Taming LLMs with Gradient Grouping

Anonymous ACL submission

Abstract

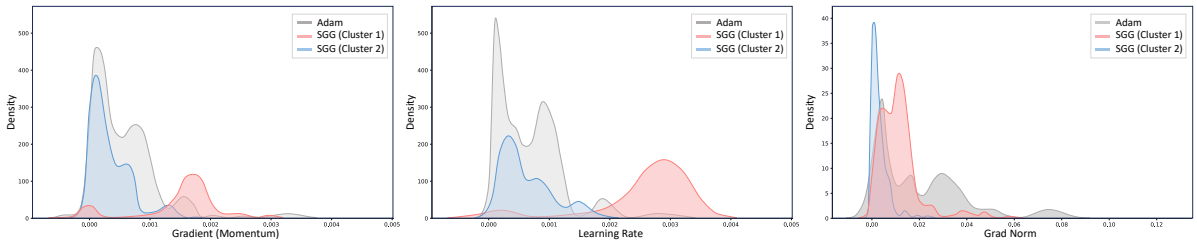
Training large language models (LLMs) poses unique challenges due to their increasing parameter counts and heterogeneous architectures. While adaptive optimizers like AdamW help adapt parameter-wise gradient variations, they struggle to estimate learning rates across LLM parameters efficiently, which causes training instability, inefficient convergence, and poor compatibility with parameter-efficient fine-tuning (PEFT) techniques such as LoRA. Inspired by the low-rank properties of LLMs, we introduce **S**caling with **G**radient **G**rouping (**SGG**), an optimizer wrapper that exploits the inherent low-rank structures to adapt learning rates through gradient clustering and cluster-specific scaling. SGG groups gradients into K clusters per T iterations and computes cluster-specific statistics to calibrate step sizes, simplifying learning rate estimation from a high-dimensional problem to tractable low-dimensional ones. As a modular wrapper, SGG integrates effortlessly with mainstream optimizers while maintaining PEFT compatibility. Experiments on C4, GLUE, and Alpaca show that SGG consistently achieves faster convergence and lower losses compared to existing methods. Furthermore, SGG's robustness across varying batch sizes and learning rates makes it a promising choice for efficient and effective LLM optimization.

1 Introduction

Optimization techniques have long served as the cornerstone in today's deep learning systems, enabling the training of increasingly powerful large language models (LLMs) that drive breakthroughs across various domains. Among these, adaptive optimizers (Kingma and Ba, 2015; Loshchilov and Hutter, 2019; You et al., 2020) stand out due to their ability to dynamically adjust learning rates for each parameter, which is beneficial given the heterogeneous LLM architectures (Liu et al., 2020; Zhang et al., 2025b; Li et al., 2024c). Yet, as LLMs scale to billions of parameters, these approaches may impose substantial memory overhead due to the storage of gradient statistics (*e.g.*, first/second moments), which limits their practical applicability, especially for resource-constrained scenarios.

To deal with this cost, parameter-efficient fine-tuning (PEFT) (Hu et al., 2021; Dettmers et al., 2024) has garnered increasing attention, which exploits the intrinsic low-rank structure within LLMs to reduce trainable parameters. However, such an efficiency often comes at the price of performance degradation compared to their full-rank counterpart (Table 4), while requiring extra architecture modification through auxiliary low-rank branches. In parallel, efforts have been made for memory-efficient optimizers (Shazeer and Stern, 2018; Luo et al., 2023; Zhu et al., 2024a), which aim to compress optimizer states by approximating gradient statistics. Though memory-efficient, specific gradient information might be discarded due to the heuristic priors, resulting in inconsistent performance across tasks (Table 5). This leaves practitioners at a deadlock: either accept performance drop with LoRA, or incur unstable gains via gradient decomposition.

In this paper, we argue that while each model parameter requires its own gradient statistics, LLM training may benefit from striking a balance between naive per-parameter tuning and fixed global ones. On the one hand, recent studies show that attention and MLP layers exhibit distinct yet internally consistent gradient behaviors (Li et al., 2024c; Zhang et al., 2025b), revealing significant patterns that previous methods fail to exploit. This observation aligns well with the low-rank priors in LLM parameters, where a small parameter count could play a dominant role in model capacity. Thus, naively tuning gradient statistics for each parameter may not only waste resources but destabilize the training, as outliers and noisy gradients may influence the optimization process (Zhao et al., 2024a; Nasiri and Garraghan, 2025), as summarized in Table 1.



(a) Gradiant Distribution (b) Learning Rate Distribution (c) Grad Norm Distribution

Figure 1: **Clustering Phenomenon in LLM Pre-training** with LLaMA-1B on C4. Selecting parameters from the 12-th FFN layers, **(a)** and **(b)** visualize distributions of parameter-wise gradients g^t and the corresponding adaptive learning rates α^t at the $5k$-th iteration, where SGG captures two clusters of gradients and scaling with two groups of learning rates. **(c)** shows group-wise (or layer-wise) L_2-grad norm distributions of all layers, where SGG captures the inherent heterogeneous patterns of different layers with group-wise learning rate scaling.

Along this line, we first conduct pilot studies with LLaMA-1B (in Figure 1) to examine gradient statistics distributions during LLM training. Interestingly, it shows consistent salient patterns with a few distinct peaks, which lines up with our intuition above. This suggests that per-parameter learning rate adaptation may be redundant, and there is room for improving efficiency along this line. Based on these insights, we introduce **S**caling with **G**radient **G**rouping (**SGG**), an optimizer wrapper that bridges this gap by unifying sparse gradient patterns with learning rate adaptation. As shown in Algorithm 1, SGG dynamically groups parameter-wise gradients of each layer through mini-batch K-Means clustering. Subsequently, it employs median absolute deviation (MAD) to measure and calibrate learning rates for each group. Thus, SGG achieves a balance between global and local adaptation while mitigating training instability.

Experiments on C4, GLUE, and Alpaca show that SGG not only improves convergence but naturally aligns with the low-rank priors of PEFT updates, achieving superior match with PEFT techniques (Table 4). More importantly, SGG operates as a plug-and-play module, providing compatibility with mainstream optimizers and PEFT techniques without architecture modifications.

Our contributions can be summarized as follows:

- We introduce SGG, an optimizer wrapper that leverages gradient clustering and cluster-specific scaling to adaptively adjust learning rates for robust and effective LLM training.
- We show that SGG is compatible with existing optimizers and PEFT techniques, offering a plug-and-play solution without extensive modifications to the overall LLM training pipeline.
- Through extensive experiments on C4, GLUE, and Alpaca, we demonstrate that SGG can bring **faster convergence**, **lower loss values**, and **reduced oscillation** compared to state-of-the-art methods for LLM training.

2 Methodology

2.1 Problem Defenition

In this subsection, we first preview the key steps of mainstream optimizers with Algorithm 1, to clarify how SGG integrates as a plug-and-play module.

The initial optimization step computes gradients g_l with respect to model parameters θ_l from the l-th layer. This procedure is typically carried out iteratively using gradient-based optimizers. At iteration t, the computation of gradient g_l^t is given by:

$$g_l^{(t)} = \nabla_{\theta_l^{(t-1)}} \mathcal{L}(\theta_l^{(t-1)}, \mathcal{D}) \quad (1)$$

where $\mathcal{D}$ denotes the training dataset and θ_l^{t-1} is the identical parameter updated at last iteration. Next, momentum estimation introduces historical gradient information to stabilize updates. As for vanilla SGD (Sinha and Griscik, 1971), this step is trivial ($m_l^t = g_l^t$), while momentum-based optimizers employ exponential moving averages (EMA) to smooth out short-term fluctuations:

$$m_l^{(t)} = \text{MomentumEstimate}(g_l^{(t)}, m_l^{(t-1)}, \beta_1) \quad (2)$$

where β_1 controls the retention of past gradients and m_l^{t-1} is the momentum output from last iteration. Subsequently, adaptive optimizers (*e.g.*, Adam and AdaGrad) compute parameter-specific rates using second-moment estimates:

$$\alpha_l^{(t)} = \text{LREstimate}(\alpha_l^{(t-1)}, m_l^{(t)}, \beta_2, \eta^{(t)}) \quad (3)$$

where η_t indicates the scheduled global learning rate at iteration t and β_1 is the decay rate similar to

Table 1: **Overview of Optimization Algorithms** from several aspects, including typical optimizers (Opt.), PEFT techniques, and optimizer wrapper. We take a neural network layer $W \in \mathbb{R}^{m\times n}$ $(m \leq n)$ as an example with LoRA rank $r \ll m$ and the cluster number $K \ll m$, where the weight and optimization states are considered. **(a)** As for optimization properties, we consider the cost of Adaptive Learning Rate (LR), where the extra state is used for its estimation (*e.g.*, second-order moment (2nd-moment), Non-negative Matrix Factorization (NMF), and SGG's cluster indices). **(b)** For LLM's low-rank property, we consider the way and cost of utilizing low-rank priors. **(c)** As for performance, we report the averaged PPL (%)↓ and Accuracy (%)↑ for C4 pre-training in Table 4 and GLUE SFT in Table 5 with the GPU memory in PyTorch implementation, where Adam (full-rank) is regraded as our baseline.

Type	Method	Adaptive LR	Basic State	Extra State	Low-Rank	Plugin	Extra Branch	C4↓	GLUE↑	GPU Memory
Classical Opt.	SGD	✗	Weight & Grad.	✗	✗	✗	✗	–	-0.50	$2mn$
Adaptive LR Opt.	Adam	Param-wise mn	Weight & Grad.	2nd-Moment mn	✗	✗	✗	23.36	86.24	$3mn$
Efficient Opt.	CAME	Param-wise mn	Weight & Grad.	NMF $2(m+n)$	NMF	✗	✗	**-1.64**	-0.85	$2mn+2(m+n)$
PEFT	LoRA	✗	Full-rank Grad.	✗	LoRA	✓	$r(m+n)$	+5.06	-0.31	$+3r(m+n)$
Opt. Wrapper	**SGG**	Group-wise K	Optimizer	Indices $(mn+K)$	Clustering	✓	✗	**-1.99**	**+1.00**	$+0$

Algorithm 1 Scaling with Gradient Grouping

Require: Parameters $\{\theta_l\}_{l=1}^{L}$, global learning rate η, optimizer momentum (β_1, β_2), learning objective $\mathcal{L}$, dataset $\mathcal{D}$, cluster number K, cluster indices $\mathcal{C}$, recluster interval T, scaling factor $\mathcal{S}$, scaling EMA decay β_3.
Ensure: Optimized model parameters θ.

Initialize:
  `RandomInit`$(\{\theta_l^0\}_{l=1}^{L})$ ▷ Model parameters
  $\{\alpha_l^0\}_{l=1}^{L} \leftarrow \eta$ ▷ Adaptive learning rates
  $\{\mathcal{C}_l\}_{l=1}^{L} \leftarrow \mathbf{0}$ ▷ Cluster assignment
  $\{\mathcal{S}_l\}_{l=1}^{L} \leftarrow \mathbf{1}$ ▷ Cluster scaling factor
for each iteration $t = 1, 2, \ldots$ **do**
  $\eta^t \leftarrow$ LearningRateScheduler(η, t)
  for each layer $l = 1, 2, \ldots, L$ **do**
    Gradient Computation
    $g_l^t \leftarrow \nabla_{\theta_l^{t-1}} \mathcal{L}(\theta^{t-1}, \mathcal{D})$
    Momentum Estimation
    $m_l^t \leftarrow$ MomentumEstimate$(g_l^t, m_l^{t-1}, \beta_1)$
    Learning Rate Adaptation
    $\alpha_l^t \leftarrow$ LREstimate$(\alpha_l^{t-1}, m_l^t, \beta_2, \eta^t)$
    if $t \mod T == 0$ **then** ▷ Re-clustering
      Assign Gradient Clusters
      $\mathcal{C}_l^t \leftarrow$ GradientCluster(m_l^t, K)
      Update Scaling Factors
      $\mathcal{S}_l^t \leftarrow$ ScaleUpdate$(\mathcal{C}_l^t, g_l^t, \beta_3)$
    end if
    Apply Cluster-Specific Scaling
    $\alpha_l^t \leftarrow \alpha_l^t \cdot \mathcal{S}_l^t[\mathcal{C}_l^t]$ ▷ Cluster-wise scaling
    Parameter Updates
    $\theta_l^t \leftarrow \theta_l^{t-1} - \alpha_l^t \cdot m_l^t$
  end for
end for

β_1. In contrast, non-adaptive methods use scheduled global rates here ($\alpha_l^t = \eta^t$). As aforementioned, this step faces the challenges of increasing memory overhead, especially for large models.

The final step applies the learning rate scaled momentum $\alpha_l^t \cdot m_l^t$ to parameter updates:

$$\theta_l^{(t)} = \theta_l^{(t-1)} - \alpha_l^{(t)} \cdot m_l^{(t)} \tag{4}$$

where the update rule is shared across optimizers, differing only in how the m_l^t and α_l^t are derived.

As an optimizer wrapper, our SGG operates on optimizer's internal states (m_l^t, α_l^t) without modifying the optimization pipeline. Thus, it allows seamless integration with any optimizer implementing the above steps, from SGD to Adam-mini (Zhang et al., 2024). In the following sections, we detail the clustering methods and scaling factor updates.

2.2 Gradient Grouping

As mentioned in Section 1, we suggest that LLM optimization requires a middle ground between per-parameter adaptation and global statistics. To validate this, we conduct empirical analysis of gradient statistics with LLaMA-1B pretraining on C4.

As shown in Figure 1, we obtain several observations: **(i)** Low-Rank Clustering: Both layer- and parameter-wise gradients form 2-4 dominant clusters per layer, with attention and MLP layers exhibiting distinct distributions. **(ii)** Outlier Impact: Less than 20% of parameter-wise gradients account for most of the distribution, which may negatively influence parameter-wise adaptation.

These findings demonstrate that existing per-parameter learning rate tuning methods could be redundant for LLM training, as a large number of model parameters exhibit negligible updates. Instead, treating gradients as more coarse-grained clusters, where parameters within a group share similar optimization dynamics, could potentially offer a promising path to reduce computation cost while improving robustness. Therefore, we design GradientCluster(m_l^t, K) in Algorithm 1, a dynamic grouping strategy that maps parameter-wise gradient of each layer to K clusters via mini-batch K-means. We also conduct experiments in Figure 2 to show the effectiveness of mini-batch K-means. By grouping gradients with similar characteristics, SGG reduces sensitivity to sparse gradients, enabling more stable learning rate adjustments.

2.3 Scaling Factor Estimation

Then, we design ScaleUpdate$(\mathcal{C}_l^t, g_l^t, \beta_3)$ to produce the accurate scaling factor S_l^t of gradient clusters at iteration t. It contains two sub-tasks: **(i)** presenting the variability and statistical properties of gradients with a robust measure; **(ii)** estimating and updating the scaling factor stability.

Table 2: **Analysis of Statistics** for group-wise gradiant estimation in SGG, where parameter-wise degenerated to Adam-like baseline. Validation Perplexity (PPL)↓ is reported with C4 pre-trained LLaMA architecture.

Statistic	Param-wise	Mean	Var.	Sign(Var.)	L2-norm	MAD
60M	34.06	30.68	31.17	30.75	30.89	**30.34**
1B	15.56	14.63	14.72	14.68	14.66	**14.56**

As for the measure of each gradient cluster, we investigate several commonly adopted statistics that reflect the significance of gradients, including Mean, Variance (like the second moment used in Adam), Sign of Variance (used in SignSGD (Bernstein et al., 2018)), and L2-norm (used in LARS (Ginsburg et al., 2018)) in Table 2. Since gradients within the same cluster should share similar structural properties, we propose *Median Absolute deviation* (MAD) as a robust estimation of the status difference of each group. For each cluster index $c \in K$ of l-th layer, $\text{MAD}_{l,c}^t$ can be computed as follows:

$$\text{MAD}_{l,c}^t = \text{median}\left(\left|m_l^t - \text{median}(m_l^t \cdot C_l^t[c])\right|\right), \tag{5}$$

where $\mathcal{C}_l^t[c]$ denotes the selected mask of the c-th cluster of the l-th layer, As shown in Table 2, MAD captures the statistical difference between the whole layer and each cluster, offering the best validation performance. Subsequently, the scaling factor $S_l[c]$ for cluster c can be computed as the inverse of MAD to ensure the group with higher gradient variability obtains smaller weights:

$$S_l^t[c] = \frac{1}{\text{MAD}_{l,c}^t + \epsilon}, \tag{6}$$

where $\epsilon = 10^{-8}$ is a small constant for numerical stability. As for stable updating, the scaling factors are updated periodically every T iterations using EMA update to smooth out short-term fluctuations:

$$S_l^t[c] = \beta_3 \cdot S_l^{t-1}[c] + (1-\beta_3) \cdot \frac{1}{\text{MAD}_{c,l}^t + \epsilon}, \tag{7}$$

where β_3 denotes EMA decay controlling the impact of history accumulation.

Table 3: **Performance Gains and Costs** of SGG. Validation Perplexity (PPL)↓, total training hours and the peak GPU memory are reported with LLaMA-1B on C4. GPU and CPU versions of SGG store the extra optimization state in GPU and CPU memory, respectively.

Method	PPL	Training Time	Memory
Adam	15.56	110h	7.8G
Adam+**SGG (GPU)**	**+6.5%** (-1.01)	**+1.8%** (+2h)	**+4.3G**
Adam+**SGG (CPU)**	**+6.5%** (-1.00)	**+8.2%** (+9h)	**+0.0G**

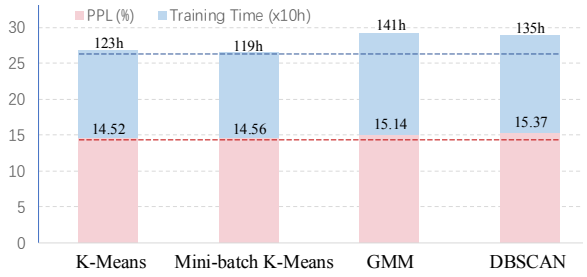


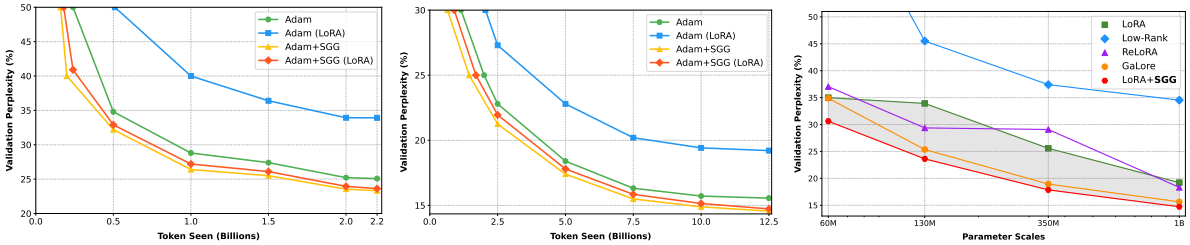
Figure 2: **Analysis of Clustering Strategies** in SGG with LLaMA-1B on C4. Validation Perplexity↓ and total pre-training hours↓ are reported, where Mini-batch K-Means achieves the best trade-off between PPL and the additional clustering costs (the lowest bar).

As for practical implementation, there are two trade-off problems between performances and efficiency in SGG. **(i)** How to choose clustering methods with a certain reclustering interval T? We consider four popular clustering methods, K-Means (MacQueen et al., 1967), Mini-batch K-Means (Sculley, 2010), GMM (Kambhatla and Leen, 1994), and DBSCAN (Ester et al., 1996), and choose the mini-batch K-Means because of its balance between performance and computional costs, as shown in Figure 2. We empirically design a flexible schedule that set the interval T as 10% of the total training iterations, as verified in Figure 5. **(ii)** Where to store the clustering indices $\{\mathcal{C}_l\}_{l=1}^L$ and scaling factors $\{\mathcal{S}_l\}_{l=1}^L$ and compute clustering? As shown in Table 3, we compare the performance, training time, and GPU memory of putting them on GPU or CPU. We found that the peak memory of computing GPU clustering is remarkable while storing $\{\mathcal{C}_l\}_{l=1}^L$ and $\{\mathcal{S}_l\}_{l=1}^L$ in the CPU memory will not delay the training. Therefore, we use the CPU version that performing clustering and storing extra optimization states on CPU, which does not require extra GPU memories and increase negligible training time, as summarized in Table 1.

3 Experiments

3.1 Experimental Setup

Task Information. To verify performance gains and generalizability of SGG, we choose 20 pub-



(a) LLaMA-130M Pre-training (b) LLaMA-1B Pre-training (c) Parameter Scaling-up

Figure 3: **C4 Pre-training** with LLaMA using full-rank and LoRA optimization methods. **(a)** and **(b)** show the fast convergence and performance gains over the baseline (Adam or LoRA) with SGG. **(c)** shows SGG yields consistent performance gains over LoRA, which orthogonally captures low-rank properties from gradients.

lic datasets for experiments, including large language datasets, Vision Question Answering (VQA) datasets, and standard multimodal large language model (MLLM) evaluation benchmarks. **(1)** Pre-training on C4: We utilized the en subset of the C4 dataset, a cleaned and high-quality version of Common Crawl's web corpus, which has been filtered for toxic content (Köpf et al., 2023). **(2)** SFT on GLUE Benchmark: We fine-tuned our model on the GLUE benchmark, a widely-used evaluation framework for NLP tasks such as sentiment analysis, question answering, and textual entailment (Wang, 2018). **(3)** PEFT with Commonsense Reasoning Tasks: Building on the LLM-Adapters framework (Hu et al., 2023), we evaluated PEFT methods on LLaMA across 8 Commonsense Reasoning (CS) datasets: BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC (ARC-Easy and ARC-Challenge) (Clark et al., 2018), and OBQA (Mihaylov et al., 2018). **(4)** RLHF with DPO: Using the TRL library, we implemented RLHF with DPO to align the model with human preferences. The Qwen2.5 0.5B model was trained on the *ultrafeedback_binarized* dataset, which includes binary preference labels (von Werra et al., 2020). **(5)** Comparison of MLLM Tasks: including VQA benchmarks such as GQA (Hudson and Manning, 2019), TextVQA (Singh et al., 2019), SciVQA^{I} (evaluation on the imageset of ScienceVQA) (Lu et al., 2022), VQAv2 (Goyal et al., 2017), and Vizwiz (Gurari et al., 2018), as well as MLLM evaluation benchmarks including POPE (Li et al.), MMBench (Liu et al., 2025), MMBench-Chinese (MMBench^{CN}) (Liu et al., 2025), SEED^{I} (Li et al., 2023), and MME (Perception) (Yin et al., 2023).

Implementation Details SGG is implemented in PyTorch, compatible with mainstream optimizers by several lines of code and no network modifications. Key hyperparameters include cluster number $K \in \{2, 3\}$, recluster interval T is 10% of total iterations, and scaling EMA decay $\beta_3 = 0.9$, empirically tuned for balancing both efficiency and performance. Clustering indices and scaling factors can also be stored in CPU memory to avoid GPU overhead, ensuring scalability. We reproduce results with gray and blue backgrounds, while other results are taken from their official papers. All experiments are conducted with A100-80G GPUs and reported with three runs.

Table 4: **Comparison Results of LLaMA Pre-training on C4** with full-rank and low-rank optimization. The validation Perplexity (%)↓ is reported for scaling-up parameters. **Bold** and **green** types denote the best results and performance gains compared to related baselines.

Method	Date	60M	130M	350M	1B
Pre-training with Full-Rank Optimizers					
Adam	ICLR'15	34.06	25.08	18.80	15.56
Adam-mini	ICML'24	34.10	24.85	19.05	16.07
LAMB	ICLR'20	33.04	24.37	18.26	15.84
LION	NeurIPS'23	32.42	24.05	18.10	15.47
Adam+**SGG**	**Ours**	**30.34**	**23.32**	**17.34**	**14.56**
Δ Gain		**-3.72**	**-1.76**	**-1.46**	**-1.00**
Pre-training with Memory-efficient Optimizers					
Adafactor	ICML'18	32.57	23.98	17.74	15.19
Low-Rank	arXiv'22	78.18	45.51	37.41	34.53
CAME	ACL'23	31.37	23.38	17.45	14.68
CAME+**SGG**	**Ours**	**30.15**	22.91	17.09	14.35
Δ Gain		**-1.22**	**-0.46**	**-0.36**	**-0.33**
APOLLO	MLSys'25	31.55	22.94	16.85	14.20
APOLLO+**SGG**	**Ours**	30.18	**22.52**	**16.54**	**13.95**
Δ Gain		**-1.37**	**-0.42**	**-0.31**	**-0.25**
Low-Rank Pre-training					
LoRA	ICLR'22	34.99	33.92	25.58	19.21
ReLoRA	ICLR'23	37.04	29.37	29.08	18.33
GaLore	ICML'24	34.88	25.36	18.95	15.64
LoRA+**SGG**	**Ours**	**30.62**	**23.62**	**17.86**	**14.73**
Δ Gain		**-4.37**	**-10.30**	**-7.72**	**-4.48**
Training Tokens		1.1B	2.2B	6.4B	13.1B

3.2 Comparison Results with LLMs

Pre-training on C4. Following GaLore (Zhao et al., 2024a), we employ LLaMA-based architec-

Table 5: **Comparison Results of GLUE Benchmark** with full-rank and low-rank (LoRA) SFT. Top-1 accuracy (%)↑ is reported for all sub-tasks, while **bold** and **green** types denote the best results and relative performance gains.

Optimizer	Rank	CoLA	STS-B	MRPC	RTE	SST2	MNLI	QNLI	QQP	Average
Full-Rank SFT										
SGD	Full	62.12	90.73	87.74	79.06	94.26	87.53	92.29	92.22	85.74
AdamW	Full	62.24	90.92	91.30	79.42	94.57	87.18	92.33	92.28	86.24
LAMB	Full	62.09	90.59	88.72	75.45	94.72	87.71	92.42	91.46	85.40
CAME	Full	62.16	90.43	89.02	75.94	94.61	87.13	92.31	91.54	85.39
APOLLO	Full	62.45	90.70	90.36	77.53	94.58	87.57	92.40	92.12	85.96
SGD+**SGG**	Full	**63.70** +1.58	90.92 +0.19	88.50 +0.76	79.42 +0.36	94.61 +0.35	87.97 +0.44	92.62 +0.33	92.67 +0.45	86.30 +0.56
AdamW+**SGG**	Full	63.36 +1.12	**91.22** +0.30	**92.65** +1.35	**80.87** +1.45	**95.58** +1.01	**88.32** +1.14	92.88 +0.55	**93.32** +1.04	**87.28** +1.00
LAMB+**SGG**	Full	62.47 +0.38	90.90 +0.31	89.46 +0.74	76.53 +1.08	94.95 +0.23	87.81 +0.10	**92.89** +0.47	91.78 +0.32	85.85 +0.45
Low-Rank SFT										
SGD (LoRA)	4	60.32	90.31	87.75	79.06	94.27	87.39	92.16	91.89	85.39
AdamW (LoRA)	4	61.38	90.57	91.07	78.70	92.89	86.82	92.18	91.29	85.61
LAMB (LoRA)	4	61.51	90.33	89.46	74.73	94.27	87.51	92.48	91.57	85.23
DoRA	4	60.38	90.50	88.24	74.73	93.69	–	92.59	–	–
GaLore (LoRA)	4	60.35	90.73	92.25	79.42	94.04	87.00	92.24	91.06	85.89
SGD+**SGG**	4	61.57 +1.25	90.50 +0.19	88.50 +0.75	80.14 +1.08	94.61 +0.37	87.65 +0.26	92.37 +0.21	92.24 +0.35	85.95 +0.56
AdamW+**SGG**	4	62.36 +0.98	**91.10** +0.53	92.12 +1.05	**80.51** +1.81	**95.06** +2.17	**88.18** +1.36	92.62 +0.44	**93.06** +1.77	**86.88** +1.27
LAMB+**SGG**	4	**62.47** +0.96	90.90 +0.57	89.46 +0.30	75.53 +0.80	94.95 +0.34	87.73 +0.12	**92.92** +0.41	91.78 +0.36	85.72 +0.49
SGD (LoRA)	8	60.57	90.29	88.48	79.42	94.32	87.44	92.23	92.10	85.61
AdamW (LoRA)	8	61.83	90.80	91.90	79.06	93.46	86.94	92.25	91.22	85.93
LAMB (LoRA)	8	61.89	90.78	89.21	79.42	94.61	87.61	92.51	91.42	85.35
DoRA	8	58.36	90.63	88.97	75.09	93.81	–	92.68	–	–
GaLore (LoRA)	8	60.06	90.82	92.01	79.78	94.38	87.17	92.20	91.11	85.94
SGD+**SGG**	8	61.61 +1.04	90.63 +0.34	89.46 +0.98	**80.87** +1.45	94.61 +0.29	87.93 +0.49	92.53 +0.30	92.48 +0.38	86.27 +0.66
AdamW+**SGG**	8	62.36 +0.53	**91.10** +0.30	**92.12** +0.22	80.51 +1.45	**95.06** +1.60	**88.17** +1.23	92.65 +0.40	**92.85** +1.63	**86.85** +0.92
LAMB+**SGG**	8	**62.47** +0.58	90.90 +0.12	89.46 +0.25	76.53 +1.80	94.95 +0.34	87.85 +0.24	**92.87** +0.36	91.78 +0.36	85.85 +0.50

tures for full-rank and low-rank pre-training on C4, maintaining consistent hyperparameters across model sizes while tuning learning rates individually. Experiments are conducted in BF16 format for memory efficiency, with learning rates calibrated within a fixed computational budget. Results show that integrating SGG with various optimizers consistently reduces validation perplexity (Table 4) and accelerates convergence (Figure 3), highlighting its efficacy as a plug-in optimization method. Further details are provided in appendix.

SFT on GLUE Benchmark. We fine-tuned the pre-trained RoBERTa-base model on GLUE tasks using SFT. Experimental results demonstrate that integrating SGG with various optimizers significantly boosts performance in both full-rank and low-rank SFT scenarios (Table 5). Notably, SGG consistently enhances accuracy across multiple GLUE tasks, with AdamW+SGG achieving particularly significant improvements. For instance, in the MRPC task, AdamW+SGG achieves gains of **+1.35%** (full-rank) and **+1.05%** (low-rank, rank 4). The MNLI task also shows substantial improvements, with increases of **+1.14%** (full-rank) and **+1.36%** (low-rank, rank 4). This plug-and-play approach demonstrates versatility and robustness across diverse rank constraints.

PEFT with Commonsense Reasoning Tasks. Following LLM-Adaptor, we evaluate eight CS tasks with top-1 accuracy (%) and GPU memory consumption, where LLaMA-7B is fine-tuned by AdamW on a unified training set with the LoRA rank $r = 32$ and evaluated with each sub-set. As shown in Table 6, SGG improves LoRA by **+2.9%** and outperforms or achieves competitive performances among classical PEFT baselines (Prefix (Li and Liang, 2021), Series (Houlsby et al., 2019), and Parallel (He et al., 2021)) and latest PEFT methods (DoRA, GaLore, and Fira (Chen et al., 2024)). View Table A3 and Appendix A for details.

Table 6: **Comparison Results of LLaMA PEFT** on Commonsense Reasoning datasets. The accuracy (%)↑ of six selected datasets and the average (AVG) results of eight datasets are reported, and Table A3 shows full results. **Bold** and **green** types denote the best results and performance gains over the baseline.

Method	BoolQ	PIQA	SIQA	WG	Arc-E	OBQA	AVG
Parallel	67.9	76.4	78.8	78.9	73.7	75.2	72.2
LoRA	68.9	80.7	77.4	78.8	77.8	74.8	74.7
DoRA	69.7	83.4	78.6	81.0	81.9	79.2	**78.4**
GaLore	69.5	82.0	75.1	18.0	80.7	78.0	62.7
Fira	69.4	82.6	78.0	**81.2**	**82.2**	**80.8**	76.9
LoRA+**SGG**	**70.3**	**83.6**	**78.8**	80.9	81.5	79.0	77.6
Δ Gain	+1.4	+2.9	+1.4	+2.1	+3.7	+4.2	+2.9

RLHF with DPO. Following the default TRL library settings, including optimization hyperparameters and the training pipeline, the effectiveness of DPO in improving preference alignment was systematically evaluated while maintaining computational efficiency. As shown in Table 8, the

Table 7: **Comparison of MLLM Tasks** with LLaVA variants and diverse optimizers. Top-1 accuracy (%)↑ is reported for selected tasks, AVG denotes the averaged results, and Table A6 shows full results. MMB and MMB^{CN} indicate MMbench and MMbench (Chinese).

Optimizer	Image Question Answering				Benchmarks			AVG
	GQA	VizWiz	SciVQA^{I}	VQA^{T}	MMB	MMB^{CN}	POPE	
BLIP-2	41.0	19.6	61.0	42.5	–	–	85.3	–
InstructBLIP	49.2	34.5	60.5	50.1	36.0	23.7	79.8	47.7
Qwen-VL	59.3	35.2	67.1	63.8	38.2	7.4	–	–
TinyLLaVA	62.0	–	69.1	59.1	66.9	–	86.4	–
MoE-LLaVA	62.6	–	**70.3**	57.0	68.0	–	85.7	–
LLaVA-Phi	–	–	68.4	48.6	59.8	–	85.0	–
LLaVA-NeXT	**64.2**	**57.6**	70.1	**64.9**	**67.4**	60.6	86.5	**67.3**
LLaVA-MOD	58.7	39.2	68.0	58.5	66.3	61.9	**87.0**	62.8
LLaVA-KD-2B	62.3	44.7	64.7	53.4	64.0	**63.7**	86.3	62.7
LLaVA-v1.5 Full-Rank SFT								
AdamW	62.0	50.0	66.8	58.2	64.3	58.3	85.9	63.6
Adafactor	62.7	48.2	70.7	57.1	66.1	60.4	86.0	64.5
LAMB	43.8	53.3	61.5	43.4	43.2	41.8	81.2	52.6
AdamW+**SGG**	62.4	50.1	68.8	**58.4**	65.6	59.9	**86.6**	64.5
Δ Gain	**+0.4**	**+0.1**	**+2.0**	**+0.2**	**+1.3**	**+1.6**	**+0.7**	**+0.9**
Adafactor+**SGG**	**62.8**	50.6	**71.6**	57.3	**66.3**	**60.8**	86.0	**65.1**
Δ Gain	**+0.1**	**+2.4**	**+0.9**	**+0.2**	**+0.2**	**+0.4**	**+0.0**	**+0.6**
LAMB+**SGG**	44.0	**53.3**	61.8	43.5	43.3	41.9	81.3	52.7
Δ Gain	**+0.2**	**+0.0**	**+0.3**	**+0.1**	**+0.1**	**+0.1**	**+0.1**	**+0.1**
LLaVA-v1.5 Low-Rank SFT (AdamW)								
LoRA	63.0	47.8	68.4	58.2	66.1	58.9	86.4	64.1
LoRA+**SGG**	**63.4**	**51.0**	**70.1**	**58.6**	**66.7**	**59.4**	**86.6**	**65.1**
Δ Gain	**+0.4**	**+2.2**	**+1.5**	**+0.4**	**+0.6**	**+0.5**	**+0.2**	**+1.0**
LLaVA-v1.5 8-bit Low-Rank SFT (AdamW)								
Q-LoRA	54.3	50.7	66.4	52.5	56.0	49.8	82.9	58.9
Q-LoRA+**SGG**	**55.1**	**51.3**	**66.7**	**53.0**	**56.1**	**51.0**	**83.4**	**59.5**
Δ Gain	**+0.8**	**+0.6**	**+0.3**	**+0.5**	**+0.1**	**+0.2**	**+0.5**	**+0.6**

combination of AdamW with SGG achieved the highest Top-1 accuracy of **72.02%** using LoRA training, demonstrating a significant gain of **1.80%** compared to the baseline.

3.3 Comparison Results with MLLMs

Following LLaVA-v1.5, the MLLM models include a pre-trained Vicuna-v1.5-7B (Chiang et al., 2023), a pre-trained 2×MLP, and a pre-trained CLIP (Radford et al., 2021), which is supervised fine-tuned for one epoch with a batch size of 64. As for the Full-Rank SFT, we consider AdamW, Adafactor, and LAMB as the baseline optimizers, whose details of optimizer hyperparameters and training settings are provided in Table A5, and also display results of mainstream MLLM methods. As shown in Table 7, results of VQA and benchmark tasks verify that SGG could improve AdamW by **+0.9%** with 64.5 average performance. With Adafactor, SGG could get extra **+0.6%** performance compared to the vanilla one, especially on VizWzi VQA task, SGG brings **+2.4%** capability. To further validate the effectiveness of SGG with PEFT and low-bit quantization scenarios, we

Table 8: **Comparison Results of DPO Tasks** with Qwen2.5-0.5B using full-rank and LoRA training. Top-1 accuracy (%)↑ is reported, while **bold** and **green** types denote the best results and gains compared to baselines.

Optimizer	Full-Rank		LoRA	
SGD	70.10		69.73	
AdamW	71.39		70.22	
LAMB	70.82		70.39	
SGD+**SGG**	70.82	**+0.72**	70.76	**+1.03**
AdamW+**SGG**	**71.85**	**+0.47**	**72.02**	**+1.80**
LAMB+**SGG**	71.32	**+0.50**	71.28	**+0.89**

conduct SFT with LoRA and 8-bit Quantization LoRA (Q-LoRA (Dettmers et al., 2024)) with the rank $r = 128$ and the scaling factor $\alpha = 256$. Table 7 shows that SGG achieves an average 65.1 score and brings additional performance gains over LoRA of **+2.2%** on the VizWiz task. With QLoRA (8-bit) SFT, SGG also brings **+0.6%** performance gain. Please refer to Table A6 for full results.

3.4 Analysis of Learning Rate Scaling

The relationship between batch size and learning rate (LR) is a critical focus in the LLM community due to its significant impact on training efficiency and stability. A key challenge is the surge phenomenon (Li et al., 2024b), where the optimal LR for Adam-like optimizers varies with batch size differently, complicating large-scale distributed training and requiring careful hyperparameter tuning to prevent instability or suboptimal convergence. Taking SFT on Alpaca (Taori et al., 2023) with Adam as an example, Figure 4 shows that SGG yields robust validation loss across varying batch sizes and learning rates, even with extremely large batch size of 4096 and LR of 1e-1. It suggests that SGG effectively mitigates gradient outliers and dynamically adjusts LRs, ensuring consistent and significant loss convergence within controlled batch size and LR configurations. Refer to Appendix C for details.

3.5 Ablation Study

We analyze SGG's clustering designs and hyperparameters from two aspects. Regarding cluster scaling, Table 9 shows that the cluster number K can be easily tuned in {2,3} for diverse tasks according to the warnings from mini-batch K-Means. The interval T can be robustly determined as 10% of total training iterations, *e.g.*, $T = 1k$ iterations for LLaMA-60M in Figure 5(a). For scaling decay β_3, Figure 5(b) verifies its choice as 0.9.

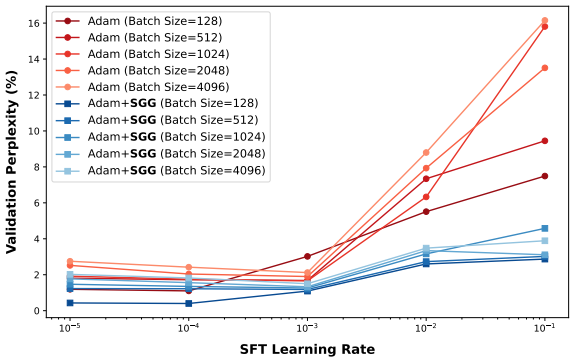


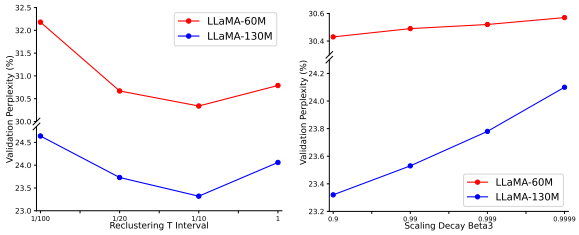
Figure 4: **Scaling-up Learning Rate** with diverse batch sizes with Qwen2.5-0.5B SFT on Alpaca. Validation loss↓ is plotted with Adam and Adam+SGG. SGG offers robust performance gains even under extreme settings.

Table 9: **Ablation Study** of the cluster number K (task-relevant) in SGG with various tasks, where ERR denotes the running error of mini-batch K-Means.

Task	Model	K=1 (Adam)	K=2	K=3	K=4
C4↓	LLaMA-60M	34.1	**30.3**	30.8	ERR
C4↓	LLaMA-130M	25.1	**23.3**	23.5	ERR
GLUE (MNLI)↑	RoBERT-Base	87.2	**88.3**	87.9	ERR
MLLM↑	LLaVA-1.5-7B	63.6	64.2	**64.5**	64.3

4 Related Work

Efficient Optimization. Adaptive learning rate optimizers (Loshchilov and Hutter, 2019) are widely used in LLM training for balancing convergence and generalization, yet they struggle to scale effectively due to their dependence on global gradient statistics, which overlook the heterogeneous nature of LLM gradients (Zhao et al., 2024b; Zhang et al., 2025b). This heterogeneity, coupled with the low-rank structure of LLM parameters, often leads to inefficient updates and suboptimal convergence (Chen et al., 2024; Zhao et al., 2024a). Traditional methods like Adam exhibit limitations in handling LLM gradient dynamics under low-rank constraints (Li et al., 2024a), prompting the development of memory-efficient optimizers such as BAdam (Luo et al., 2025) and LISA (Pan et al., 2025). Techniques like Adam-mini (Zhang et al., 2024) and APOLLO (Zhu et al., 2024a) further demonstrate that reduced learning rates or SGD-like memory footprints can achieve competitive performance. However, challenges persist, particularly in scaling optimization for large models, as evidenced by the surge phenomenon in optimal learning rate and batch size scaling (Li et al., 2024b). Innovations like SPAM (Huang et al., 2025) and CAME (Luo et al., 2023) introduce momentum reset and confidence-guided strategies to stabilize training. SGG addresses these issues by clustering gradients and applying cluster-specific scaling, ensuring tailored learning rates for each parameter group.



(a) Recluster Interval T (b) Scaling Decay β_3

Figure 5: **Ablation Study** of two robust hyper-parameters in SGG with LLaMA pre-training on C4.

Parameter-efficient Fine-tuning. LLM applications have advanced significantly through Parameter-efficient Fine-tuning (PEFT), with LoRA (Hu et al., 2021) emerging as a cornerstone technique. LoRA reduces the computational cost by learning fewer trainable parameters, which are low-rank perturbations to pre-trained weights. Recent extensions, such as DoRA variants (Liu et al., 2024c; Nasiri and Garraghan, 2025), further enhance adaptation efficiency while maintaining performance. However, LoRA and its variants face notable limitations: reliance on Dropout for regularization often fails in short training regimes (Kamalakara et al., 2022), suboptimal initialization slows convergence in sparse data scenarios, and static scaling factors hinder adaptive learning rate tuning (Dettmers et al., 2023). Efforts like LoRA+ (Hayou et al., 2024) and LoRA-XS (Bałazy et al., 2024) address some issues but remain challenged in multi-modality perception tasks (Ma et al., 2024) and broader PEFT applications (Zhang et al., 2025a). These limitations highlight the need for low-rank optimization that is migratable and adjusts learning rates with low-rank properties, which could be overcome with the gradient clustering and cluster-specific scaling techniques in our SGG.

5 Conclusion

This paper introduces an efficient optimizer wrapper SGG that captures the inherent low-rank properties of LLM structures to estimate group-wise adaptive learning rates through gradient clustering and cluster-specific scaling. Extensive experiments across LLM pre-training, SFT and DPO fine-tuning, and MLLM applications verify the consistent performance gains of SGG upon popular optimizers and LoRA with ignorable extra costs.

6 Limitations

Border Impact. As LLM applications extend, the efficient algorithm of resource-constrained and large-scale optimization grows vital. Our proposed SGG offers a new angle for developing efficient and migratable optimizers by employing gradient clustering to estimate adaptive learning rates rather than applying NMF in parameter or gradient statistic estimation like LoRA variants and memory-efficient optimization. It makes SGG highly versatile and integrates seamlessly with full-rank optimizers, memory-efficient optimizers, and LoRA algorithms to achieve significant performance gains across various LLM & MLLM applications with minimal extra cost.

Limitations and Future Works. While SGG demonstrates promising results in improving the efficiency and stability of LLM training, several limitations warrant further investigation. First, the current gradient grouping mechanism relies on heuristic clustering algorithms, which may not fully capture the nuanced gradient patterns in more complex architectures. Future work could explore more sophisticated clustering techniques or even learnable grouping strategies to enhance adaptability. Second, SGG's performance has been primarily validated on a limited set of benchmarks; extending evaluations to a broader range of tasks and model architectures would provide deeper insights into its generalizability. Lastly, the computational overhead of gradient clustering, though minimal, could be further optimized for extremely large-scale models. Addressing these limitations could pave the way for more robust and scalable optimization frameworks in the future.

References

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A frontier large vision-language model with versatile abilities. *arXiv preprint arXiv:2308.12966*.

Klaudia Bałazy, Mohammadreza Banaei, Karl Aberer, and Jacek Tabor. 2024. Lora-xs: Low-rank adaptation with extremely small number of parameters. *arXiv preprint arXiv:2405.17604*.

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. 2018. signsgd: compressed optimisation for non-convex problems. In *International Conference on Machine Learning*.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Yuxuan Cai, Jiangning Zhang, Haoyang He, Xinwei He, Ao Tong, Zhenye Gan, Chengjie Wang, and Xiang Bai. 2024. Llava-kd: A framework of distilling multimodal large language models. *arXiv preprint arXiv:2410.16236*.

Xi Chen, Kaituo Feng, Changsheng Li, Xunhao Lai, Xiangyu Yue, Ye Yuan, and Guoren Wang. 2024. Fira: Can we achieve full-rank training of llms under low-rank constraint? *arXiv preprint arXiv:2410.01623*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. 2023. Instructblip: Towards general-purpose vision-language models with instruction tuning. *arXiv preprint arXiv:2305.06500*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Knowledge Discovery and Data Mining*.

Boris Ginsburg, Igor Gitman, and Yang You. 2018. Large batch training of convolutional networks with layer-wise adaptive rate scaling. In *International Conference on Learning Representations (ICLR)*.

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. 2017. Making the v in vqa matter: Elevating the role of image understanding

in visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6904–6913.

Danna Gurari, Qing Li, Abigale J Stangl, Anhong Guo, Chi Lin, Kristen Grauman, Jiebo Luo, and Jeffrey P Bigham. 2018. Vizwiz grand challenge: Answering visual questions from blind people. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3608–3617.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. 2024. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *International Conference on Learning Representations (ICLR)*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. *ArXiv*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*.

Tianjin Huang, Ziquan Zhu, Gaojie Jin, Lu Liu, Zhangyang Wang, and Shiwei Liu. 2025. Spam: Spike-aware adam with momentum reset for stable llm training. *arXiv preprint arXiv:2501.06842*.

Drew A Hudson and Christopher D Manning. 2019. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6700–6709.

Siddhartha Rao Kamalakara, Acyr Locatelli, Bharat Venkitesh, Jimmy Ba, Yarin Gal, and Aidan N Gomez. 2022. Exploring low rank training of deep neural networks. *arXiv preprint arXiv:2209.13569*.

Nanda Kambhatla and Todd K. Leen. 1994. Classifying with gaussian mixtures and clusters. In *Advances in Neural Information Processing Systems (NeurIPS)*, page 681–688, Cambridge, MA, USA. MIT Press.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Andreas Köpf, Yannic Kilcher, Dimitri Von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, et al. 2023. Openassistant conversations-democratizing large language model alignment. *Advances in Neural Information Processing Systems*, 36:47669–47681.

Bohao Li, Rui Wang, Guangzhi Wang, Yuying Ge, Yixiao Ge, and Ying Shan. 2023. Seed-bench: Benchmarking multimodal llms with generative comprehension. *arXiv preprint arXiv:2307.16125*.

Guangyan Li, Yongqiang Tang, and Wensheng Zhang. 2024a. Lorap: Transformer sub-layers deserve differentiated structured compression for large language models. *arXiv preprint arXiv:2404.09695*.

Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. 2022. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International conference on machine learning*, pages 12888–12900. PMLR.

Shuaipeng Li, Penghao Zhao, Hailin Zhang, Xingwu Sun, Hao Wu, Dian Jiao, Weiyan Wang, Chengjun Liu, Zheng Fang, Jinbao Xue, et al. 2024b. Surge phenomenon in optimal learning rate and batch size scaling. *arXiv preprint arXiv:2405.14578*.

Siyuan Li, Juanxi Tian, Zedong Wang, Luyuan Zhang, Zicheng Liu, Weiyang Jin, Yang Liu, Baigui Sun, and Stan Z Li. 2024c. Unveiling the backbone-optimizer coupling bias in visual representation learning. *arXiv preprint arXiv:2410.06373*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pages 4582–4597.

Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Xin Zhao, and Ji-Rong Wen. Evaluating object hallucination in large vision-language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Bin Lin, Zhenyu Tang, Yang Ye, Jiaxi Cui, Bin Zhu, Peng Jin, Junwu Zhang, Munan Ning, and Li Yuan. 2024. Moe-llava: Mixture of experts for large vision-language models. *arXiv preprint arXiv:2401.15947*.

Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. 2024a. Llava-next: Improved reasoning, ocr, and world knowledge.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2024b. Visual instruction tuning. *Advances in neural information processing systems*, 36.

Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. 2020. Understanding the difficulty of training transformers. In *Conference on Empirical Methods in Natural Language Processing*.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024c. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. 2025. Mmbench: Is your multi-modal model an all-around player? In *European conference on computer vision*, pages 216–233. Springer.

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

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. 2022. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in Neural Information Processing Systems*, 35:2507–2521.

Qijun Luo, Hengxu Yu, and Xiao Li. 2025. Badam: A memory efficient full parameter optimization method for large language models. *Advances in Neural Information Processing Systems*, 37:24926–24958.

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

Feipeng Ma, Hongwei Xue, Guangting Wang, Yizhou Zhou, Fengyun Rao, Shilin Yan, Yueyi Zhang, Siying Wu, Mike Zheng Shou, and Xiaoyan Sun. 2024. Visual perception by large language model's weights. *arXiv preprint arXiv:2405.20339*.

James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

Hamid Nasiri and Peter Garraghan. 2025. Edora: Efficient weight-decomposed low-rank adaptation via singular value decomposition. *arXiv preprint arXiv:2501.12067*.

Rui Pan, Xiang Liu, Shizhe Diao, Renjie Pi, Jipeng Zhang, Chi Han, and Tong Zhang. 2025. Lisa: layer-wise importance sampling for memory-efficient large language model fine-tuning. *Advances in Neural Information Processing Systems*, 37:57018–57049.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*.

D. Sculley. 2010. Web-scale k-means clustering. In *International Conference on World Wide Web*.

Noam M. Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. *ArXiv*, abs/1804.04235.

Fangxun Shu, Yue Liao, Le Zhuo, Chenning Xu, Lei Zhang, Guanghao Zhang, Haonan Shi, Long Chen, Tao Zhong, Wanggui He, et al. 2024. Llava-mod: Making llava tiny via moe knowledge distillation. *arXiv preprint arXiv:2408.15881*.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. 2019. Towards vqa models that can read. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8317–8326.

Naresh K. Sinha and Michael P. Griscik. 1971. A stochastic approximation method. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-1(4):338–344.

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.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. 2020. Trl: Transformer reinforcement learning. https://github.com/huggingface/trl.

Alex Wang. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Qinghao Ye, Haiyang Xu, Jiabo Ye, Ming Yan, Anwen Hu, Haowei Liu, Qi Qian, Ji Zhang, and Fei Huang. 2024. mplug-owl2: Revolutionizing multi-modal large language model with modality collaboration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13040–13051.

Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. 2023. A survey on multimodal large language models. *arXiv preprint arXiv:2306.13549*.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2020. Large batch optimization for deep learning: Training BERT in 76 minutes. In *International Conference on Learning Representations (ICLR)*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Dan Zhang, Tao Feng, Lilong Xue, Yuandong Wang, Yuxiao Dong, and Jie Tang. 2025a. Parameter-efficient fine-tuning for foundation models. *arXiv preprint arXiv:2501.13787*.

Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhiquan Luo. 2025b. Why transformers need adam: A hessian perspective. *Advances in Neural Information Processing Systems*, 37:131786–131823.

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

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

Rosie Zhao, Depen Morwani, David Brandfonbrener, Nikhil Vyas, and Sham Kakade. 2024b. Deconstructing what makes a good optimizer for language models. *arXiv preprint arXiv:2407.07972*.

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

Hanqing Zhu, Zhenyu Zhang, Wenyan Cong, Xi Liu, Sem Park, Vikas Chandra, Bo Long, David Z Pan, Zhangyang Wang, and Jinwon Lee. 2024a. Apollo: Sgd-like memory, adamw-level performance. *arXiv preprint arXiv:2412.05270*.

Yichen Zhu, Minjie Zhu, Ning Liu, Zhiyuan Xu, and Yaxin Peng. 2024b. Llava-phi: Efficient multi-modal assistant with small language model. In *Proceedings of the 1st International Workshop on Efficient Multimedia Computing under Limited*, pages 18–22.

Appendix

A Implementation Details

SGG is implemented in PyTorch and seamlessly integrates with mainstream optimizers such as SGD and Adam, requiring no modifications to the network architecture and only minimal changes to the optimization loop. Key hyperparameters include the number of clusters $K \in \{2, 3\}$, the recluster interval T (typically set to 10% of the total training iterations), and the scaling factor EMA decay $\beta_3 = 0.9$. These hyperparameters are empirically tuned to balance computational efficiency and optimization performance. To ensure scalability, clustering indices and scaling factors are stored in CPU memory, reducing GPU memory overhead while maintaining efficient access during training. This design choice allows SGG to handle large-scale models without significant memory bottlenecks.

The core of SGG involves grouping gradients into clusters and applying cluster-specific scaling factors to the learning rates. For optimizers like **Adam**, the momentum estimates m_l^t (which provide a smoothed representation of the gradients) are flattened and clustered instead. The clustering is performed using the `MiniBatchKMeans` algorithm from the `sklearn` library, which is efficient and suitable for large datasets. During clustering, the flattened gradients or momentum estimates are reshaped into a 2D array of shape $(N, 1)$, where N is the total number of elements in the gradient tensor. After clustering, each gradient element is assigned to a cluster, and the scaling factors $\mathcal{S}_l$ are updated using an EMA of the median gradient magnitudes within each cluster. These scaling factors are then applied to the learning rates during the parameter update step, enabling adaptive and cluster-specific optimization. The entire process is computationally efficient, with clustering performed on the CPU and only the final scaling factors transferred to the GPU for parameter updates.

B Evaluation Setups and Experimental Results

B.1 LLM Pre-training on C4

We conducted extensive pre-training experiments on LLaMA-based large language models using the C4 dataset. The C4 dataset, a meticulously cleaned and processed version of Common Crawl's web corpus, serves as a benchmark for pre-training language models and learning word representations. To closely replicate real-world pre-training conditions, we implemented a no-repetition training protocol over a substantial data volume, scaling our experiments across model sizes up to 7 billion parameters. We provide a comprehensive overview of the LLaMA architecture and the specific hyperparameters employed during pre-training (Tabel A1). The hyperparameters are standardized across all model sizes, with a maximum sequence length of 256 tokens and a batch size of 131,000 tokens. Across all experiments, we implemented a learning rate warmup phase for the initial 10% of the training steps, followed by a cosine annealing schedule that gradually reduces the learning rate to 10% of its initial value.

For each model size (ranging from 60 million to 1 billion parameters), we performed a systematic hyperparameter search to identify the optimal learning rate from the set $\{0.01, 0.005, 0.001, 0.0005, 0.0001\}$, with selection criteria based on validation perplexity. Notably, SGG demonstrated remarkable robustness to hyperparameter variations, maintaining stable performance across different model sizes with a consistent learning rate.

B.2 LLM SFT on GLUE Benchmark

The GLUE benchmark, a widely-used evaluation framework for NLP tasks such as sentiment analysis, question answering, and textual entailment (Wang, 2018), serves as a robust platform for assessing model performance. In this study, we fine-tuned the pre-trained RoBERTa-Base model on the GLUE benchmark using the Hugging Face implementation. The model was trained for 30 epochs with a batch size of 16 for all tasks, except for CoLA, which utilized a batch size of 32. We meticulously tuned the learning rate and scale factor for the SGG optimization technique. Table A2 details the hyperparameters employed for fine-tuning RoBERTa-Base with SGG.

The results, as presented in Table 5, demonstrate the efficacy of SGG in enhancing model performance across various GLUE sub-tasks. Notably, SGG consistently improves the top-1 accuracy when applied to different optimizers, including SGD, AdamW, and LAMB. For instance, SGD+SGG achieves a significant performance gain of **+1.58** on CoLA and **+0.76** on MRPC compared to the standard SGD optimizer. Similarly, AdamW+SGG shows remarkable improvements, with gains of **+1.45** on RTE and **+1.01** on SST2.

Table A1: Hyperparameters of LLaMA models for evaluation.

Params	Hidden	Intermediate	Heads	Layers	Steps	Data Amount
60M	512	1376	8	8	10K	1.3 B
130M	768	2048	12	12	20K	2.6 B
350M	1024	2736	16	24	60K	7.8 B
1 B	2048	5461	24	32	100K	13.1 B

Table A2: Hyperparameters of fine-tuning RoBERTa base.

	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
Batch Size	16	16	16	32	16	16	16	16
# Epochs	30	30	30	30	30	30	30	30
Learning Rate	2E-05	1E-05	3E-05	3E-05	1E-05	1E-05	1E-05	2E-05
Rank Config.	Full							
Max Seq. Len.	512							
Batch Size	16	16	16	32	16	16	16	16
# Epochs	30	30	30	30	30	30	30	30
Learning Rate	2E-05	1E-05	3E-05	3E-05	1E-05	1E-05	1E-05	2E-05
Rank Config.	$r = 4$							
Max Seq. Len.	512							
Batch Size	16	16	16	32	16	16	16	16
# Epochs	30	30	30	30	30	30	30	30
Learning Rate	2E-05	2E-05	2E-05	3E-05	1E-05	2E-05	2E-05	3E-05
Rank Config.	$r = 8$							
Max Seq. Len.	512							

These enhancements underscore the advantage of SGG in stabilizing and accelerating the convergence of gradient-based optimization methods, particularly in low-rank settings where computational efficiency is crucial. The consistent performance gains across multiple tasks and optimizers highlight SGG's potential as a robust technique for fine-tuning large-scale language models, making it a valuable addition to the NLP toolkit.

B.3 LLM PEFT with Commonsense Reasoning Tasks

Following LLM-Adaptor (Hu et al., 2023), we evaluate eight Commonsense Reasoning tasks with top-1 accuracy (%) and GPU memory consumption, including BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-Easy (ARC-E) and ARC-Challenge (ARC-C) (Clark et al., 2018), and OBQA (Mihaylov et al., 2018). As SFT setups in LLM-Adaptor, we combine the training datasets from all sub-tasks to fine-tune the pre-trained LLaMA-7B for 3 epochs using AdamW optimizer with a basic learning rate of 1e-4, a batch size of 32, the rank $r = 32$. Then, we evaluate each sub-task individually using its respective testing dataset. Three classical PEFT baselines, Prefix-tuning (Prefix) (Li and Liang, 2021), Series Adapter (Series) (Houlsby et al., 2019), and Parallel Adapter (Parallel) (He et al., 2021), and three popular PEFT methods, DoRA (Liu et al., 2024c), GaLore (Zhao et al., 2024a), and Fira (Chen et al., 2024), are compared in Table A3. Our SGG consisently improves eight sub-tasks over LoRA by **+2.9%** without extra GPU memory, achieving competitive performances with well-designed PEFT methods with LoRA+SGG.

B.4 LLM RLHF with DPO

In our experiments, we employed the Direct Preference Optimization (DPO) approach to fine-tune the Qwen2.5-0.5B model using the *ultrafeedback_binarized* dataset, which contains binary preference labels that facilitate the alignment of the model with human preferences (von Werra et al., 2020). The training process was conducted using both full-rank and LoRA strategies, with the latter being particularly effective in reducing the number of trainable parameters while maintaining competitive performance. The hyperparameters for the training included a learning rate of 5.0×10^{-7} for full-rank training and 5.0×10^{-6} for LoRA, with a single training epoch and a batch size of 2 per de-

Table A3: **Full Comparison Results of LLaMA PEFT** on eight commonsense reasoning datasets with the accuracy (%)↑ and the GPU memory↓, where only the weights and optimization states are considered. ChatGPT results are obtained by Zero-shot CoT with gpt-3.5-turbo API. **Bold** and **green** types denote the best results and performance gains compared to related baselines.

Model	PEFT	Memory	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	Arc-E	Arc-C	OBQA	Average
ChatGPT	–	–	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
LLaMA-7B	Prefix	0.05G	64.3	76.8	73.9	42.1	72.1	72.9	54.0	60.6	64.6
	Series	0.42G	63.0	79.2	76.3	67.9	75.7	74.5	57.1	72.4	70.8
	Parallel	1.49G	67.9	76.4	78.8	69.8	78.9	73.7	57.3	75.2	72.2
	LoRA	0.35G	68.9	80.7	77.4	78.1	78.8	77.8	61.3	74.8	74.7
	DoRA	0.26G	69.7	83.4	78.6	**87.2**	81.0	81.9	**66.2**	79.2	**78.4**
	GaLore	0.26G	69.5	82.0	75.1	32.2	18.0	80.7	65.8	78.0	62.7
	Fira	0.26G	69.4	82.6	78.0	76.8	**81.2**	**82.2**	64.4	**80.8**	76.9
	LoRA+**SGG**	0.35G	**70.3**	**83.6**	**78.8**	81.7	80.9	81.5	65.3	79.0	77.6
	Δ *Gain*	+0.00	**+1.4**	**+2.9**	**+1.4**	**+3.6**	**+2.1**	**+3.7**	**+4.0**	**+4.2**	**+2.9**

vice. Gradient accumulation was set to 8 steps, and gradient checkpointing was enabled to optimize memory usage.

The optimization process utilized several optimizers, including SGD, AdamW, and LAMB, with and without the addition of the SGG (Stochastic Gradient with Gain) technique. As shown in Table 8, the inclusion of SGG consistently improved the Top-1 accuracy across all optimizers. For instance, AdamW with SGG achieved a Top-1 accuracy of 71.85% in full-rank training, representing a gain of **0.47%** over the baseline AdamW. Similarly, in LoRA training, AdamW with SGG reached 72.02%, a significant improvement of **1.80%** compared to the baseline. These results underscore the advantage of SGG in enhancing the optimization process, particularly in scenarios where computational efficiency and model performance are critical.

The LoRA configuration used a rank (r) of 32 and alpha (α) of 16, which provided a balance between model complexity and performance. The evaluation strategy was set to steps, with evaluations conducted every 50 steps, and logging was performed every 25 steps to monitor the training progress. The output directory was designated as `Qwen2-0.5B-DPO`, and the `no_remove_unused_columns` flag was enabled to retain all columns in the dataset during training.

B.5 MLLM SFT with LLaVA Variants

To validate the generalization capability of the SGG-equipped optimizer, we also verify it on some variants of LLaVA (Liu et al., 2024b). *i.e.* LLaVA-v1.5-7b, LLaVA-LoRA, LLaVA-v1.3. And we choose some mainstream multi-modal LLMs at Table 7, *e.g.* BLIP (Li et al., 2022), InstructBLIP (Dai et al., 2023), Qwen-VL (Bai et al., 2023), Qwen-VL-Chat, mPLUG-Owl2 (Ye et al., 2024), and some variant of LLaVA, Tiny-LLaVA (Zhou et al., 2024), MoE-LLaVA (Lin et al., 2024), LLaVA-Phi (Zhu et al., 2024b), LLaVA-NeXT (Liu et al., 2024a), LLaVA-MOD (Shu et al., 2024), and LLaVA-KD-2B (Cai et al., 2024).

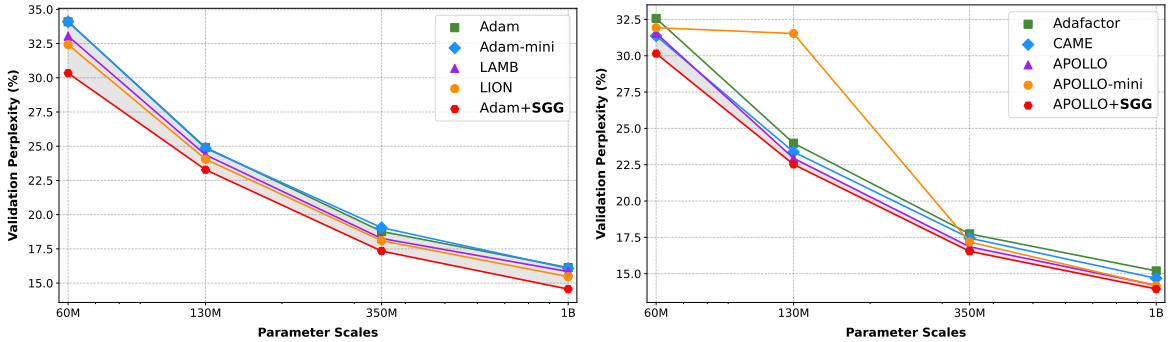
Setup and Settings: Following the LLaVA-v1.5, we use a pre-trained Vicuna-v1.5-7B (Chiang et al., 2023) as the language decoder. A pre-trained 2×MLP as the connector to align the visual tokens to text tokens. The connector was trained by the `LCS-558K` datasets for one epoch. For the visual encoder, CLIP (Radford et al., 2021) encodes and extracts the visual representation from the images. In our experiments, we validate three different optimizers: AdamW, Adafactor, and LAMB. The details of the optimizer hyperparameters and some training settings are shown in Table A5.

Supervised Fine-tuning: We keep the visual encoder frozen and update the parameters of the connector and LLM for training. For the Full-Rank Supervised Fine-Tuning (SFT), the learning rate was set to 2e-5, the batch size was 64, and training one epoch on `llava-v1.5-mix665k` dataset. To further validate the effectiveness of SGG in the light parameters and low-bit quantization scenario, we conducted an experiment to train the Low-Rank (LoRA) and 8-bit Quantization LoRA (Q-LoRA (Dettmers et al., 2024)) SFT method. These methods have unique advantages in parameter efficiency and training speed. For the LoRA and Q-LoRA SFT, the rank r of LoRA is 128, the learning rate scaling factor α is 256, the batch size set is 64, and training one epoch. These low-rank methods are based on the LLaVA-v1.5.

Results: Table 7 and Table A6 present the re-

Table A4: **Full Comparison Results of LLaMA Pre-training on C4** using full-rank and memory-efficient optimization with model size ranging from 60M to 1B. The validation perplexity (PPL)↓ and GPU memory (Mem.)↓ are reported, where only the weights and optimization states are considered for the memory. **Bold** and **green** types denote the best results and performance gains compared to related baselines.

Method	Date	60M		130M		350M		1B	
		PPL	Mem.	PPL	Mem.	PPL	Mem.	PPL	Mem.
Adam	ICLR'25	34.06	0.36G	25.08	0.76G	18.80	2.06G	15.56	7.80G
Adam-mini	ICML'24	34.10	0.23G	24.85	0.48G	19.05	1.32G	16.07	4.75G
LAMB	ICLR'20	33.04	0.36G	24.37	0.77G	18.26	2.07G	15.84	7.81G
LION	NIPS'23	32.42	0.36G	24.05	0.75G	18.10	2.05G	15.47	7.73G
Adam+**SGG**	**Ours**	**30.34**	0.36G	**23.32**	0.76G	**17.34**	2.06G	**14.56**	7.80G
Δ *Gain*		**-3.72**	+0.00	**-1.76**	+0.00	**-1.46**	+0.00	**-1.00**	+0.00
Adafactor	ICML'18	32.57	0.24G	23.98	0.61G	17.74	1.53G	15.19	6.65G
Low-Rank	arXiv'22	78.18	0.26G	45.51	0.54G	37.41	1.08G	34.53	3.57G
CAME	ACL'23	31.37	0.25G	23.38	0.62G	17.45	1.55G	14.68	6.70G
CAME+**SGG**	**Ours**	**30.15**	0.25G	22.91	0.62G	17.09	1.55G	14.35	6.70G
Δ *Gain*		**-1.22**	+0.00	**-0.46**	+0.00	**-0.36**	+0.00	**-0.33**	+0.00
APOLLO	MLSys'25	31.55	0.24G	22.94	0.52G	16.85	1.22G	14.20	4.38G
APOLLO+**SGG**	**Ours**	30.18	0.24G	**22.52**	0.52G	**16.54**	1.22G	**13.95**	4.38G
Δ *Gain*		**-1.37**	+0.00	**-0.42**	+0.00	**-0.31**	+0.00	**-0.25**	+0.00
LoRA	ICLR'22	34.99	0.36G	33.92	0.80G	25.58	1.76G	19.21	6.17G
ReLoRA	ICLR'23	37.04	0.36G	29.37	0.80G	29.08	1.76G	18.33	6.17G
GaLore	ICML'24	34.88	0.24G	25.36	0.52G	18.95	1.22G	15.64	4.38G
GaLore+SPAM	ICLR'25	32.39	0.24G	23.98	0.52G	18.28	1.22G	14.73	6.17G
LoRA+**SGG**	**Ours**	**30.62**	0.36G	**23.62**	0.80G	**17.86**	1.76G	**14.73**	6.17G
Δ *Gain*		**-4.37**	+0.00	**-10.30**	+0.00	**-7.72**	+0.00	**-4.48**	+0.00
Training Tokens		1.1B		2.2B		6.4B		13.1B	



(a) Full-Rank Optimizers (b) Memory-efficient Optimizers

Figure A1: **Parameter Scaling-up** on C4 pre-training with various optimization methods.

sults of SGG on VQA and benchmark tasks. Table 7 shows the results of seven representative tasks, while Table A6 displays the full results of nine tasks. For the Full-Rank SFT, on the AdamW optimizer, SGG achieves 64.5 average performance on the 7 different tasks, which brings **+0.9%** performance compared to the AdamW baseline. On the Adafactor, SGG could get extra **+0.6%** performance compared to the vanilla Adafactor, especially on VizWzi VQA task, SGG could bring **+2.4%** capability. With LAMB+SGG, our performance can reach 52.7. For the LoRA SFT, our SGG could achieve 65.1 scores, and on the VizWiz task, it brings additional performance gains of **+2.2%**. For the 8-bit experiments, Table 7 shows that our SGG with AdamW could also bring some performance.

C Empirical Analysis

C.1 Analysis of Gradient Clustering

Figure 1 illustrates the gradient clustering phenomenon observed during the pre-training of the LLaMA-1B model on the C4 dataset, focusing on gradients, adaptive learning rates, and gradient norms. LLMs exhibit unique gradient dynamics due to their massive scale, sparse activations, and hierarchical structure. SGG leverages these characteristics to improve optimization efficiency and convergence. Gradients in LLMs often follow a heavy-tailed distribution, with a small fraction of

Table A5: Details of the hyperparameters for the optimizers and experiment settings.

Method	AdamW	Adafactor	LAMB
Modules and datasets			
LLM	Vicuan-v1.5-7B		
Vision encoder	CLIP-L-336px		
Connector	2×MLP		
Pretrain data	`LCS-558K`		
SFT data	`llava-v1.5-mix665k`		
Basic SFT settings			
Learning rate	$2e^{-5}$	$2e^{-5}$	$2e^{-5}$
Batch size	64	64	64
Betas	(0.9, 0.999)	✗	(0.9, 0.999)
Epsilon	$1e^{-8}$	($1e^{-30}$, $1e^{-3}$)	$1e^{-6}$
Weight decay	✗	✗	✗
LR scheduler	Cosine	Cosine	Cosine
Warmup ratio	0.03	0.03	0.03
Clip threshold	✗	1.0	✗
Clamp value	✗	✗	10
Cluster number	3	3	2
Recluster interval	1,000	1,000	1,000
Decay rate	(0.95, 0.9)	(0.95, 0.9)	(0.95, 0.9)
Low-Rank hyperparameters			
LoRA (r=128, α=256)	✓	✗	✗
8bit LoRA (r=128, α=256)	✓	✗	✗

parameters contributing disproportionately to the overall gradient magnitude. SGG addresses this by flattening gradients into high-dimensional vectors and applying clustering algorithms (*e.g.,* k-means) to group parameters with similar behaviors. This results in two distinct clusters: one for parameters with large gradients (associated with salient features or rare tokens) and another for those with smaller gradients (associated with frequent but less informative tokens). Adaptive learning rates are then computed separately for each cluster, ensuring stability for parameters with large gradients and faster convergence for those with smaller gradients. This contrasts with baseline methods that apply uniform learning rates, failing to account for the heavy-tailed gradient distributions typical of LLMs.

Figure 3(c) depicts the layer-wise L_2-gradient norm distributions across all layers of the LLaMA-1B model. Gradient norms vary significantly across layers due to the hierarchical nature of LLMs. Earlier layers (*e.g.,* embedding and low-level transformer layers) exhibit smaller gradient norms, as they focus on general syntactic and semantic patterns. In contrast, deeper layers (*e.g.,* higher-level transformer layers) tend to have larger gradient norms, as they model complex, context-dependent relationships. SGG captures these patterns by grouping parameters based on gradient norms and applying layer-wise learning rate scaling. This ensures earlier layers receive larger updates for faster learning of general patterns, while deeper layers receive smaller updates to maintain stability and prevent overfitting. Baseline methods, which lack such adaptive scaling, often struggle to optimize all layers simultaneously, leading to suboptimal convergence and poor generalization.

The clustering of gradients, adaptive learning rates, and gradient norms in LLMs are deeply interconnected phenomena. The heavy-tailed gradient distribution directly influences adaptive learning rates, as parameters with large gradients are assigned smaller learning rates to prevent instability. This, in turn, affects gradient norms, as learning rate scaling impacts the magnitude of parameter updates. SGG's ability to capture these relationships and adaptively scale learning rates based on gradient clustering and norm distributions leads to more stable and efficient optimization compared to baseline methods. Furthermore, the hierarchical structure of LLMs introduces additional complexity, as different layers exhibit distinct gradient behaviors. SGG addresses this by leveraging layer-wise clustering and scaling, ensuring each layer is optimized according to its specific role. This is particularly critical for LLMs, where the interplay between low-level and high-level features is essential for capturing the nuances of natural language. By preserving the inherent structure of the optimization landscape, SGG not only improves convergence but also enhances the model's ability to generalize to unseen data.

C.2 Analysis of Learning Rate Scaling

We analyze the impact of learning rate scaling on the validation perplexity of the Qwen2.5-0.5B model fine-tuned on the Alpaca dataset. The experiments were conducted with varying batch sizes {128, 512, 1024, 2048, 4096} and learning rates {1e-1, 1e-2, 1e-3, 1e-4, 1e-5}, using both the Adam optimizer and Adam with SGG. The model was trained for 3 epochs with LoRA (rank=8) and followed the official settings of the Alpaca framework. The results, as depicted in Figure 4, demonstrate several key trends. First, as the batch size increases, the validation perplexity generally decreases, indicating that larger batch sizes contribute to more stable and efficient training. This effect is particularly pronounced when SGG is applied, suggesting that SGG enhances the model's ability to generalize even under extreme batch size settings. Second, lower learning rates (*e.g.,* 1e-4, 1e-5) consistently yield better performance, especially when

Table A6: **Full Comparison Results with Mainstream MLLMs**. Compared with counterparts. Top-1 accuracy (%) is reported. AVG: The average of the nine benchmarks for comprehensive comparison except for MME. †: reproduced results using the official code. Lots of the results are reported from LLaVA-KD (Cai et al., 2024).

Method	LLM	Optimizer	Image Question Answering					Benchmarks					AVG
			VQAv2	GQA	VizWiz	SciVQA^{I}	TextVQA	MME	MMBench	MMBench^{CN}	POPE	SEED^{I}	
BLIP-2	Vicuna-13B	AdamW	65.0	41.0	19.6	61.0	42.5	–	–	–	85.3	–	–
InstructBLIP	Vicuna-7B	AdamW	–	49.2	34.5	60.5	50.1	–	36.0	23.7	79.8	–	–
Qwen-VL	Qwen-7B	AdamW	78.8	59.3	35.2	67.1	63.8	–	38.2	7.4	–	–	–
Qwen-VL-Chat	Qwen-7B	AdamW	78.2	57.5	38.9	68.2	61.5	–	60.6	56.7	–	–	–
mPLUG-Owl2	LLaMA2-7B	AdamW	79.4	56.1	54.5	68.7	54.3	–	66.5	–	85.8	–	–
TinyLLaVA†	Qwen1.5-4B	AdamW	79.9	63.4	46.3	72.9	59.0	–	67.9	67.1	85.2	–	–
TinyLLaVA	Phi2-2.7B	AdamW	79.9	62.0	–	69.1	59.1	–	66.9	–	86.4	–	–
Bunny	Phi2-2.7B	AdamW	79.8	62.5	43.8	70.9	56.7	–	68.6	37.2	–	–	–
Imp-3B	Phi2-2.7B	AdamW	–	63.5	54.1	72.8	59.8	–	72.9	46.7	–	–	–
MobileVLM	MLLaMA-2.7B	AdamW	–	59.0	–	61.0	47.5	–	59.6	–	84.9	–	–
MobileVLMv2	MLLaMA-2.7B	AdamW	–	61.1	–	70.0	57.5	–	63.2	–	84.7	–	–
MoE-LLaVA	Phi2-2.7B	AdamW	79.9	62.6	–	70.3	57.0	–	68.0	–	85.7	–	–
LLaVA-Phi	Phi2-2.7B	AdamW	71.4	–	–	68.4	48.6	–	59.8	–	85.0	–	–
LLaVA-NeXT	Vicuna-1.5-7B	AdamW	81.8	64.2	57.6	70.1	64.9	1519.0	67.4	60.6	86.5	70.2	69.3
LLaVA-NeXT	Vicuna-1.5-13B	AdamW	82.8	65.4	60.5	73.6	67.1	1575.0	70.0	64.4	86.2	71.9	71.3
MiniCPM-V	MiniCPM-2.4B	AdamW	–	51.5	50.5	74.4	56.6	–	64.0	62.7	79.5	–	–
MiniCPMv2	MiniCPM-2.4B	AdamW	–	52.1	60.2	76.3	73.2	–	68.5	67.2	86.3	–	–
LLaVA-MOD	Qwen1.5-1.8B	AdamW	–	58.7	39.2	68.0	58.5	–	66.3	61.9	87.0	–	–
LLaVA-KD-2B	Qwen1.5-1.8B	AdamW	79.0	62.3	44.7	64.7	53.4	–	64.0	63.7	86.3	–	–
LLaVA-v1.5/1.6 Full-Rank SFT													
LLaVA-v1.5	Vicuna-1.5-7B	AdamW	78.5	62.0	50.0	66.8	58.2	1510.7	64.3	58.3	85.9	66.2	65.6
LLaVA-v1.5	Vicuna-1.5-7B	Adafactor	–	62.7	48.2	70.7	57.1	1462.5	66.1	60.4	86.0	66.8	–
LLaVA-v1.5	Vicuna-1.5-7B	LAMB	–	43.8	53.3	61.5	43.4	1090.9	43.2	41.8	81.2	50.4	–
LLaVA-v1.5	Vicuna-1.5-7B	AdamW+**SGG**	79.2	62.4	50.1	68.8	58.4	1526.3	65.6	59.9	86.6	66.4	66.4
Δ gain compared to AdamW			**+0.7**	**+0.4**	**+0.2**	**+2.0**	**+0.2**	**+15.6**	**+1.3**	**+1.6**	**+0.7**	**+0.2**	**+0.8**
LLaVA-v1.5	Vicuna-1.5-7B	Adafactor+**SGG**	–	62.8	50.6	71.6	57.3	1477.2	66.3	60.8	86.0	67.3	–
Δ gain compared to Adafactor			–	**+0.1**	**+2.4**	**+0.9**	**+0.2**	**+14.7**	**+0.2**	**+0.4**	**+0.0**	**+0.5**	–
LLaVA-v1.5	Vicuna-1.5-7B	LAMB+**SGG**	–	44.0	53.3	61.8	43.5	1122.9	43.3	41.9	81.3	50.5	–
Δ gain compared to LAMB			–	**+0.2**	**+0.0**	**+0.3**	**+0.1**	**+32.0**	**+0.1**	**+0.1**	**+0.1**	**+0.1**	–
LLaVA-v1.5 Low-Rank SFT (AdamW)													
LLaVA-v1.5	Vicuna-1.5-7B	LoRA	79.1	63.0	47.8	68.4	58.2	1466.2	66.1	58.9	86.4	67.8	66.2
LLaVA-v1.5	Vicuna-1.5-7B	LoRA+**SGG**	–	63.4	51.0	70.1	58.6	1477.8	66.7	59.4	86.6	68.2	–
Δ gain compared to LoRA			–	**+0.4**	**+2.2**	**+1.5**	**+0.4**	**+11.6**	**+0.6**	**+0.5**	**+0.2**	**+0.4**	–

combined with larger batch sizes, highlighting the importance of balancing these hyperparameters. Notably, SGG provides robust performance gains across all configurations, significantly reducing validation perplexity compared to standard Adam optimization. This improvement is attributed to SGG's ability to guide the optimization process more effectively, particularly in scenarios with large batch sizes and varying learning rates. Overall, the results underscore the effectiveness of SGG in enhancing model performance, even in challenging training conditions, and emphasize the critical role of hyperparameter tuning in achieving optimal results.