# AdaLomo: Low-memory Optimization with Adaptive Learning Rate

**Kai Lv[1,2*], Hang Yan[2], Qipeng Guo[2†], Haijun Lv[2], Xipeng Qiu[1]**
[1]School of Computer Science, Fudan University
[2]Shanghai AI Laboratory
klv21@m.fudan.edu.cn, {yanhang,guoqipeng,lvhaijun}@pjlab.org.cn
xpqiu@fudan.edu.cn

## Abstract

Large language models have achieved remarkable success, but their extensive parameter size necessitates substantial memory for training, thereby setting a high threshold. While the recently proposed low-memory optimization (LOMO) reduces memory footprint, its optimization technique, akin to stochastic gradient descent, is sensitive to hyper-parameters and exhibits suboptimal convergence, failing to match the performance of the prevailing optimizer for large language models, AdamW. Through analysis of the Adam optimizer, we found that, compared to momentum, the adaptive learning rate is more critical for bridging the gap. Building on this insight, we introduce the low-memory optimization with adaptive learning rate (AdaLomo), which offers an adaptive learning rate for each parameter and exhibits superior convergence performance compared to LOMO theoretically. To maintain memory efficiency, we employ non-negative matrix factorization for the second-order moment estimation. Additionally, we suggest the use of a grouped update normalization to stabilize convergence. Our experiments on instruction-tuning, further pre-training and from-scratch pre-training demonstrate that AdaLomo achieves results on par with AdamW, while significantly reducing memory requirements, thereby lowering the hardware barrier to training large language models. The code is accessible at https://github.com/OpenLMLab/LOMO.

## 1 Introduction

Large language models (Scao et al., 2022; Zhang et al., 2022; Touvron et al., 2023a,b) have garnered increasing attention due to their exceptional capabilities across a diverse range of tasks. Either supervised fine-tuning or further pre-training can lead to enhanced performance. As the number of parameters grows, the substantial GPU memory required for training sets a high hardware threshold. Recently, Lv et al. (2023) has proposed low-memory optimization (LOMO) to train large language models in a memory-saving approach by simultaneously backpropagating gradients and updating parameters during the backward pass, enabling the fine-tuning of all parameters of a 7B model on a consumer-grade RTX 3090.

While LOMO's performance on the SuperGLUE (Wang et al., 2019) benchmark is comparable to popular parameter-efficient fine-tuning methods (Ding et al., 2023; Hu et al., 2022), it falls short on a broader range of tasks against adaptive optimization methods like Adam (Kingma and Ba, 2015), exhibiting a convergence gap. We attribute this to its reliance on the naive stochastic gradient descent optimization approach. We analyze the differences in optimization methods between Adam and LOMO. Compared to LOMO, Adam incorporates both the first and second moment estimation in its optimizer state, which are the moving averages of the gradient and the squared gradient, respectively. Based on our theoretical and empirical analysis, we identify that the second moment estimation is the pivotal factor influencing the convergence of training large language models between LOMO and Adam.

The second-order moment estimation in Adam serves to offer an adaptive learning rate for each parameter. Expanding on this concept, we introduce the low-memory optimization with adaptive learning rate (AdaLomo), which similarly provides an adaptive learning rate for each parameter, thus exhibiting superior convergence performance compared to LOMO theoretically. To retain memory efficiency, inspired by Adafactor (Shazeer and Stern, 2018), we employ non-negative matrix factorization (Yu et al., 2018) for the second-order moment estimation in the optimizer state. We advocate for the use of a grouped update normalization to stabilize convergence instead of global

*Work done during internship at Shanghai AI Laboratory.
†Corresponding author.



*Findings of the Association for Computational Linguistics: ACL 2024*, pages 12486–12502

| Method | Trainable Params (Billion) | Memory (GB) | | | |
|---|---|---|---|---|---|
| | | Param | Gradient | Optimizer State | Total |
| LoRA | $N$ | $2M$ | $O(N)$ | $O(N)$ | $\sim \mathbf{2M}$ |
| AdamW | $\boldsymbol{M}$ | $2M$ | $2M$ | $12M$ | $16M$ |
| AdaLomo | $\boldsymbol{M}$ | $2M$ | $O(N)$ | $O(N)$ | $\sim \mathbf{2M}$ |

Table 1: Trainable parameter number and memory usage under mixed-precision training. $N \ll M$ and $O(M+N) = O(M)$, where $M$ is the number of model parameters. AdaLomo's memory consumption is comparable to LoRA, and its trainable parameter number is equivalent to AdamW.

update normalization, which nearly doubles the training speed of AdaLomo while maintaining its performance. Moreover, under identical conditions, AdaLomo's memory utilization accounts for only approximately 40% of that consumed by Adafactor. The number of trainable parameters and the GPU memory consumption for model state under mixed-precision training among AdaLomo, the popular LoRA (Hu et al., 2022) method, and the AdamW optimizer (Loshchilov and Hutter, 2019) are compared in Table 1.

Our contributions are as follows:

1. We examined the distinctions between the LOMO and Adam optimization techniques. Analysis in Section 2.2 revealed that the primary difference in performance between LOMO and Adam, especially when training large language models, stems from Adam's incorporation of second-moment estimation.

2. We introduce AdaLomo, which provides an adaptive learning rate for each parameter while maintaining memory efficiency, democratizing the training of large language models. In AdaLomo, we also employ grouped update normalization to stabilize the training process.

3. We evaluate the performance of large language models post instruction-tuning with AdaLomo across five benchmarks spanning diverse tasks. The results are comparable to both AdamW and LoRA. Furthermore, when AdaLomo is used for pre-training from scratch and further pre-training on Chinese and Python code, its performance is on par with that of AdamW.

4. We profile the memory consumption and throughput of AdaLomo. Its reduced memory usage and reasonable computational overhead make it a viable option for training large language models.

## 2 Preliminaries

In the subsequent sections of this paper, we use $\boldsymbol{\theta_t}$ to denote the parameters of the model at the $t^{th}$ step of the training process. $\theta_{t,i}$ represents the parameter at the $i^{th}$ gradient computation during the backpropagation process of $\boldsymbol{\theta_t}$. We use $\boldsymbol{g_t}$ to represent the gradient of $\boldsymbol{\theta_t}$, and $g_{t,i}$ to denote the gradient of $\theta_{t,i}$. The first and second moment estimation at the $t^{th}$ training step, which are the moving averages of the gradient and the square of the gradient respectively, are represented by $\boldsymbol{m_t}$ and $\boldsymbol{v_t}$. The symbol $\alpha$ represents the learning rate.

### 2.1 Fused Backward

In the training process, the memory is primarily consumed by the optimizer states, parameters, and gradients. The fused backward proposed in LOMO refers to the process that simultaneously calculates gradients and updates parameters during backpropagation. This can effectively reduce the memory consumption of gradients.

For a given parameter $\theta_{t,i}$, its gradient $g_{t,i}$ resides in the GPU memory until the gradient $g_{t,i+1}$ corresponding to the subsequent parameter $\theta_{t,i+1}$ is computed. Subsequently, LOMO utilizes a standard gradient descent approach for parameter updates, as depicted by the following equation:

$$\theta_{t,i} = \theta_{t-1,i} - \alpha \times g_{t,i}. \tag{1}$$

For transformer-based language models, $g_{t,i}$ is unnecessary in subsequent backpropagation steps and can be eliminated from memory. Consequently, at any given moment, the memory retains the gradients of only two consecutive parameters. The memory usage for gradients remains constant regardless of the language model's scale, yielding an $O(1)$ memory footprint. In the case of large language models, such as LLaMA-65B (Touvron et al., 2023a) with its 82 layers and 723 weight matrices, the memory consumption for gradients becomes negligible compared to that for parameters or optimizer states.

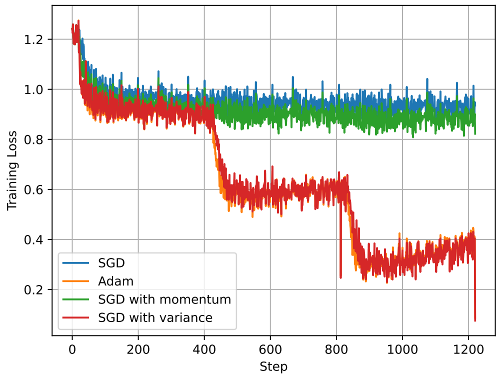



Figure 1: Empirical analysis on different optimization methods. Both Adam and SGD with variance exhibit a stepwise decline in loss as the training epochs increase, ultimately achieving a lower loss than both SGD and SGD with momentum.

**Gradient Normalization** Gradient normalization is instrumental in mitigating gradient vanishing and explosion. Nevertheless, its integration into LOMO presents challenges. Specifically, gradient normalization necessitates the computation of a scaling factor derived from the gradients of all parameters. This factor subsequently informs parameter updates. In the context of LOMO, however, the gradients for all parameters have not yet been computed. To incorporate gradient normalization within LOMO, two backward passes are essential: the first backward pass to get the the overall gradient scaling factor and the second updating the parameters using the derived scaling factor.

This process almost doubles the training time for LOMO. In AdaLomo, we employ grouped update normalization, which requires only a single backward pass to complete.

## 2.2 Analysis on the Two Moments in Adam

LOMO exhibits efficient memory usage, essentially pushing the optimization of large language models with gradients to the extreme. However, the naive gradient descent method shown in Equation 1 faces challenges such as the propensity to get trapped in saddle points and sensitivity to the learning rate (Dauphin et al., 2014; Darken et al., 1992). Building upon SGD, a series of advanced optimization methods have been proposed that have been proven both theoretically and practically to address these challenges (Ruder, 2016). These methods typically introduce additional optimizer states, such as momentum (Qian, 1999), Nesterov accelerated gradient (Nesterov, 1983), and moving averages of squared past gradients (Duchi et al., 2011; Zeiler, 2012; Kingma and Ba, 2015), leading to extra memory consumption. Among these, the Adam series of optimizers are most widely used in training large language models, simultaneously incorporating first-moment ($\boldsymbol{m_t}$) and second-moment ($\boldsymbol{v_t}$) estimation for parameter updates, as demonstrated in the following equation,

$$\begin{cases} \boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t, \\ \boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2, \\ \hat{\boldsymbol{m}}_t = \dfrac{\boldsymbol{m}_t}{1-\beta_1^t}, \\ \hat{\boldsymbol{v}}_t = \dfrac{\boldsymbol{v}_t}{1-\beta_2^t}, \\ \boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \dfrac{\hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{v}}_t}+\epsilon}, \end{cases} \tag{2}$$

where $\epsilon$ is a small quantity introduced to prevent division by zero in calculations. The hyperparameters $\beta_1, \beta_2 \in [0, 1)$ dictate the exponential decay rates of the respective moving averages.

**Theoretical Analysis** Qi et al. (2023) found that the SGD optimizer is highly sensitive to the network's Lipschitz constant. A significant variance in the Lipschitz constant across different layers results in substantial gradient disparities, leading to inconsistent step sizes in parameter updates with SGD. In contrast, the Adam optimizer employs an adaptive learning rate approach, normalizing update values and demonstrating robustness to variations in the Lipschitz constant. Kim et al. (2021) demonstrated that self-attention structures lack a bounded Lipschitz constant, suggesting that the gradient disparities across different layers in transformer architectures could be significant. Therefore, incorporating an adaptive learning rate into LOMO could enhance optimization for the widely used Transformer architecture (Vaswani et al., 2017).

**Empirical Analysis** We empirically investigated the differences in convergence behaviors between Adam and SGD under the fine-tuning of large language models. To ablatively analyze the roles of the first and second moments of the gradients in Adam, we conducted experiments retaining only the first-order moment estimate or the second-order moment estimation in Adam, respectively. The update rule retaining only the first-order moment estimation (or momentum) is:

$$\begin{cases} \boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t, \\ \hat{\boldsymbol{m}}_t = \dfrac{\boldsymbol{m}_t}{1-\beta_1^t}, \\ \boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \times \hat{\boldsymbol{m}}_t. \end{cases} \tag{3}$$

Meanwhile, the update rule retaining only the second-order moment estimation (or variance) is:

$$\begin{cases} \boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2, \\ \hat{\boldsymbol{v}}_t = \dfrac{\boldsymbol{v}_t}{1-\beta_2^t}, \\ \boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \dfrac{\boldsymbol{g}_t}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon}. \end{cases} \tag{4}$$

The results of the convergence analysis are shown in Figure 1. In the instruction-tuning scenario, we trained LLaMA-7B (Touvron et al., 2023a) with the Alpaca dataset (Taori et al., 2023; Wang et al., 2023) for three epochs. The loss curve of Adam during these three epochs exhibits a step-like decline, achieving a significantly smaller empirical loss compared to SGD.

Through our analysis on Adam above, we found that its second-order moment estimation has a significantly greater impact on its convergence than the first-order moment estimation. The second-order moment estimation is particularly effective for handling sparse data, allowing parameters that are infrequently updated to receive larger update steps.

Furthermore, the second-order moment in the optimizer's state has been proven to be decomposable or compressible to reduce memory usage. For example, Adafactor (Shazeer and Stern, 2018) decomposes the second moment $v_{t,i} \in \mathbb{R}^{m\times n}$ by minimizing the I-divergence into $r_{t,i} \in \mathbb{R}^{m\times 1}$ and $c_{t,i} \in \mathbb{R}^{1\times n}$ such that

$$v_{t,i} = r_{t,i} c_{t,i} / (\mathbf{1}_m^T r_{t,i}). \tag{5}$$

The update formulas for $r_{t,i}$ and $c_{t,i}$ in Adafactor are as follows:

$$r_{t,i} = \beta_1 r_{t-1,i} + (1-\beta_1) g_{t,i}^2 \, \mathbf{1}_n, \tag{6}$$

$$c_{t,i} = \beta_2 c_{t-1,i} + (1-\beta_2) \mathbf{1}_m^T g_{t,i}^2, \tag{7}$$

where $\mathbf{1}_n$ and $\mathbf{1}_m^T$ are all-ones vectors of dimensions $n \times 1$ and $1 \times m$, respectively.

## 3 Method

In this section, we introduce our proposed memory-efficient optimization algorithm, Adalomo. This algorithm has demonstrated performance comparable to the current de facto optimization method for large language models, AdamW, requiring less memory consumption.

### 3.1 AdaLomo

**Algorithm 1** AdaLomo

**Require:** model $f(\cdot)$ with parameter $\boldsymbol{\theta}$ , learning rate $\alpha$, max step $T$, training dataset $\mathcal{D}$, loss function $\mathcal{L}$, decay coefficient $\beta$, regularization constant $\epsilon$

```
for t = 1 to T do
    sample batch B = (x, y) ⊂ D
    ŷ ← f(x, θ)                                  ▷ forward pass
    ℓ ← L(y, ŷ)
    for each parameter θ_i in the order of back-propagation do
        g_{t,i} = ∇_{θ_{t-1,i}} ℓ                ▷ g_{t,i-1} needed for computing g_{t,i}
        r_{t,i} = β r_{t-1,i} + (1 − β) g_{t,i}^2 1_n
        c_{t,i} = β c_{t-1,i} + (1 − β) 1_m^T g_{t,i}^2
        v_{t,i} = r_{t,i} c_{t,i} / (1_m^T r_{t,i})
        u_{t,i} = g_{t,i} / v_{t,i}
        û_{t,i} = u_{t,i} / max(1, RMS(u_{t,i})) × max(ε, RMS(θ_{t-1,i}))
        θ_{t,i} = θ_{t-1,i} − α_t û_{t,i}
        g_{t,i-1} ← None                         ▷ clear g_{t,i-1}
    end for
end for
```

Based on the analysis in Section 2.2, to achieve improved optimization while maintaining low memory consumption, we decided to incorporate a second-order moment estimation and discard the first-order moment. In our pursuit of further memory efficiency, we applied non-negative matrix factorization to the second-order moment, inspired by Adafactor. Specifically, for each parameter $\theta_i$ within the model parameters $\boldsymbol{\theta}$, we introduce two optimizer states, $r_i$ and $c_i$. For parameters of size $m \times n$, we store only $r_i$ and $c_i$ instead of storing $v_i$. The size of the optimizer states is $m + n$, which is negligible compared to the size of the parameters.

Contrary to Adafactor, we update the optimizer state, update the parameters and discard the gradients during the gradient backpropagation process, which reduces our memory footprint to just 40%

| Model | Method | MMLU | BBH | GSM8K | HumanEval | AlpacaFarm | Avg. |
|---|---|---|---|---|---|---|---|
| LLaMA-7B | N/A | 31.5 | 32.3 | 10.9 | 11.6 | 4.2 | 18.1 |
| | LoRA | 33.5 | 34.8 | 12.3 | 11.0 | 41.1 | 26.5 |
| | AdamW | 39.3 | 34.4 | 9.6 | 11.6 | 50.6 | 29.1 |
| | LOMO | 30.7 | 34.0 | 12.0 | **12.8** | 30.6 | 24.0 |
| | AdaLomo | **39.5** | **36.0** | **14.4** | 11.0 | **53.3** | **30.8** |
| LLaMA-13B | N/A | 45.2 | 38.5 | 19.5 | 14.0 | 5.3 | 24.5 |
| | LoRA | 48.3 | 40.3 | 20.2 | **19.5** | 49.1 | 35.5 |
| | AdamW | 49.4 | 40.2 | 21.8 | 18.9 | 61.0 | 38.2 |
| | LOMO | 44.2 | 38.9 | 21.3 | 16.5 | 38.4 | 31.8 |
| | AdaLomo | **50.0** | **41.5** | **25.3** | 18.9 | **62.9** | **39.7** |
| LLaMA-30B | N/A | 57.7 | 51.8 | 40.3 | 20.1 | 7.1 | 35.4 |
| | LoRA | 59.3 | **52.3** | 42.8 | **26.2** | 63.3 | 48.8 |
| | AdamW | 57.3 | 49.5 | 36.6 | 21.3 | 65.5 | 46.1 |
| | LOMO | 56.3 | 51.5 | 44.4 | 18.9 | 57.8 | 45.8 |
| | AdaLomo | **59.4** | 52.1 | **48.5** | 25.6 | **69.6** | **51.0** |
| LLaMA-65B | N/A | 62.4 | 58.7 | 53.9 | 20.7 | 4.7 | 40.1 |
| | LoRA | 62.7 | 58.7 | **60.5** | **32.9** | 69.6 | **56.9** |
| | AdamW | **63.0** | 57.9 | 55.3 | 28.1 | 73.1 | 55.5 |
| | LOMO | 62.1 | 56.9 | 57.6 | 28.1 | 65.2 | 54.0 |
| | AdaLomo | 62.7 | **59.0** | 59.7 | 29.9 | **73.4** | **56.9** |

Table 2: Performance of the LLaMA series models on various benchmarks after instruction-tuning with different optimization techniques. Bolded numbers indicate the best results for models of the same size on a given benchmark. “N/A" denotes that no instruction-tuning is performed.

of that required by Adafactor. During parameter updates, we compute $v_i = r_i c_i$ using $r_i$ and $c_i$ to provide adaptive learning rate for the parameters, which ensures that the optimization of AdaLomo is theoretically superior to that of LOMO based on the preceding analysis. Additionally, we employ grouped update normalization, which nearly doubles training speed compared to the naive gradient norm used in LOMO. The details of the algorithm are presented in Algorithm 1.

### 3.2 Grouped Update Normalization

We utilize grouped update normalization in the AdaLomo update process, which entails adaptive modifications for the update of each parameter and helps maintain model stability especially during large-scale training. Grouped update normalization ensures that each parameter's update is meaningful and not overshadowed by large gradient values from other parameters, facilitating faster convergence and sustained stability. In contrast, global update normalization, where all parameters share a single scaling factor, might lead to some parameters updating too rapidly or too slowly, thereby affecting both convergence speed and stability. This is especially evident in large language models where different layers and parameters can exhibit considerable variations in gradient magnitudes, rendering global scaling potentially less effective.

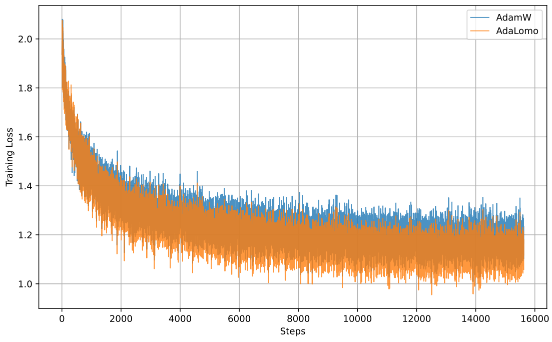
As shown in line 11 of Algorithm 1, for the update matrix $u_i$ for parameter $\theta_i$, before applying it to the weight matrix, we divide it by the parameter-wise root-mean-square (RMS) of $u_i$ [1]. Additionally, we utilize the parameter-wise RMS of $\theta_i$ to ensure the update step size is proportional to the magnitude of the parameter.

Furthermore, it's worth noting that grouped update normalization integrates seamlessly with AdaLomo's fused backward process. While global update normalization requires two backward passes as gradient normalization mentioned in Section 2.1, grouped update normalization allows us to normalize the update matrices within a single fused backward pass.

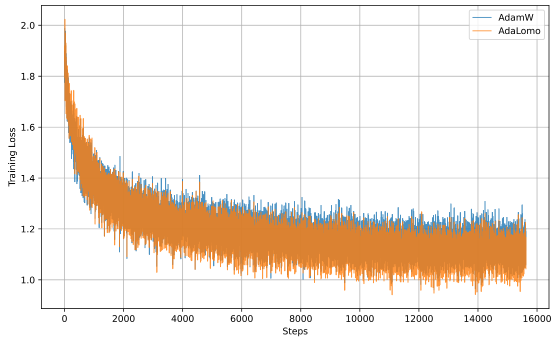
## 4 Experiments

In this section, we evaluate the efficacy of AdaLomo in instruction-tuning, further pre-training and from-scratch pre-training. Additionally, we assess memory usage and throughput. Ex-

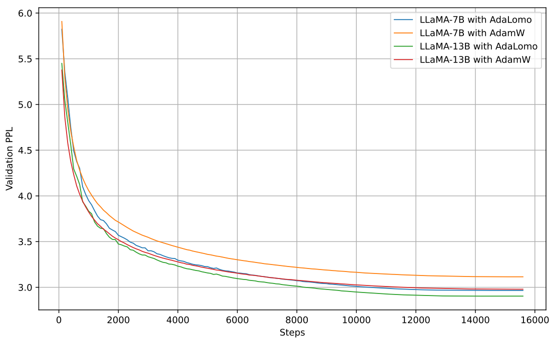
[1] The root-mean-square (RMS) of $u$ is given by $RMS(u) = \sqrt{\frac{\sum_{i=1}^{n} u_i^2}{n}}$, where $n$ is the number of elements in $u$.



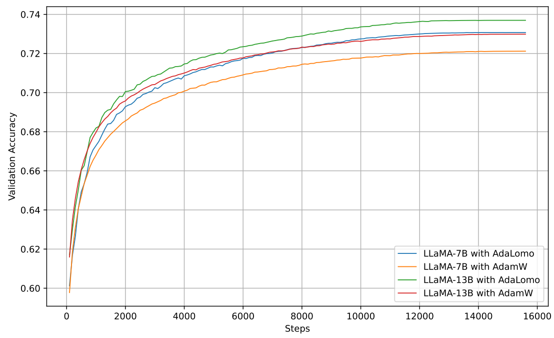

(a) Training loss curve for LLaMA-7B.




(b) Training loss curve for LLaMA-13B.




(c) Perplexity of the validation set.




(d) Next-token accuracy of the validation set.

Figure 2: Results of further pre-training in the Chinese domain.

periments are performed using the LLaMA series of models, which have parameter sizes ranging from 7 billion to 65 billion.

## 4.1 Instruction Tuning

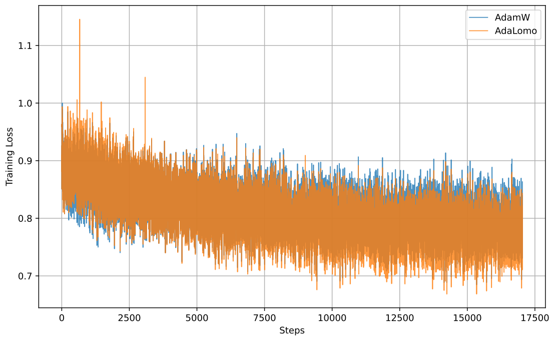
We utilized GPT-4-Alpaca (Peng et al., 2023) as the training data to fine-tune LLaMA, incorporating 52k instruction-following demonstrations generated by GPT-4 using the Alpaca method. Besides the unaltered vanilla model and LOMO, we compared LoRA and AdamW, two prevalent methods for instruction-tuning large language models, which act as strong baselines.

We evaluated the trained models across diverse tasks: knowledge-based tasks (MMLU (Hendrycks et al., 2021)), general reasoning tasks (BBH (Suzgun et al., 2023)), mathematical tasks (GSM8K (Cobbe et al., 2021)), coding tasks (HumanEval (Chen et al., 2021)), and instruction-following tasks (AlpacaFarm (Dubois et al., 2023)). For MMLU, BBH, and GSM8K, the answers are obtained by generating, and are assessed using accuracy. The HumanEval task is evaluated using pass@1. The AlpacaFarm task is assessed by comparing the win rate of responses against those from GPT-3.5 (Brown et al., 2020), as scored by GPT-4 (OpenAI, 2023). Training and evaluation are conducted using templates provided in the Alpaca repository.

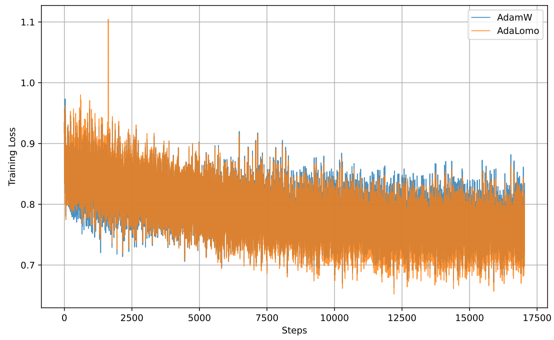
The results are presented in Table 2. Compared to the vanilla model, models trained using these methods generally exhibit improved performance, especially in instruction-following capabilities. LOMO's performance on general reasoning (BBH), mathematics (GSM8K), and coding (HumanEval) tasks was comparable to that of LoRA and AdamW across all model sizes. However, its performance on knowledge-based tasks (MMLU) and instruction-following tasks (AlpacaFarm) is relatively inferior. The performance gap between LOMO and both LoRA and AdamW on these two tasks decreases as the model size increases. By incorporating the second-order moment estimation, AdaLomo addresses LOMO's limitations, achieving comparable results with AdamW across various benchmarks for all model sizes.

## 4.2 Further Pre-training

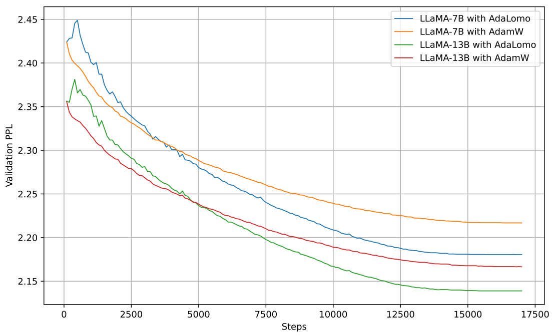
Further pre-training refers to the additional large-scale unsupervised learning applied to a pre-trained model. We conduct further pre-training on the LLaMA model with parameter sizes of 7B and 13B in two domains: Chinese and Python code. The LLaMA model had limited exposure to data from



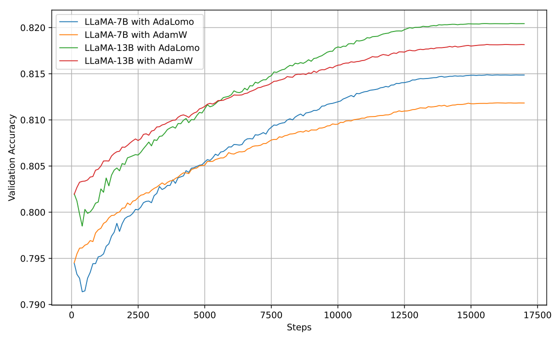

(a) Training loss curve for LLaMA-7B.




(b) Training loss curve for LLaMA-13B.




(c) Perplexity of the validation set.




(d) Next-token accuracy of the validation set.

Figure 3: Results of further pre-training in the Python code domain.

these two domains during its initial pre-training phase. Baidu-baike is a Chinese online encyclopedia. We scraped 2 million entries from Baidu-baike for further pre-training in the Chinese domain. Additionally, we extracted 2.2 million entries from the Python subset of the StarCoder (Li et al., 2023) training dataset for further pre-training in the Python code domain. Beyond this, we set aside 2,000 entries as a validation set.

We choose AdamW as the baseline for comparison. The training hyper-parameters and data samples are detailed in Appendix D. We plot the loss curve during the model's training process and tested the perplexity and accuracy of the next-token prediction every 100 steps on the validation set.

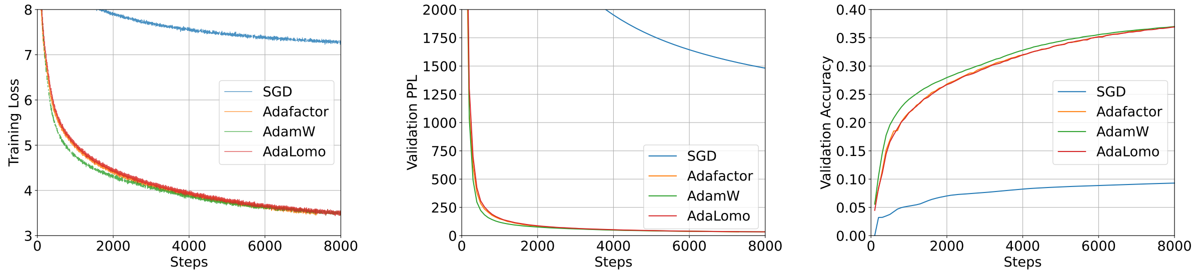
As shown in Figure 2a and 2b, during the further pre-training in Chinese, the loss curves of AdaLomo and AdamW overlap significantly, with AdaLomo's curve slightly below that of AdamW. The fluctuation range of their losses is at a similar level. Figure 2c and 2d also indicate that AdaLomo ultimately achieved a slightly lower perplexity and accuracy on the validation set than AdamW. Both methods effectively reduced LLaMA's perplexity in Chinese.

Figure 3 presents the results of further pre-training in the Python code domain. The overall findings are similar to those in the Chinese domain, with some differences. The enhancement of LLaMA's capabilities in the Python code domain through further pre-training is relatively less pronounced. This is because, in terms of perplexity, the original LLaMA performs better on Python code than on Chinese. Although AdaLomo exhibited some fluctuations during the initial warmup phase (with a perplexity difference of less than 0.02), it converged to a more optimal point at a faster rate thereafter. The LLaMA-13B model exhibited less fluctuation than the LLaMA-7B model. We attribute these fluctuations to AdaLomo's reliance on $\boldsymbol{g}_t^2$ over $\boldsymbol{v}_{t-1}$ during the early stages of training, and the fact that AdaLomo does not utilize momentum.

Grouped update normalization effectively substitutes the role of gradient normalization. It enables stable training even without the use of gradient normalization, which is essential to prevent gradient explosion but with a decrease in throughput for LOMO. A detailed comparison regarding gradient normalization are shown in Appendix B.

### 4.3 Pre-training from Scratch

We conducted a from-scratch pre-training on the C4 corpus (Raffel et al., 2020) using a model with




(a) Training loss curve. (b) Validation perplexity. (c) Validation next-token accuracy.

Figure 4: Results of pre-training LLaMA-1.1B from scratch on C4 corpus.

1.1 billion parameters based on the LLaMA architecture[2]. The batch size was set to 1024, with a maximum data length of 2048 tokens, and warmup steps of 300 using a cosine scheduler. We report the training loss for the first 8000 steps, along with the perplexity and accuracy on the validation set, as shown in Figure 4.

Our results indicate that AdamW, Adafactor, and AdaLomo exhibit comparable convergence performance, significantly outperforming SGD. This highlights the effectiveness of AdaLomo in the pre-training context.

### 4.4 Memory and Throughput Profile

We evaluate the max allocated memory and throughput of AdamW, Adafactor, LoRA, LOMO, and AdaLomo, with the results in Figure 5. We employ ZeRO-3 (Rajbhandari et al., 2020) for distributed training. Throughput is measured in terms of tokens processed per GPU per second (TGS). Detailed numerical results and more specific experimental settings can be found in Appendix F.

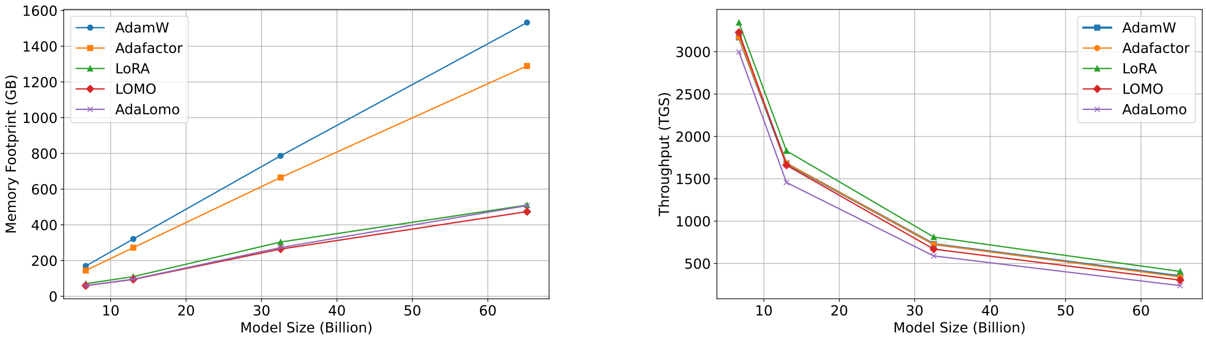
Among the evaluated methods, AdamW exhibits the highest memory consumption. Adafactor reduces memory usage compared to AdamW by decomposing the second-order moment, resulting in memory savings proportional to the model's parameter size. AdaLomo, in comparison to LOMO, introduce an adaptive learning rate for each parameter. Nevertheless, its memory consumption remains close to that of LOMO and is comparable to LoRA, which trains with very few parameters. Due to fewer trainable parameters requiring communication during training, LoRA achieves the highest throughput. AdaLomo, which necessitates additional computations during parameter updates, shows slightly lower throughput than LOMO. All methods are tested with a consistent batch size, yet AdaLomo retains residual memory capacity, suggesting the potential for an increased batch size and greater throughput. Overall, the throughput of these methods is at the same level.

[2]The architecture is consistent with TinyLlama-1.1B (Zhang et al., 2024).

## 5 Related Work

Previous research has extensively explored memory-efficient optimizers. Adafactor (Shazeer and Stern, 2018) employs non-negative matrix factorization and approximates the second-order moment estimate $\boldsymbol{v} \in \mathbb{R}^{m \times n}$ using the outer product of $\boldsymbol{r} \in \mathbb{R}^{m \times 1}$ and $\boldsymbol{c} \in \mathbb{R}^{1 \times n}$, achieving sublinear memory consumption. The SM3 algorithm (Anil et al., 2019) introduces the cover of the parameters or, more specifically, a set of $k$ non-empty parameter groups. Each parameter is assigned an adaptive learning rate based on this cover. For a parameter matrix of size $m \times n$, the sets can be divided by rows and columns, resulting in $m + n$ sets. This reduces the memory requirement from $O(m \times n)$ to $O(m + n)$, analogous to Adafactor's memory consumption. Another line to reduce memory usage is by utilizing low-precision storage for the optimizer state. Ramesh et al. (2021) and Rae et al. (2021) explored the stability of 16-bit optimizers. The 8-bit Optimizer (Dettmers et al., 2022), using block-wise and dynamic exponent quantization, quantizes the optimizer states of SGDM and Adam to 8 bits. The 4-bit optimizer (Sun et al., 2020), employing the newly proposed FP4 format and the adaptive gradient scaling technique. To decrease the memory used by gradients, LOMO updates parameters simultaneously during the gradient computation in the backward pass.

Additionally, there exists a series of memory-efficient optimization methods designed exclusively for fine-tuning. BBT (Sun et al., 2022b) and BBTv2 (Sun et al., 2022a) utilize evolutionary gradient-free algorithms to optimize continu-




(a) Memory usage with different methods.

(b) Throughput with different optimization methods.

Figure 5: Memory footprint and throughput using different optimization methods.

ous prompts without model updates. MeZO (Malladi et al., 2023) employs zeroth-order optimization methods, estimating gradients using two forward passes and optimizing the model in-place, thus equating memory consumption with inference. Parameter-efficient fine-tuning (PEFT) (Ding et al., 2023) methods selectively add or pick a subset of parameters for optimization, freezing the majority of the model parameters. In comparison, AdaLomo updates all parameters using a gradient-based method, suitable for both pre-training and fine-tuning, with memory consumption comparable to PEFT methods.

## 6 Conclusion

In this paper, we introduce AdaLomo, designed to reduce the training barriers for large language models. By incorporating an adaptive learning rate and utilizing grouped update normalization, AdaLomo achieves results comparable to AdamW in instruction-tuning, further pre-training and from-scratch pre-training. Concurrently, the memory footprint of AdaLomo is on par with the PEFT methods.

## Limitations

While AdaLomo is memory-efficient when training large language models, it primarily reduces the memory occupied by gradients and the optimizer states. Therefore, for models with a significant amount of activation values occupying memory, the reduction in memory usage by employing AdaLomo is limited. Thus, AdaLomo is best suited for training models with a large number of parameters. Additionally, while our experiments show that the throughput decrease is minimal, AdaLomo introduce some extra computational overhead, suggesting a direction for further improvement. This framework can be extended to optimizers using other update methods, such as SM3, and can also be adapted to methods related to optimizer states compression.

## Ethics statement

This paper employs open-source models LLaMA and the OpenAI API, all in compliance with their respective licenses. The datasets utilized, including AlpacaGPT4, MMLU, BBH, GSM8K, HumanEval and AlpacaFarm, permit public and freeusage. Resources used in constructing further pre-training datasets are openly available.

## Acknowledgments

This work was supported by the National Key Research and Development Program of China (No.2022ZD0160102). The computations in this research were performed using the CFFF platform of Fudan University.

## References

Rohan Anil, Vineet Gupta, Tomer Koren, and Yoram Singer. 2019. Memory efficient adaptive optimization. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 9746–9755.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Christian Darken, Joseph Chang, John Moody, et al. 1992. Learning rate schedules for faster stochastic gradient search. In *Neural networks for signal processing*, volume 2, pages 3–12. Citeseer.

Yann N. Dauphin, Razvan Pascanu, Çaglar Gülçehre, KyungHyun Cho, Surya Ganguli, and Yoshua Bengio. 2014. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2933–2941.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 2022. 8-bit optimizers via block-wise quantization. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nat. Mac. Intell.*, 5(3):220–235.

Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Alpacafarm: A simulation framework for methods that learn from human feedback. *CoRR*, abs/2305.14387.

John C. Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Hyunjik Kim, George Papamakarios, and Andriy Mnih. 2021. The lipschitz constant of self-attention. In *International Conference on Machine Learning*, pages 5562–5571. PMLR.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy V, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Moustafa-Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. Starcoder: may the source be with you! *CoRR*, abs/2305.06161.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

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

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D. Lee, Danqi Chen, and Sanjeev Arora. 2023. Fine-tuning language models with just forward passes. *CoRR*, abs/2305.17333.

Yurii Nesterov. 1983. A method for unconstrained convex minimization problem with the rate of convergence o (1/k2). In *Dokl. Akad. Nauk. SSSR*, volume 269, page 543.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with GPT-4. *CoRR*, abs/2304.03277.

Xianbiao Qi, Jianan Wang, and Lei Zhang. 2023. Understanding optimization of deep learning via jacobian matrix and lipschitz constant.

Ning Qian. 1999. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151.

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, H. Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant M. Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew J. Johnson, Blake A. Hechtman, Laura Weidinger, Iason Gabriel, William Isaac, Edward Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. 2021. Scaling language models: Methods, analysis & insights from training gopher. *CoRR*, abs/2112.11446.

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

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, page 20. IEEE/ACM.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8821–8831. PMLR.

Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilic, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, and et al. 2022. BLOOM: A 176b-parameter open-access multilingual language model. *CoRR*, abs/2211.05100.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4603–4611. PMLR.

Tianxiang Sun, Zhengfu He, Hong Qian, Yunhua Zhou, Xuanjing Huang, and Xipeng Qiu. 2022a. Bbtv2: Towards a gradient-free future with large language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 3916–3930. Association for Computational Linguistics.

Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. 2022b. Black-box tuning for language-model-as-a-service. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 20841–20855. PMLR.

Xiao Sun, Naigang Wang, Chia-Yu Chen, Jiamin Ni, Ankur Agrawal, Xiaodong Cui, Swagath Venkataramani, Kaoutar El Maghraoui, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2020. Ultra-low precision 4-bit training of deep neural networks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed Chi, Denny Zhou, and Jason Wei. 2023. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13003–13051. Association for Computational Linguistics.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

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

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *CoRR*, abs/1905.00537.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13484–13508. Association for Computational Linguistics.

Jinshi Yu, Guoxu Zhou, Andrzej Cichocki, and Shengli Xie. 2018. Learning the hierarchical parts of objects by deep non-smooth nonnegative matrix factorization. *IEEE Access*, 6:58096–58105.

Matthew D. Zeiler. 2012. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: open pre-trained transformer language models. *CoRR*, abs/2205.01068.

# A Empirical Analysis on the Two Moments

We also empirically investigated the differences in convergence behaviors between Adam and SGD under the function $f(x, y) = x^2 + y^2 - 2e^{-5[(x-1)^2+y^2]} - 3e^{-5[(x+1)^2+y^2]}$.

The results of the convergence analysis are shown in Figure 6. Starting from the same initial point, Adam converges to the global optimum while SGD gets trapped at a local optimum.

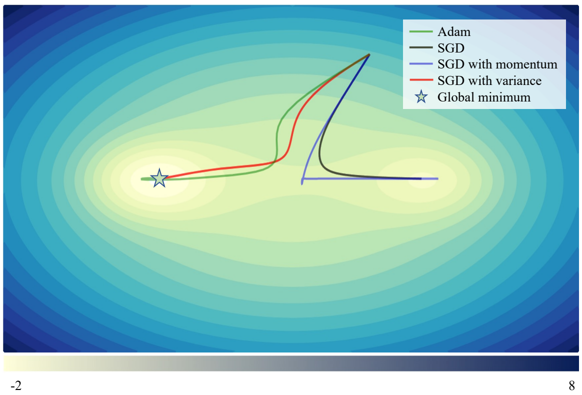



Figure 6: Empirical analysis on different optimization methods. Loss trajectories of different optimizers starting from the same initial point. Both Adam and SGD with variance converge to the global optimum on the left, while SGD and SGD with momentum converge to the local optimum on the right.

# B Gradient Normalization for AdaLomo

We conduct experiments on the LLaMA-7B to assess the effects of using gradient normalization during the further pre-training of AdaLomo. Comparative experiments in the Chinese domain are illustrated in Figure 7, while those in the Python code domain are shown in Figure 8. Our results indicate that the convergence performance of AdaLomo is unaffected by the use or absence of gradient normalization. We attribute this to the grouped update normalization feature within AdaLomo. Avoiding the use of gradient normalization can eliminate the need for two backward passes, thus preventing computational redundancy during training.

# C Instruction Tuning

## C.1 Hyper-parameters

Hyper-parameters used by different optimization methods and models for instruction-tuning are shown in Table 3.

## C.2 Templates

Templates used for instruction-tuning on Alpaca-GPT4 are shown in Table 4.

## C.3 More Results

In Table 5, we include a comparison of Adafactor on LLaMA-7B. The results show that Adafactor's performance is similar to AdaLomo's. Both Adafactor and AdaLomo significantly outperform LOMO on instruction-following task (Alpaca-Farm).

# D Further Pre-training

## D.1 Hyper-parameters

Hyper-parameters used for further pre-trianing are shown in Table 6.

## D.2 More Results

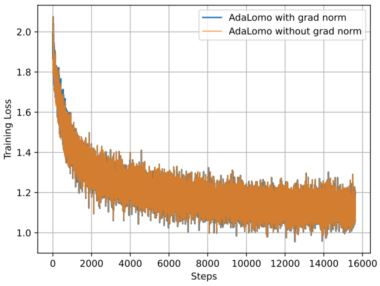
We present the results of further pre-training in the Chinese domain and the Python code domain on the LLaMA-7B model in Figure 9 and Figure 10, respectively. It can be observed that AdaLomo, AdamW, and Adafactor exhibit similar convergence speeds and final performance, while SGD performs poorly in both domains. This experiment confirms our hypothesis: second-order moments are crucial for optimizing transformer-based large language models.

# E Pre-training from Scratch

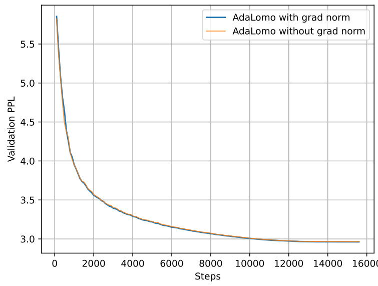
Our experimental comparisons and learning rates are shown in Table 7, with AdamW's weight decay set to 0.01.

# F Memory and Throughput Profile

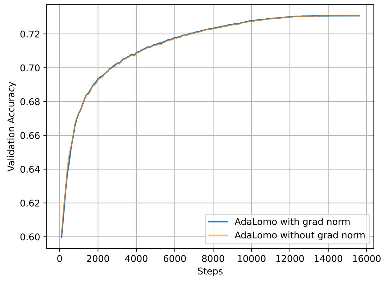
The hyper-parameters used to profile memory and throughput and the detailed results are shown in Table 8. The experiments are conducted on A800 with NVLink. For practical scenarios, we employ pynvml (Python NVIDIA Management Library) to record system-level memory usage.




(a) Training loss curve.

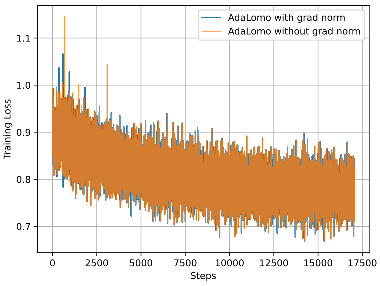



(b) Validation perplexity.

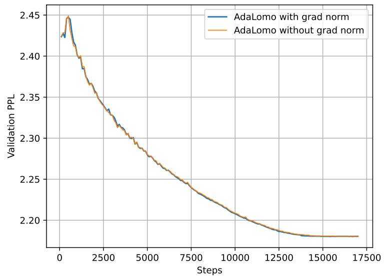



(c) Validation next-token accuracy.

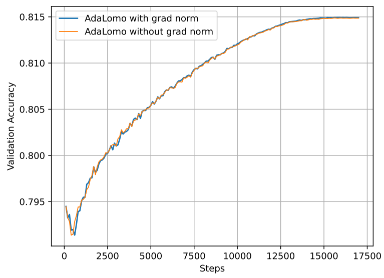
Figure 7: Results of further pre-training of LLaMA-7B with AdaLomo in the Chinese domain with and without gradient normalization.




(a) Training loss curve.

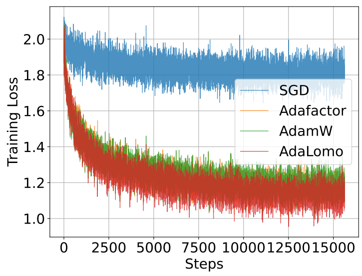



(b) Validation perplexity.

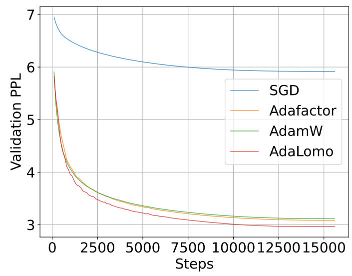



(c) Validation next-token accuracy.

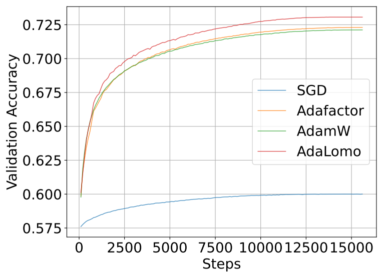
Figure 8: Results of further pre-training of LLaMA-7B with AdaLomo in the Python code domain with and without gradient normalization.




(a) Training loss curve.

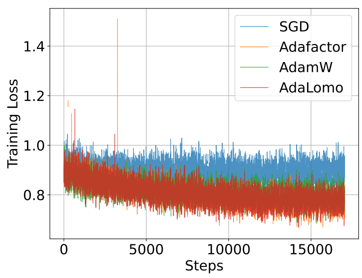



(b) Validation perplexity.

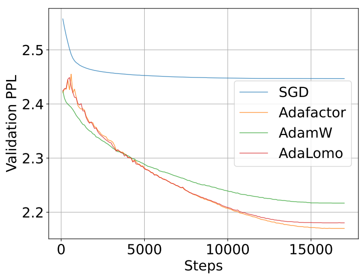



(c) Validation next-token accuracy.

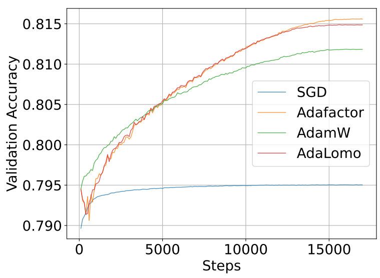
Figure 9: Results of further pre-training in the Chinese domain.




(a) Training loss curve.




(b) Validation perplexity.




(c) Validation next-token accuracy.

Figure 10: Results of further pre-training in the Python code domain.

| | LLaMA-7B | | | | LLaMA-13B | | | |
|---|---|---|---|---|---|---|---|---|
| | LoRA | AdamW | LOMO | AdaLomo | LoRA | AdamW | LOMO | AdaLomo |
| Learning Rate | 3E-04 | 2E-05 | 1E-02 | 5E-04 | 3E-04 | 2E-05 | 1E-02 | 5E-04 |
| Batch Size | 128 | | | | | | | |
| Ecochs | 3 | | | | | | | |
| Warmup Steps | 0.03 * Total Steps | | | | | | | |
| | LLaMA-30B | | | | LLaMA-65B | | | |
| | LoRA | AdamW | LOMO | AdaLomo | LoRA | AdamW | LOMO | AdaLomo |
| Learning Rate | 3E-04 | 2E-05 | 1E-02 | 5E-04 | 3E-04 | 1E-05 | 1E-02 | 5E-04 |
| Batch Size | 128 | | | | | | | |
| Ecochs | 3 | | | | | | | |
| Warmup Steps | 0.03 * Total Steps | | | | | | | |

Table 3: Hyper-parameters for instruction-tuning.

| **Template for entries with input** |
|---|
| Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.<br><br>### Instruction:<br>{instruction}<br><br>### Input:<br>{input}<br><br>### Response:{response} |
| **Template for entries without input** |
| Below is an instruction that describes a task. Write a response that appropriately completes the request.<br><br>### Instruction:<br>{instruction}<br><br>### Response:{response} |

Table 4: Templates used for instruction-tuning.

| Model | MMLU | BBH | GSM8K | HumanEval | AlpacaFarm | Avg. |
|---|---|---|---|---|---|---|
| LLaMA-7B | 31.5 | 32.3 | 10.9 | 11.6 | 4.2 | 18.1 |
| LoRA | 33.5 | 34.8 | 12.3 | 11.0 | 41.1 | 26.5 |
| AdamW | 39.3 | 34.4 | 9.66 | 11.6 | 50.6 | 29.1 |
| LOMO | 30.7 | 34.0 | 12.0 | **12.8** | 30.6 | 24.0 |
| Adafactor | **40.8** | 35.8 | **14.9** | 11.0 | 47.7 | 30.0 |
| AdaLomo | 39.5 | **36.0** | 14.4 | 11.0 | **53.3** | **30.8** |

Table 5: Performance of the LLaMA-7B after instruction-tuning with different optimization techniques.

| Method | AdamW | AdaLomo |
|---|---|---|
| Sequence Length | 2048 | |
| Learning Rate | 1E-05 | 3E-01 |
| Batch Size | 128 | |
| Warmup Steps | 0.03 * Total Steps | |

Table 6: Hyper-parameters used for further pre-training.

| | SGD | Adafactor | AdamW | AdaLomo |
|---|---|---|---|---|
| LR | 1e-3 | 1e-3 | 2e-5 | 1e-3 |

Table 7: Hyper-parameters for pre-training from scratch.

| Model | Optimizer | GPUs | Micro Batch Size | Memory (GB) | Throughput (TGS) |
|---|---|---|---|---|---|
| LLaMA-7B | AdamW | 4 | 8 | 169.4 | 3169.4 |
| | Adafactor | | | 144.3 | 3169.5 |
| | LoRA | | | 70.6 | 3344.6 |
| | LOMO | | | 59.6 | 3228.2 |
| | AdaLomo | | | 59.6 | 2997.4 |
| LLaMA-30B | AdamW | 16 | 4 | 786.2 | 728.6 |
| | Adafactor | | | 665.0 | 726.5 |
| | LoRA | | | 303.7 | 811.6 |
| | LOMO | | | 264.3 | 669.1 |
| | AdaLomo | | | 272.8 | 589.0 |
| LLaMA-13B | AdamW | 8 | 4 | 320.7 | 1679.6 |
| | Adafactor | | | 272.3 | 1683.4 |
| | LoRA | | | 110.0 | 1829.8 |
| | LOMO | | | 94.4 | 1659.9 |
| | AdaLomo | | | 95.8 | 1456.3 |
| LLaMA-65B | AdamW | 32 | 2 | 1532.6 | 349.1 |
| | Adafactor | | | 1289.4 | 341.1 |
| | LoRA | | | 510.5 | 405.7 |
| | LOMO | | | 473.8 | 303.3 |
| | AdaLomo | | | 507.7 | 238.1 |

Table 8: Hyper-parameters and detailed results in memory and throughput profile.