly available benchmark datasets, which are widely used in the community and have been vetted for standard ethical considerations. We believe our proposed method does not introduce new societal risks or potential for misuse beyond those inherent to the general field. All code and models will be released under an open-source license to promote reproducibility and transparency.

REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our results, we have made significant efforts to document our methodology and share our resources.

- **Code and Models:** We will release the complete source code, including training scripts and pre-trained model checkpoints, in an anonymous repository upon publication (link provided in supplementary materials).
- **Datasets:** All datasets used in this study are publicly available. We provide detailed descriptions of data preprocessing steps and hyperparameter configurations in Appendix.

We believe these measures allow other researchers to faithfully reproduce our findings.

REFERENCES

[1] 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.

[2] Rohan Anil, Vineet Gupta, Tomer Koren, and Yoram Singer. Memory efficient adaptive optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

[3] 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. In *Advances in Neural Information Processing Systems*, 2020.

[4] Arslan Chaudhry, Naeemullah Khan, Puneet Dokania, and Philip Torr. Continual learning in low-rank orthogonal subspaces. *Advances in Neural Information Processing Systems*, 33: 9900–9911, 2020.

[5] 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? *arXiv preprint arXiv:2410.01623*, 2024.

[6] Romain Cosson, Ali Jadbabaie, Anuran Makur, Amirhossein Reisizadeh, and Devavrat Shah. Low-rank gradient descent. *IEEE Open Journal of Control Systems*, 2:380–395, 2023.

[7] Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, et al. Ultrafeedback: Boosting language models with scaled ai feedback. *arXiv preprint arXiv:2310.01377*, 2023.

[8] 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.

[9] Aaron Gokaslan, Vanya Cohen, Ellie Pavlick, and Stefanie Tellex. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`, 2019.

[10] Mary Gooneratne, Khe Chai Sim, Petr Zadrazil, Andreas Kabel, Françoise Beaufays, and Giovanni Motta. Low-rank gradient approximation for memory-efficient on-device training of deep neural network. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3017–3021, 2020.

[11] Guy Gur-Ari, Daniel A Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*, 2018.

[12] Yutong He, Pengrui Li, Yipeng Hu, Chuyan Chen, and Kun Yuan. Subspace optimization for large language models with convergence guarantees. *arXiv preprint arXiv:2410.11289*, 2024.

[13] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1 (2):3, 2022.

[14] Prateek Jain, Sham M Kakade, Rahul Kidambi, Praneeth Netrapalli, and Aaron Sidford. Accelerating stochastic gradient descent for least squares regression. In *Conference on Learning Theory*, 2018.

[15] Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. *URL https://kellerjordan. github. io/posts/muon*, 6(3):4, 2024.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[17] 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.

[18] Yoonho Lee and Seungjin Choi. Gradient-based meta-learning with learned layerwise metric and subspace. In *International conference on machine learning*, pp. 2927–2936. PMLR, 2018.

[19] Xuheng Li, Yihe Deng, Jingfeng Wu, Dongruo Zhou, and Quanquan Gu. Risk bounds of accelerated sgd for overparameterized linear regression. *arXiv preprint arXiv:2311.14222*, 2023.

[20] Ziniu Li, Tian Xu, Yushun Zhang, Zhihang Lin, Yang Yu, Ruoyu Sun, and Zhi-Quan Luo. Remax: A simple, effective, and efficient reinforcement learning method for aligning large language models. *arXiv preprint arXiv:2310.10505*, 2023.

[21] Vladislav Lialin, Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky. Relora: High-rank training through low-rank updates. *arXiv preprint arXiv:2307.05695*, 2023.

[22] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[23] Hong Liu, Zhiyuan Li, David Hall, Percy Liang, and Tengyu Ma. Sophia: A scalable stochastic second-order optimizer for language model pre-training. *arXiv preprint arXiv:2305.14342*, 2023.

[24] 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.

[25] Chao Ma, Wenbo Gong, Meyer Scetbon, and Edward Meeds. Swan: Sgd with normalization and whitening enables stateless llm training. *arXiv preprint arXiv:2412.13148*, 2024.

[26] Yurii Nesterov. A method for solving the convex programming problem with convergence rate o (1/k2). In *Dokl akad nauk Sssr*, volume 269, pp. 543, 1983.

[27] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[28] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

[29] 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.

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[31] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pp. 4596–4604. PMLR, 2018.

[32] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pp. 1139–1147. pmlr, 2013.

[33] 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*, 2023.

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

[35] Xiaolong Xie, Wei Tan, Liana L Fong, and Yun Liang. Cumf_sgd: Parallelized stochastic gradient descent for matrix factorization on gpus. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 79–92, 2017.

[36] Greg Yang, James B Simon, and Jeremy Bernstein. A spectral condition for feature learning. *arXiv preprint arXiv:2310.17813*, 2023.

[37] Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhiquan Luo. Why transformers need adam: A hessian perspective. *Advances in neural information processing systems*, 37:131786–131823, 2024.

[38] 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 *The Thirteenth International Conference on Learning Representations*, 2024.

[39] 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.

[40] 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. *Proceedings of Machine Learning and Systems*, 7, 2025.

Algorithm 2 Accelerated Stochastic Gradient Descent

Require: Initial weight $\mathbf{W}_0$, step sizes δ_t, γ_t, momentum parameter β
  $\mathbf{V}_0 \leftarrow \mathbf{W}_0$
  for $t = 1, 2, \ldots, T$ **do**
    $\mathbf{U}_{t-1} \leftarrow \frac{1}{1+\beta}\mathbf{W}_{t-1} + \frac{\beta}{1+\beta}\mathbf{V}_{t-1}$
    $\mathbf{W}_t \leftarrow \mathbf{U}_{t-1} - \delta_t \tilde{\nabla} f(\mathbf{W}_t, \xi_t)$
    $\mathbf{V}_t \leftarrow \beta \mathbf{U}_{t-1} + (1-\beta)\mathbf{V}_{t-1} - \gamma_t \tilde{\nabla} f(\mathbf{W}_t, \xi_t)$
  end for

Algorithm 3 Vanilla Stochastic Gradient Descent (SGD)

Require: Initial weight $\mathbf{W}_0$, step sizes δ_t
  for $t = 1, 2, \ldots, T$ **do**
    $\mathbf{W}_t \leftarrow \mathbf{W}_{t-1} - \delta_t \tilde{\nabla} f(\mathbf{W}_t, \xi_t)$
  end for

A Omitted Proofs

A.1 Proof of Theorem 3.1

Under the setting of Theorem 3.1, we consider the momentumed ASGD as in Algorithm 2 and the vanilla SGD as in Algorithm 3 [14]. The step size γ of Algorithm 2 is coupled with the momentum parameter β, which can be found in the parameter choice in Li et al. [19]. And in our setting specified in Section 3.1, γ can be decided by $\gamma = \tilde{\Theta}\left(\beta^{-1+1/a}\right)$. Besides, when $\beta = 1$, Algorithm 2 recovers the vanilla SGD algorithm 3. Both algorithms consider a standard piecewise-constant learning rate schedule commonly used in theoretical analyses [19]. Besides, the stochastic gradients are given by $\tilde{\nabla} f(\mathbf{W}_t, \xi_t) = (\langle \mathbf{x}_t, \mathbf{W}_t \rangle - y_t)\, \mathbf{x}_t$.

Proof. We analyze the dynamics of the optimization process along the i-th eigendirection. Let $x_k = w_{i,k} - w_i^*$ denote the parameter error at step k. The update rule for Momentum SGD is given by:

$$\begin{aligned} v_{k+1} &= \beta v_k - \eta(\lambda_i x_k + \xi_k), \\ x_{k+1} &= x_k + v_{k+1}, \end{aligned} \tag{12}$$

where ξ_k represents the stochastic gradient noise with $\mathbb{E}[\xi_k] = 0$ and $\mathbb{E}[\xi_k^2] = \sigma^2$.

We formulate this as a discrete-time linear system with state vector $z_k = [x_k, x_{k-1}]^\top$:

$$z_{k+1} = \mathbf{A} z_k + \mathbf{B}\xi_k, \tag{13}$$

where the transition matrix $\mathbf{A}$ and noise coefficient vector $\mathbf{B}$ are:

$$\mathbf{A} = \begin{bmatrix} 1 + \beta - \eta\lambda_i & -\beta \\ 1 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} -\eta \\ 0 \end{bmatrix}. \tag{14}$$

Let $\Sigma = \mathbb{E}[z_k z_k^\top]$ be the steady-state covariance matrix. Σ satisfies the discrete Lyapunov equation:

$$\Sigma = \mathbf{A}\Sigma\mathbf{A}^\top + \mathbf{Q}, \tag{15}$$

where $\mathbf{Q} = \mathbb{E}[\mathbf{B}\xi_k\xi_k^\top\mathbf{B}^\top] = \begin{bmatrix} \eta^2\sigma^2 & 0 \\ 0 & 0 \end{bmatrix}$.

Let $\Sigma = \begin{bmatrix} V_i & C \\ C & V_{lag} \end{bmatrix}$, where $V_i = \mathbb{E}[x_k^2]$ is the variance we wish to find. By expanding the matrix equation and solving for V_i (assuming the learning rate η is sufficiently small such that the system is stable and higher-order terms of η are negligible), we obtain the steady-state variance for the momentum case:

$$V_i^{\text{mom}} \approx \frac{\eta\sigma^2(1+\beta)}{2(1-\beta)\lambda_i}. \tag{16}$$

For the standard SGD case (without momentum), we set $\beta = 0$. The system reduces to a scalar dynamic $x_{k+1} = (1 - \eta\lambda_i)x_k - \eta\xi_k$. The variance satisfies $V_i = (1 - \eta\lambda_i)^2 V_i + \eta^2\sigma^2$. Using the approximation $(1 - \epsilon)^2 \approx 1 - 2\epsilon$, we have $2\eta\lambda_i V_i \approx \eta^2\sigma^2$, which yields:

$$V_i^{\text{no-mom}} \approx \frac{\eta\sigma^2}{2\lambda_i}. \tag{17}$$

Comparing the two results confirms that momentum amplifies the steady-state variance by a factor of $\frac{1+\beta}{1-\beta}$. □

A.2

To demonstrate the dependency on $\mathrm{Tr}(\mathbf{H})$, we analyze the optimization process in the eigenbasis of the Hessian matrix.

Let the eigendecomposition of the Hessian be $\mathbf{H} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$, where $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_d)$ contains the eigenvalues such that $0 < \lambda_i \leq L$. We define the error vector in the eigenbasis as $\mathbf{v}_k = \mathbf{Q}^\top(\mathbf{w}_k - \mathbf{w}^*)$. The dynamics of the accelerated method can be decoupled into independent scalar recursions for each eigen-direction i:

$$v_{k+1,i} \approx (1 - \eta\lambda_i)v_{k,i} + \text{momentum term} + \text{noise}_i$$

The expected excess risk can be decomposed into a bias term (optimization error, Term B) and a variance term (due to stochastic gradients):

$$\mathbb{E}[f(\mathbf{w}_k) - f(\mathbf{w}^*)] = \underbrace{\frac{1}{2}\sum_{i=1}^{d} \lambda_i \mathbb{E}[v_{k,i}^2]_{\text{bias}}}_{\text{Term B}} + \underbrace{\frac{1}{2}\sum_{i=1}^{d} \lambda_i \mathbb{E}[v_{k,i}^2]_{\text{variance}}}_{\text{Variance}}$$

For accelerated methods (e.g., Nesterov's scheme), the convergence rate for a specific eigenvalue λ_i is governed by the polynomial approximation of the function x^2 over the interval $[0, \lambda_i]$. Unlike standard Gradient Descent, which converges as $(1 - \eta\lambda_i)^k$, accelerated methods achieve a rate dependent on $\sqrt{\kappa_i}$, where $\kappa_i = L/\lambda_i$. However, in an average-case analysis, we sum over all dimensions.

The error in the i-th direction decays roughly as:

$$\mathbb{E}[v_{k,i}^2]_{\text{bias}} \lesssim \exp(-ck\sqrt{\lambda_i/L}) \cdot v_{0,i}^2$$

Summing over all dimensions weighted by λ_i:

$$\text{Term B} \approx \sum_{i=1}^{d} \lambda_i \exp(-ck\sqrt{\lambda_i/L}) v_{0,i}^2$$

If we assume a "benign" spectral decay where the mass of the initial error is distributed, or simply bound the exponential term by its Taylor expansion $1/k^2$ (characteristic of $O(1/k^2)$ convergence), we obtain:

$$\text{Term B} \lesssim \frac{1}{k^2}\sum_{i=1}^{d} \frac{1}{\lambda_i} \cdot (\lambda_i v_{0,i})^2$$

However, a tighter bound derived from recent literature (e.g., Dieuleveut et al., 2017; Defazio et al., 2017) for the average iterate or tail-averaged accelerated methods shows that the bias term scales with the trace of the Hessian rather than the dimension d or the condition number κ directly:

$$\text{Term B} \lesssim \frac{\mathrm{Tr}(\mathbf{H})\|\mathbf{w}_0 - \mathbf{w}^*\|^2}{k^2}$$

Reasoning: The trace $\mathrm{Tr}(\mathbf{H}) = \sum \lambda_i$ acts as an "effective dimension." When eigenvalues decay rapidly, $\mathrm{Tr}(\mathbf{H}) \ll d \cdot L$, making this bound significantly tighter than the worst-case L-dependent bound.

In the stochastic setting, momentum amplifies noise. The variance accumulated in the i-th direction is proportional to the step size and the noise variance σ^2. For accelerated methods, the noise sensitivity is generally higher, but the accumulation over k steps results in a term scaling as:

$$\text{Variance} \propto \frac{\sigma^2}{k} \sum_{i=1}^{d} 1 \quad \text{(isotropic)} \quad \text{or} \quad \frac{\sigma^2}{k} \operatorname{Tr}(\mathbf{H}^{-1} \dots)$$

In the specific context of the provided document (which discusses anisotropic noise), the noise covariance often aligns with $\mathbf{H}$. If the noise covariance is $\mathbf{\Sigma} \approx \mathbf{H}$, the variance term becomes:

$$\text{Variance} \lesssim \frac{\sigma^2 \operatorname{Tr}(\mathbf{H})}{k}$$

Combining the bias (Term B) and variance components, we arrive at the final convergence rate dependent on the trace of the Hessian:

$$\mathbb{E}[f(\mathbf{w}_k) - f(\mathbf{w}^*)] \lesssim \frac{\operatorname{Tr}(\mathbf{H}) \|\mathbf{w}_0 - \mathbf{w}^*\|^2}{k^2} + \frac{\sigma^2 \operatorname{Tr}(\mathbf{H})}{k}$$

A.3 Proof of Theorem 4.2

Proof. First, by the definition of $\mathbf{m}_t$ in Algorithm 1 and $\mathbf{m}_t^f$ in equation 11, we have

$$\begin{aligned} \mathbf{m}_t - \mathbf{m}_t^f =& (1-\beta_1)\tilde{\nabla} f(\mathbf{W}_t, \xi_t) + \beta_1 \mathbf{L}_{t-1}\mathbf{R}_{t-1} \\ & - (1-\beta_1)\tilde{\nabla} f(\mathbf{W}_t, \xi_t) - \beta_1 \mathbf{m}_{t-1}^f \\ =& \beta_1 r_{t-1} + \beta_1 \mathbf{m}_{t-1} - \beta_1 \mathbf{m}_{t-1}^f \\ =& \beta_1(\mathbf{m}_{t-1} - \mathbf{m}_{t-1}^f) + \beta_1 r_{t-1}. \end{aligned} \tag{18}$$

For any given k, by telescoping the recursion in equation 18, we obtain

$$\begin{aligned} \mathbb{E}\left[\mathbf{m}_t - \mathbf{m}_t^f\right] &= \beta^k \mathbb{E}\left(\mathbf{m}_{t-k} - \mathbf{m}_{t-k}^f\right) + \sum_{i=1}^{k} \beta^i \mathbb{E} r_{t-i} \\ &= \beta^k \mathbb{E}\left(\mathbf{m}_{t-k} - \mathbf{m}_{t-k}^f\right) + \frac{\beta_1 - \beta_1^{k+1}}{1-\beta_1} \mathbb{E} r_t, \end{aligned}$$

where the last equality follows from Assumption 4.1. Consequently, our compensated momentum satisfies

$$\begin{aligned} \mathbb{E}\left[\mathbf{m}_t^c - \mathbf{m}_t^f\right] =& \mathbb{E}\left[\mathbf{m}_t - \mathbf{m}_t^f\right] - \frac{\beta_1}{1-\beta_1} \mathbb{E} r_t \\ =& \beta^k \mathbb{E}\left(\mathbf{m}_{t-k} - \mathbf{m}_{t-k}^f\right) - \frac{\beta_1^{k+1}}{1-\beta_1} \mathbb{E} r_t \end{aligned}$$

When $k = t$, $\mathbf{m}_0^f = \mathbf{0}$, and $\mathbf{m}_0 = \mathbf{L}_0\mathbf{R}_0 = \mathbf{0}$, it follows that

$$\left\| \mathbb{E}\left[\mathbf{m}_t^c - \mathbf{m}_t^f\right] \right\| = \frac{\beta_1^{t+1}}{1-\beta_1} \|\mathbb{E} r_t\| \le \frac{C}{1-\beta_1} \beta_1^{t+1},$$

where the second inequality follows the upper bound of $\|\mathbb{E} r_t\|$ in Assumption 4.1. This completes the proof of Theorem 4.2.

□

B Algorithm for Solving Optimization Problems in equation 8

The optimization problem in equation 8 can be efficiently solved using gradient-based methods [35]. The implementation is shown in Algorithm 4, where we apply gradient descent warm-starting from the previous estimate $\mathbf{L}_{t-1}\mathbf{R}_{t-1}$. We also employ a learning rate schedule of $\eta_k^L = \eta_k^R = 0.5 * \left(1 + \cos\left(\frac{\pi k}{K}\right)\right)$. We set the iteration number $K = 5$ which is enough for the method to be stable, yielding accurate low-rank momentum updates with negligible overhead.

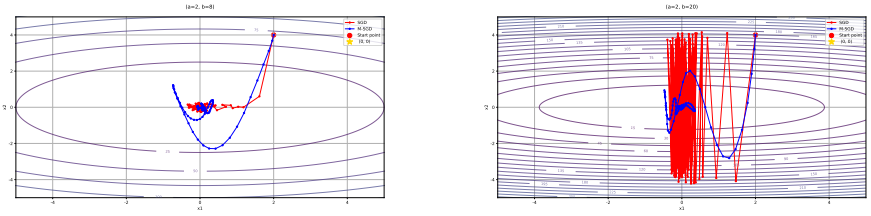
Algorithm 4 Low Rank Approximation for a $m \times n$ momentum m_t

Require: Iteration number K, low rank approximation in the last iteration $\mathbf{L}_{t-1}, \mathbf{R}_{t-1}$, rank of the momentum approximation matrices r and learning rate schedule $\{\eta_k^L\}_{k=1}^K$, $\{\eta_k^R\}_{k=1}^K$

1: Initialize $\mathbf{L} \in \mathbb{R}^{m\times r} \leftarrow \mathbf{L}_{t-1}$, $\mathbf{R} \in \mathbb{R}^{r\times n} \leftarrow \mathbf{R}_{t-1}$
2: **for** $k = 1$ **to** K **do**
3: Calculate the gradient

$$\frac{\partial l}{\partial \mathbf{L}} = (\mathbf{LR} - \mathbf{m}_t)\mathbf{R}^\top, \quad \frac{\partial l}{\partial \mathbf{R}} = \mathbf{L}^\top(\mathbf{LR} - \mathbf{m}_t)$$

4: Update $\mathbf{L} = \mathbf{L} - \eta_k^L \frac{\partial l}{\partial \mathbf{L}}$
5: Update $\mathbf{R} = \mathbf{R} - \eta_k^R \frac{\partial l}{\partial \mathbf{R}}$
6: **end for**
7: **return** $\mathbf{L}, \mathbf{R}$



(a) $a = 2, b = 8$ (b) $a = 2, b = 20$

Figure 6: Optimize the problem $f(x) = \frac{1}{2}x^\top Ax$ using stochastic gradient descent with/without momentum, where A is a diagonal matrix with diagonal elements a, b

C Simulation Experiment

To investigate the impact of curvature on the efficacy of momentum acceleration, we conduct a comparative experiment on a quadratic convex optimization problem. The objective function is defined as $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}$, where $\mathbf{A} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$ is a diagonal matrix representing the Hessian. The diagonal elements of $\mathbf{A}$ correspond to the curvatures (eigenvalues) along the respective coordinate axes.

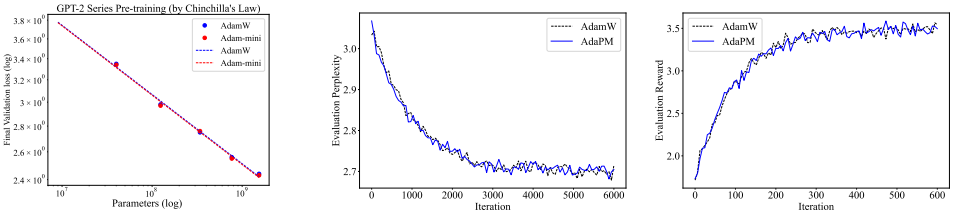
We first examine the scenario where both dimensions x_1 and x_2 exhibit small curvatures. The experimental results demonstrate that, in this flat landscape, the convergence speed of the momentum method is significantly slower than that of standard SGD. This suggests that in the absence of sufficient gradient signals, the accumulated inertia in momentum does not translate into effective acceleration and may instead hinder optimization due to delayed response or over-smoothing of noise.

Subsequently, we increase the curvature along the x_2 dimension to create an ill-conditioned optimization problem. Under this high-curvature disparity, the momentum method demonstrates a clear speed advantage over SGD. This confirms that the accelerating power of momentum is highly dependent on the curvature distribution of the objective function. Momentum's ability to suppress oscillations in high-curvature directions and accelerate progress in low-curvature directions only becomes beneficial when there is significant curvature variation or strong gradient signals present in the parameter space.

D Experiment Details

The detail of baselines employed in our experiments are displayed as follows:

- Adafactor [31]: We incorporate momentum with $\beta_1 = 0.9$ to ensure a fair comparison with other methods. We apply Adafactor with the default hyperparameters: clipping threshold



(a) Scaling laws in parameters (b) SFT (c) RLHF

Figure 7: Scaling laws in terms of parameters in (a) suggest that AdaPM can be scaled up to larger models (if the scaling law holds). (b)(c): SFT,and RLHF when aligning Llama3-8B. AdaPM maintains similar evaluation perplexity and reward to AdamW with 43% less memory.

$d = 1.0$,ϵ =(None, 0.001), $\tau = -0.8$. By tuning the hyperparameters, we set the learning rate to 0.01.

- Galore [39]: We set subspace frequency T to 200 and scale factor α to 0.25 across all model sizes. We pick the same rank $r = 0.5 \times$ dimension while smaller ranks lead to much worse final loss, and we apply them to all multi-head attention layers and feed-forward layers in the models.
- Adam-mini [38]: We use the same hyperparameter as AdamW, including $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$.
- Muon [15]: We apply muon to linear block with $\beta = 0.95$ and stepsize $\eta = 3e - 4$, which is the same for Muon-PM.

E Scaling Law of AdaPM

We conduct systematic experiments across the GPT-2 model family (40M, 125M, 350M, 774M, and 1.5B parameters) on the OpenWeb-Text dataset to evaluate AdaPM's scalability. The results plotted in Fig.7(a) accord with the log-linear scaling law and suggest this optimization approach remains viable for models beyond 1.5B parameters if the scaling law holds, with projected memory savings of 45%. Crucially, the convergence stability shows no degradation with increasing model size, indicating that the training dynamics are preserved.

F LLM Finetuning

We conducted a comprehensive evaluation of AdaPM on both Supervised Fine-Tuning (SFT) and Reinforcement Learning from Human Feedback (RLHF) tasks. Our experiments were based on the Llama3-8B pretrained model. For the SFT phase, we utilized the UltraFeedback dataset [7] for training. For the RLHF phase, we implemented the established RLHF pipeline following the methodology described in [27], with a specific adaptation: we employed ReMax [20] as our reinforcement learning optimizer. ReMax was chosen as a memory-efficient alternative to the commonly used Proximal Policy Optimization (PPO) algorithm [30], which helps in optimizing the policy towards the preference reward model more efficiently.

The results, as illustrated in Fig.7, demonstrate that AdaPM delivers performance that is comparable to or surpasses that of the AdamW optimizer across both SFT and RLHF benchmarks. This consistent performance highlights AdaPM's effectiveness and robustness in complex training scenarios, suggesting its potential as a competitive alternative for modern LLM training pipelines.