Lotus: Low-Rank Efficient LLM Training with Adaptive Subspace Switching

Anonymous ACL submission

Abstract

Memory-efficient learning is crucial for reducing GPU consumption and enabling scalable training of large language models. Low-rank adaptation has proven effective for fine-tuning by injecting low-rank matrices into frozen pre-trained weights. However, these methods often degrade to full-rank training due to limited expressiveness and disrupted optimization dynamics. Conversely, projecting gradient updates within a low-rank subspace improves both training performance while simultaneously decreasing memory overhead. In this paper, we propose **Lotus**, a method that speeds up gradient projection via randomized SVD and further reduces memory cost. In addition, we propose an **adaptive subspace switching strategy** guided by the average displacement of the unit gradient, which enables dynamic subspace updates for improved convergence performance. Experimental results demonstrate that Lotus is currently **the most efficient method**, surpassing full-rank training in pre-training LLaMA-type models on the C4 dataset, as well as fine-tuning across multiple tasks. Our code will soon be available.

1 Introduction

With the rapid advancement of large language models (LLMs), GPU memory requirements for training have substantially increased. Specifically, strategies such as using larger batch sizes and longer sequence lengths—which enhance model stability and generalization—have significantly elevated memory consumption. The memory footprint during LLM training typically comprises four components: model weights, gradients, optimizer states and activations. Researchers have proposed various memory-efficient techniques addressing different aspects mentioned above, including activation recomputation (Korthikanti et al., 2023), mixed-precision training, memory-efficient optimizers (Shazeer and Stern, 2018; Dettmers et al., 2021), and parameter-efficient fine-tuning (PEFT). Among these methods, low-rank adaptations of model weights, such as Lora (Hu et al., 2022) and its variants (Lialin et al., 2023; Zhang et al., 2023), have gained attention due to their reduced memory usage and comparable performance to full-parameter fine-tuning. Recent researches increasingly focus on leveraging low-rank matrix decomposition methods like Singular Value Decomposition (SVD) on model weights to improve performance (Meng et al., 2024; Lingam et al., 2024; Sun et al., 2024).

From another perspective, the model weights may not inherently possess a low-rank structure,

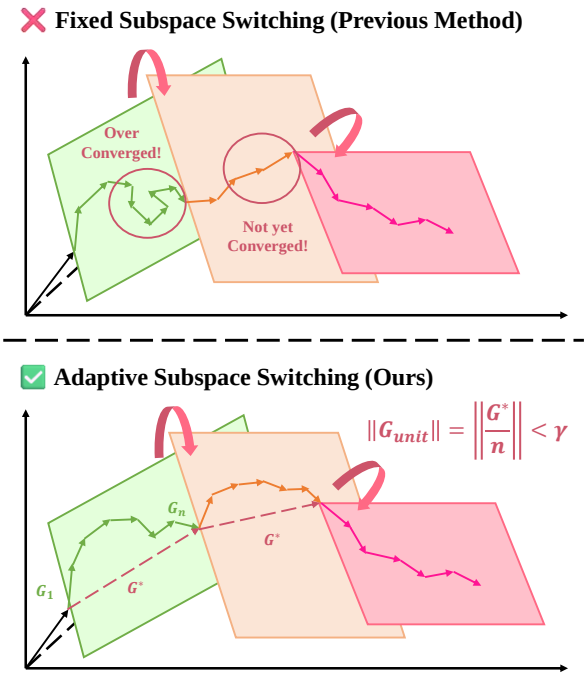

Figure 1: The comparison between previous method(e.g. GaLore) with fixed switching frequency and our greedy search strategy that update the subspace adaptively. G^* is the displacement of the gradient in a subspace. When the average displacement of unit gradient vector G_{unit} is lower than γ, the subspace will be switched.

so as to the representation of learned new feature, leading to inferior model performance (Chen et al., 2025). Compared with decomposing weight matrix, recent studies have proposed innovative approaches to activation (Chen et al., 2025), gradient, and optimizer states with low-rank paradigm. (Hao et al., 2024) first finds that LoRA updates can be viewed as performing random projection to the gradient. (He et al., 2025) improves the random projection matrix and initializes weights based on gradient information. Meanwhile, GaLore(Zhao et al., 2024a) leverages low-rank gradients for subspace updates and optimizes weights by projection. Several GaLore variants (Huang et al.; He et al., 2024; Chen et al., 2024) progress on relieving memory further.

Nevertheless, previous methodologies often exhibit inherent trade-offs between memory, computation time and performance, predominantly due to reliance on computationally intensive SVD processes and the frequent necessity to project gradients between full-rank and low-rank spaces. Meanwhile, GaLore requires specifying subspace update intervals based on subspace rank and prior knowledge, introducing ambiguity and practical difficulties. Issues arise when gradients oscillate within a subspace due to saddle points or minima, while fixed intervals may prematurely or belatedly trigger subspace changes. Such weaknesses are visualized in Figure 1. In contrast, recent adaptive approaches, like shrinking - rank strategies by (Refael et al., 2024), incur complex calculations.

Our Approach To address the trade-off and limitations mentioned, we propose Lotus: **Lo**w-Rank Efficient LLM **T**raining with Adaptive S**u**bspace **S**witching to improve training efficiency (memory & speed) and model performance. We leverage randomized SVD proposed in (Halko et al., 2011) to reduce computational complexity and saving time for periodic updating of low-rank projection. Inspired by the physics concept that displacement per unit time represents speed, we analogously propose that the gradient displacement per unit time reflects convergence status. Consequently, our central idea is to adaptively update subspaces based on the average displacement of unit gradient. We leverage the following theoretical insights:

1. While gradient magnitudes fluctuate throughout optimization, unit gradient vectors typically highlight stable optimization pathways.

2. The alignment between the unit gradient vector and subspace geometry significantly influences update effectiveness and efficiency.

Thus, we leverage an adaptive update strategy governed by the Euclidean distance of low-rank unit gradients—dynamically switching subspaces when directional consistency indicates diminishing returns. The summary of our contributions is as follows:

1. We introduce Lotus, a greedy search strategy to adaptively update subspace based on the average displacement of the unit gradient vector for better performance.

2. We leverage a more memory and time efficient method without offloading strategies based on random SVD rather than random projection or standard SVD to cooperate with update strategy for boosting the performance.

3. Lotus can further save about 40 % memory consumption on gradient and optimizer states, and reduce 10 % to 30% time cost compared with GaLore. To performance, Lotus also exceeds related algorithms greatly.

2 Related Works

Low-rank in Weight LoRA shows the potential of memory-efficient learning for LLM with the intrinsic low-rank structure (Hu et al., 2022). [(Schotthöfer et al., 2022)] constrains the rank of weight matrices to find "winning-ticket" dynamically in dense network. ReLoRA (Lialin et al., 2023) utilizes locally low-rank updates to train high-rank networks. RandLoRA (Albert et al., 2025) updates a learned linear combinations of low-rank, non-trainable random matrices. Dora (Liu et al., 2024) decomposes weights to directional matrix and magnitude vector for fine-tuning. Further works improve the low-rank adapter with the help of SVD to the weights in the initialization of the adapters (Meng et al., 2024), only training the top-r singular values (Sun et al., 2024), or the corresponding coefficients (Lingam et al., 2024). Lora-Null (Tang et al., 2025) finds the null space of representative activations and initializes the LoRA adapter with the pre-trained weights in that null space. (Wang et al., 2024) achieves high-rank updates with low costs by selecting skeletons from the pre-trained weight and learning a small matrix instead. (Jaiswal et al., 2024) unifies weight compression and memory-efficient fine-tuning as one to do adaptive low-rank weight projection.

Low-rank in Optimizer States Except for projecting the optimizer state to low-rank corresponding to the gradient structure, Fira (Chen et al., 2024) enables full-rank training under low-rank optimizer constraints by leveraging norm-based scaling and a gradient norm-growth limiter. Adapprox (Zhao et al., 2024b) finds the key effect of top singular values in the second-order momentum of Adam and approximates it by random projection. APOLLO (Zhu et al., 2024) adopts low-rank scaling factors by projecting gradients randomly, and channel-wise updates instead of element-wise in Adam to reduce optimizer memory cost. Alice (Gong et al., 2025) leverages Fisher information matrix approximation to Adam for saving memory. COSMOS (Liu et al., 2025) merges full-second-order on dominant gradient subspace and Newton-Schulz approximation on residual directions.

Low-rank in Gradient Flora first down-projects gradients by random projection matrix and up-projects low-rank gradients for optimization, and GoRA improves Flora by adaptive random projection matrix and gradient-related initialization (Hao et al., 2024; He et al., 2025). GaLore (Zhao et al., 2024a) factorizes gradients based on SVD and optimizes full-rank weights after projecting the low-rank update back to the original space, and its variants try to use less memory and relieve the complexity by random projection or layer-wise adaptation to low-rank gradients (Huang et al.; He et al., 2024; Zhang et al., 2024). Adarankgrad (Refael et al., 2024) uses adaptive subspace based on lower intrinsic rank with training process. (Torroba-Hennigen et al., 2025) leverages linear gradient transformations meeting Kronecker-factored as equivalence to a linear adapter. (Liang et al., 2024) uses online PCA to update the projection matrix in subspace learning. (Chang et al., 2025) dynamically prunes and expands ranks based on gradient-derived importance scores. (Chen et al., 2025) approximates matrix multiplication with only critical rows and columns and improves back propagation. (Zhang et al., 2025) proposes importance sampling subspace selection to improve the performance when projecting gradient.

In contrast to previous methods, our approach focuses on enforcing low-rank structure in the gradients without altering the optimization process or switching subspaces based on rank heuristics. We determine the subspace switching frequency directly from the gradient information.

3 Methodology

3.1 Randomized SVD with Power Iteration

Computing the full SVD for large matrices is computationally intensive and memory demanding. To enhance the optimization of this process, we formulate the following optimization problem $\min_{Q,U} \left\|G - QU^\top\right\|_F^2$ to find a low-rank approximation of any gradient matrix G, where $Q \in \mathbb{R}^{n\times r}$ and $U \in \mathbb{R}^{n\times r}$. Our implementation utilizes the Gaussian sampling variant of the randomized SVD algorithm (Halko et al., 2011) , which is an effective approximation method so far. Then the optimization problem becomes $\arg\min_{Q\in\mathbb{R}^{n\times r}} \left\|G - QQ^\top G\right\|_F$, and approximates the gradient matrix G as $G_{\text{app},r} \approx QQ^\top G$. The computational complexity of randomized SVD mainly arises from matrix multiplication and decomposition. For matrices G with slowly decaying singular spectra, standard randomized SVD can incur high approximation errors. Thus, we enhance precision via power iterations, which is particularly beneficial for large, spectrally flat matrices. Formally, if the singular values of G are Σ, the singular values of $\left(GG^\top\right)^q G$ are Σ^{2q+1}. Let $G' = \left(GG^\top\right)^q G$, G' can be easily derived as follows:

$$G' = \left(Q\Sigma U^\top U\Sigma Q^\top\right)^q Q\Sigma U^\top = Q\Sigma^{2q+1}U^\top \quad (1)$$

This method enhances the separation between significant and insignificant singular vectors, thereby increasing the precision of the resulting low-rank approximation.

Theorem 3.1 *(upper bound for the approximation error) Let the gradient matrix $G \in \mathbb{R}^{m\times n}$ and draw a Gaussian matrix $\Omega \sim \mathcal{N}(0,1)^{n\times(r+p)}$ with a oversampling parameter $p \approx 5$. Define $Q = \operatorname{orth}(G\Omega)$. For any fixed rank r, randomized SVD yields:*

$$\mathbb{E}\left\|G - QQ^\top G\right\|_2 \leq \left(1 + \frac{4\sqrt{r+p}}{p-1}\right)^{1/2} \sigma_{r+1}(G) \quad (2)$$

with an analogous high-probability bound. Because $\sigma_{r+1}(G)$ is precisely the spectral tail energy that GaLore already truncates, replacing the deterministic SVD with rSVD perturbs the algorithm only by an $O\left(\sigma_{r+1}(G)\right)$ term and therefore preserves both convergence properties and memory savings. The proof of Theorem3.1 can be found in Appendix A. Hence, replacing the SVD with randomized SVD maintains GaLore's memory footprint and convergence characteristics virtually unchanged. The details are shown in Algorithm 1.

Table 1: Comparison between GaLore, LoRA, and Lotus. Assume $\mathbf{W} \in \mathbb{R}^{n\times m}(n \geq m)$ and $r, p \ll \min\{m, n\}$.

	GaLore	**LoRA**	**Lotus**
Weights	mn	$mn + mr + nr$	mn
Optim States	$mr + 2nr$	$2mr + 2nr$	$mr + 2nr$
Temporary Memory	kmn	kmn	$(m + n)(r + p)$
Multi-Subspace	✓	✘	✓
Adaptive Subspace	✘	✘	✓
Pre-Training	✓	✘	✓
Fine-Tuning	✓	✓	✓

Algorithm 1: Efficient Low-rank Projector

Input: weight matrix $W \in \mathbb{R}^{m\times n}$ with $m \leq n$, rank r

$G_t \in \mathbb{R}^{m\times n} \leftarrow \nabla_W \varphi_t(W)$
Initialize Random Matrix $\Omega \in \mathbb{R}^{n\times r}$
$Y = G \cdot \Omega \in \mathbb{R}^{m\times r}$
$Q = \textit{QR Decomposition}(Y) \in \mathbb{R}^{m\times r}$
$B = Q^T \cdot W \in \mathbb{R}^{r\times n}$
$\hat{U}, S, V =$ *Singular Value Decomposition*(B)
$U = Q\hat{U} \in \mathbb{R}^{m\times r}$

return $U \in \mathbb{R}^{m\times r}$

3.2 Adaptive Subspace Switching

Refreshing the orthogonal projector with the latest full-rank gradient realigns the low-rank basis with the its current dominant directions, so the top r singular vectors reclaim energy that had drifted outside the stale subspace; the Frobenius norm of the compressed gradient therefore "jumps back up." Yet because different spectral components of the gradient drift at different speeds, a fixed update frequency both wastes compute on already stable directions and allows fast-moving ones to leak energy before the next refresh. An adaptive switching schedule throttles and accelerates the process according to subspace drift to solve this imbalance.

To quantify how much displacement such a schedule can preserve, consider the ideal scenario in which every projected gradient step points in exactly the same direction; then the cumulative displacement after k steps is

$$D_{\text{ideal}} = \left\| \sum_{i=0}^{k-1} -\alpha \hat{g}_{t-i} \right\|_2 = \alpha \left\| \sum_{i=0}^{k-1} \hat{g}_{t-i} \right\|_2 \quad (3)$$

In the "best-aligned" case where all $\hat{g}$ are parallel and have unit norm, $D_{\text{ideal}} \approx k\alpha$, where α is the learning rate. Then the actual displacement would be:

$$D_{\text{actual}} = \left\| \sum_{i=0}^{k-1} \Delta w_{t-i} \right\|_2 = \left\| \sum_{i=0}^{k-1} -\alpha P_k \hat{g}_{t-i} \right\|_2 \quad (4)$$

Then, we define the path-efficiency ratio:

$$\rho_t = \frac{D_{\text{actual}}}{D_{\text{ideal}}} = \frac{\left\| \sum_{i=0}^{k-1} P_k \hat{g}_{t-i} \right\|_2}{\left\| \sum_{i=0}^{k-1} \hat{g}_{t-i} \right\|_2} \in [0, 1] \quad (5)$$

when $\rho_t \approx 1$, the gradients remain confined within a narrow directional cone, indicating that the current subspace P_k is sufficiently representative for optimization. If $\rho_t \ll 1$, significant cancellation occurs between successive steps, suggesting that the gradients exhibit substantial directional variation or extend beyond the span of the subspace P_k.

Lotus adaptively switches the subspace when $\rho_t < \gamma$ and $t - t_{\text{last}} \geq T_{\min}$, with threshold $\gamma \in (0, 1)$. Noticing that we set a minimum interval condition, constraint $t - t_{\text{last}} \geq T_{\min}$ is imposed to prevent excessive subspace switches during the initial noisy phase of optimization. Without this safeguard, the metric ρ_t may frequently fall below the threshold γ due to stochastic fluctuations, triggering unnecessary subspace transitions. Such premature switches would incur computational overhead without meaningful improvements in convergence. Especially, $k = 1$ means that ρ_t reduces to the single-step displacement-gradient ratio.

Lemma 3.2 (one-step projected decrease) *If $\rho_t \geq \rho$ and the loss has standard L-smoothness. We can apply the standard upper bound for L-smooth functions to the subspace projected update rule, then:*

$$\mathcal{L}(w_{t+1}) \leq \mathcal{L}(w_t) - \alpha\rho^2 \|g_t\|_2^2 + \frac{1}{2}\alpha^2 L \|g_t\|_2^2 \quad (6)$$

Where $w_t \in \mathbb{R}^d$ is the parameter vector in iteration t and $g_t = \nabla \mathcal{L}(w_t)$ is the gradient of the loss function at step t. Choosing $\alpha < 2\rho^2/L$ guarantees a strict decrease.

Corollary 3.3 (k-step block) *If $\rho_{t-i} \geq \rho$ for $i = 0, \ldots, k-1$, a standard form generalized to k steps can be obtained:*

$$\mathcal{L}(w_{t+1}) = \mathcal{L}(w_{t-k+1}) - \alpha\rho^2 \sum_{i=0}^{k-1} \|g_{t-i}\|_2^2 + \frac{1}{2}\alpha^2 L \sum_{i=0}^{k-1} \|g_{t-i}\|_2^2 \tag{7}$$

The preceding analysis demonstrates that the adaptive subspace switching method remains convergent throughout the training process. On this basis, a comparative assessment of the convergence speeds between the two methodologies can be conducted by examining their respective worst-case performances under fixed-step and adaptive strategies. Under the fixed policy ρ_t can linearly decay from 1 to a minimal value $\underline{\rho}$. Assume $\rho_{t+i} = 1 - \frac{i}{T}(1-\underline{\rho}), \quad i = 0, \ldots, T-1$. The average squared ratio over one block is:

$$\bar{\rho}_{\text{fix}}^2 = \underline{\rho}^2 + \frac{(1-\underline{\rho})^2}{3} \tag{8}$$

For strong rotation ($\underline{\rho} \ll 1$) one has $\bar{\rho}_{\text{fix}}^2 \approx (1-\rho)^2/3 \ll 1$.

For worst-case behavior for Louts, the adaptive rule enforces $\rho_t \geq \gamma$ for all t, at most past steps in any window can violate this bound. Hence, $\bar{\rho}_{\text{ada}}^2 \geq \gamma^2$. Selecting $\gamma > \sqrt{(1-\underline{\rho})^2/3}$ to get the path-efficiency ratio comparison:

$$\bar{\rho}_{\text{ada}}^2 > \bar{\rho}_{\text{fix}}^2 \tag{9}$$

Then, let $\Delta_t = \mathcal{L}(w_t) - \mathcal{L}_\infty$, for fixed-interval over one switch of step T,

$$\Delta_{t+T} \leq \Delta_t - \alpha\bar{\rho}_{\text{fix}}^2 \sum_{i=0}^{T-1} G_{t+i} + \frac{1}{2}\alpha^2 L \sum_{i=0}^{T-1} G_{t+i} \tag{10}$$

With $\alpha < 2\bar{\rho}_{\text{fix}}^2/L$, we define $c_{\text{fix}} = \alpha\left(\bar{\rho}_{\text{fix}}^2 - \frac{1}{2}\alpha L\right) > 0$, the same for the adaptive scheme, The switch interval is at most k, analogously $c_{\text{ada}} = \alpha\left(\gamma^2 - \frac{1}{2}\alpha L\right)$. Since $\gamma^2 > \bar{\rho}_{\text{fix}}^2$, it can easily be deduced that $c_{\text{ada}} > c_{\text{fix}}$. Hence, by combining the above results, we arrive at the formal statement of Theorem 3.3.

Algorithm 2: Lotus Algorithm

Input: Full-rank gradient $G_F \in \mathbb{R}^{m \times n}$; avg. unit gradient displacement threshold γ; verifying gap η

Initialize: Project count $T \leftarrow 0$

if *Initialization or Subspace Update* **then** ▷ `initial step or triggered switch`

  $O_G \leftarrow$ EFFI. LOW-RANK PROJECT(G_F)

  $G_{\text{init}} \leftarrow O_G \cdot G_F$

  $d_{\text{init}} \leftarrow$ NORMALIZE(G_{init})

  $T \leftarrow 1$

end

$G_{\text{cur}} \leftarrow O_G \cdot G_F$

$d_{\text{cur}} \leftarrow$ NORMALIZE(G_{cur})

$T \leftarrow T + 1$

if $T \bmod \eta = 0$ **then** ▷ `periodic check`

  $\Delta d \leftarrow d_{\text{cur}} - d_{\text{init}}$

  $\|\bar{d}\| \leftarrow \|\Delta d\|/T$ ▷ `avg. displace`

  if $\|\bar{d}\| < \gamma$ **then** ▷ `convergence?`

    Trigger Subspace Update

  end

end

Theorem 3.4 (faster convergence with adaptive policy) *Let N_{fix} and N_{ada} denote the number of iterations required by the fixed and adaptive step size policies, respectively, to achieve the gradient tolerance condition $\sum_{t=0}^{N-1} G_t \leq \epsilon$, where $G_t = \|g_t\|_2^2$ and the step size constraint $\alpha < 2\rho_{fix}^2/L$. Then the following inequality holds:*

$$N_{\text{ada}} \leq \frac{c_{\text{fix}}}{c_{\text{ada}}} \frac{k}{T} N_{\text{fix}} < N_{\text{fix}} \tag{11}$$

This result demonstrates that Lotus's adaptive subspace switching achieves the same convergence criterion with strictly fewer iterations compared to the fixed policy, highlighting its computational efficiency. Please check Appendix A for more details.

3.3 Lotus

In this section, we introduce Lotus, a training strategy designed to simultaneously accelerate computation and reduce memory usage. Lotus uses a power-iteration-based randomized SVD to markedly accelerate the gradient projection step. In addition, it

Table 2: We compare the performance of several low-rank training algorithms with Lotus by pre-training LLaMA models of varying sizes on the C4 dataset. Here, r denotes the rank of the low-rank factorization, and d_{model} denotes the hidden states dimension of each model size. We use NVIDIA H100 GPUs for this experiment.

Method	**60M**	**130M**	**350M**	**1B**
Full Rank	34.06	25.08	18.80	15.56
GaLore	34.88	25.36	18.95	15.64
Low Rank	78.18	45.51	37.41	142.53
LoRA	34.99	33.92	25.58	19.21
ReLoRA	37.04	29.37	29.08	18.33
AdaRankGrad	<u>34.24</u>	<u>25.22</u>	<u>18.91</u>	**14.71**
Lotus	**33.75**	**24.87**	**18.91**	<u>15.33</u>
r / d_{model}	128 / 256	256 / 768	256 / 1024	512 / 2048
Training Tokens	1.1B	2.2B	6.4B	13.1B

incorporates a novel, more flexible path-efficient switching policy: we define the path efficiency ρ_t of the accumulated gradient displacement, and whenever ρ_t drops below a preset threshold while the time since the previous switch exceeds $T_{\min}$, the algorithm triggers a subspace recomputation. The details are in Algorithm 2. This mechanism guarantees that, compared to fixed-interval subspace switching, the adaptive strategy reaches the same gradient threshold in fewer iterations, thereby achieving faster convergence.

Improved Computational Performance This conclusion can easily be concluded by a straightforward analysis that the time complexity of SVD is $O(mn\min\{m,n\})$ while randomized SVD has a lower complexity of $O\left(mn(r+p)+m(r+p)^2\right)$. When $r,p \ll \min\{m,n\}$, the complexity of randomized SVD simplifies to $O(mn(r+p))$, offering a substantial improvement over the $O(mn\min\{m,n\})$ complexity of standard SVD. This efficiency renders randomized SVD an attractive choice for leading singular vector extraction in large-scale applications.

Reduced Memory Analysis In GaLore, the SVD component typically requires an additional GPU workspace of size $O(kmn)$, whereas Lotus only requires $O((m+n)(r+p))$ workspace during the gradient projection step. Since $r \ll \min(m,n)$, in practice, the workspace used by Lotus is smaller by approximately a factor of $\frac{r}{\min(m,n)}$ compared to GaLore because the peak memory footprint only includes the current layer's gradient plus the workspace. Under this circumstance, Lotus consistently achieves more than 40% memory reduction per layer. A comparison between GaLore, LoRA and Lotus is in Table 1.

4 Experiments

Implementation Details We utilize GaLore (Zhao et al., 2024a) as our codebase for model training and evaluation. All model architectures involved in the experiment are consistent with GaLore. The data format in training and validation is BF16. We manually tune the hyper-parameters needed in the experiments to achieve optimal performance. Detailed hyperparameter settings are provided in the Appendix B.

4.1 Pre-Training on C4

To assess the effectiveness of Lotus, we pre-train LLaMA models of varying sizes on the C4 dataset, following the experimental settings established by GaLore. The C4 dataset (Raffel et al., 2020), a clean version of the Common Crawl web corpus, is widely used for pre-training language models. We evaluate model performance using perplexity as the primary metric. The corresponding experimental results are reported in Table 2. Additionally, we visualize the reduction in validation perplexity for Lotus compared to the open-source baseline, GaLore, during the initial 10k pre-training steps.

4.2 Fine-Tuning Tasks

We fine-tune RoBERTa-Base model on 8 GLUE (Wang et al., 2018) tasks to compare the results with full rank fine-tuning, Lora, GaLore and AdaRankGrad, showing the results in Table 3. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA,

Table 3: Evaluating Lotus on the GLUE fintuning tasks using pre-trained RoBERTa-Base. We compare Lotus with various memory-efficient training methods and report the average metrics.

Method	Memory	CoLA	STS-B	MRPC	RTE	SST2	MNLI	QNLI	QQP	Avg
Full Fine-Tuning	747M	62.24	90.92	91.30	79.42	94.57	87.18	86.28	92.28	86.28
LoRA (rank=4)	257M	61.38	90.57	91.07	78.70	92.89	86.82	92.18	**91.29**	85.61
GaLore (rank=4)	253M	60.35	90.73	92.25	79.42	94.04	87.00	92.24	91.06	85.89
AdaRankGrad (rank=4)	202M	61.40	**90.97**	92.60	81.23	**94.80**	86.60	92.50	90.40	86.31
Lotus (rank=4)	251M	**64.02**	90.79	**93.14**	**83.39**	94.72	**87.47**	**93.00**	91.01	**87.19**
LoRA (rank=8)	264M	61.83	90.80	91.90	79.06	93.46	86.94	92.25	**91.22**	85.93
GaLore (rank=8)	257M	60.06	90.82	92.01	79.78	94.38	87.17	92.20	91.11	85.94
AdaRankGrad (rank=8)	237M	62.00	90.89	93.20	81.23	94.80	86.50	92.60	89.70	86.36
Lotus (rank=8)	254M	**63.44**	**91.06**	**93.35**	**81.58**	**94.95**	**87.32**	**93.11**	91.15	**86.99**

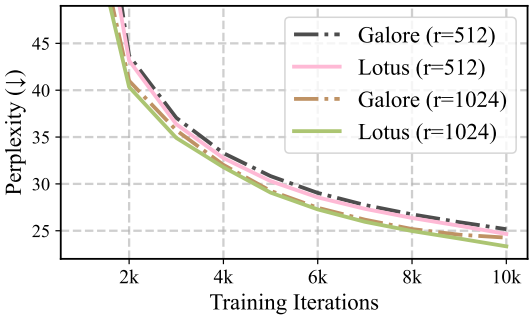

Figure 2: We apply Lotus and GaLore with the AdamW optimizer to pre-train a LLaMA 1B model on the C4 dataset for 10K steps, reporting validation perplexity throughout training. Results indicate that Lotus consistently outperforms GaLore in perplexity, regardless of whether the rank is set to 512 or 1024.

Pearson correlation for STS-B, F1 score for MRPC, and accuracy for other tasks. We set the threshold γ=0.1 and verifying gap η=50 as our baseline throughout the fine-tuning tasks. Lotus exceeds most of the tasks in previous methods and saves the memory cost. The hyper-parameters of the experiments would be shown in the Appendix C. We use NVIDIA RTX 4090 GPUs in all fine-tuning experiments.

Furthermore, we present the time efficiency metrics of GaLore, AdaRankGrad, and Lotus in Table 4. Our method demonstrates substantial time savings, particularly with an increased number of subspace updates. In this context, TSA refers to the total switching amount during training; TTC represents the total time cost (in minutes); and SSE (Subspace-Switching Efficiency) is defined as the ratio of TSA to TTC. ARG denotes AdaRankGrad. Notably, we report only the TTC for ARG, as its subspace update objective differs from that of Lotus. Specifically, ARG requires more training epochs to achieve optimal performance compared to Lotus.

Table 4: The consumption related to time for 8 tasks in GLUE benchmark. The green figure indicates the efficiency improvement ratio in comparison with GaLore.

Method	TSA	TTC	SSE
GaLore (rank=4)	3536	2191	1.6
ARG (rank=4)	-	1806	-
Lotus (rank=4)	**11614**	**1771**	**6.5** ↑306%
GaLore (rank=8)	3544	2217	1.6
ARG (rank=8)	-	1902	-
Lotus (rank=8)	**11736**	**1843**	**6.3** ↑320%

We also evaluate Lotus on the SQuAD (Rajpurkar et al., 2016). The evaluation scores of GaLore and Lora are from GaLore(Zhao et al., 2024a). Lotus outperforms GaLore in both Exact Match and F1 score.

Table 5: Evaluating Lora, GaLore and Lotus on SQuAD fine-tuning task using pre-trained BERT-Base model.

Method	Exact Match	F1
Baseline	80.83	88.41
LoRA (rank=16)	77.99	86.11
GaLore (rank=16)	80.52	88.29
Lotus (rank=16)	**80.78**	**88.32**

4.3 Ablation Studies

Analysis of Randomized SVD and Adaptive Subspace Switching As shown in Table 6, we give the performance gap in fine-tuning RoBERTa-Base on GLUE. We find that gradient low-rank projection with rSVD can achieve similar performance to the one without it, while adaptive subspace switching is essential for improving the performance.

Table 6: Ablation study of Lotus. Here, rSVD refers to the randomized SVD, and AdaSS denotes adaptive subspace switching, which are proposed in our paper.

Rank	rSVD	AdaSS	Avg
4			85.89
4	✓		85.89
4	✓	✓	**87.19**
8			85.94
8	✓		86.07
8	✓	✓	**86.99**

Subspace Switching Frequency with Displacement We present the frequency distribution visualization from fine-tuning RoBERTa-Base on the STS-B dataset in Figure 3. The subspace update frequencies concentrate in 300 while exhibiting diversity in other 3 numbers, demonstrating both the effectiveness and independence of our adaptive subspace approach. The drifting trends share some similar patterns but have different subspaces update frequencies, which are shown in Figure 4.

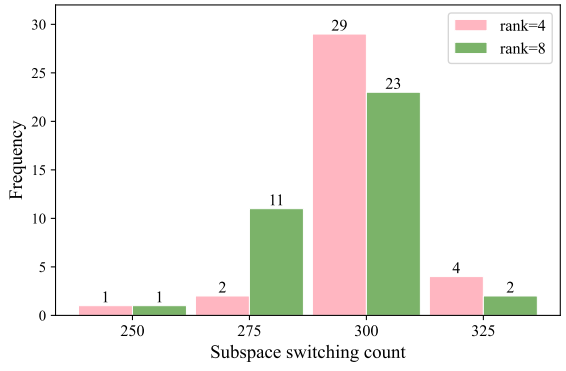

Figure 3: Visualization example of fine-tuning on STS-B with average unit gradient displacement threshold γ=0.005 and verifying gap η=25 under different rank.

What does the η and γ mean? The verifying gap η aims to avoid mistakenly updates of subspace by fluctuated gradients and saving time. The

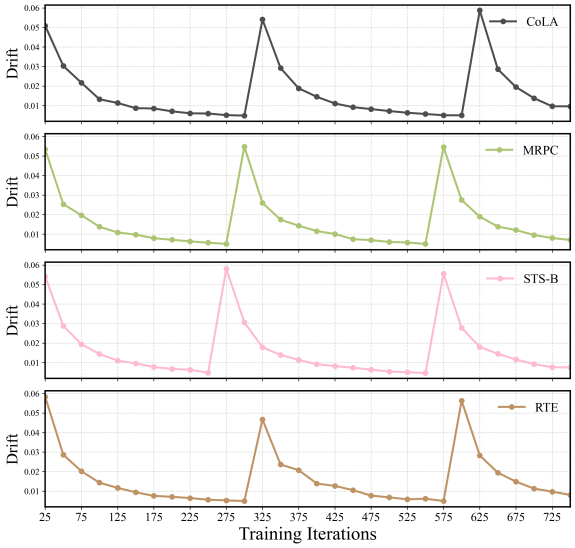

Figure 4: Visualization of the average displacement of unit gradient drifting trends (drift) in updated subspaces for 4 fine-tuning tasks with rank=4.

threshold γ to average unit gradient norm serves to regulate the optimization process. When γ is set above 0.02, it imposes rigorous supervision on the average displacement. Conversely, γ below 0.01 results in more lenient regulation. Notably, γ should never be configured below 0.005, as this would cause infrequent subspace updates compromising optimization effectiveness. Comparative analysis revealed that while values of γ within the 0.005-0.02 range, and η within the 25-100 produce generally comparable results during fine-tuning, they exhibit either marginally positive or negative effects relative to the baseline. Detail experiments are shown in Table 9 in Appendix C.

5 Conclusion

In this paper, we introduce Lotus, an algorithm that not only speed up the gradient projection process, save memory consumption in memory efficient learning by randomized SVD, but also improve the convergence performance further to surpass the full-rank pretraining and fine-tuning. Specifically, we leverage adaptive subspace switching frequency guided by the average displacement of the unit gradient. Unlike previous work focusing on the rank of the subspace, we utilize the gradient itself to update the subspace, which is more explainable and effective. Lotus tackles the memory-efficiency trade-off in LLM training through a novel approach that boosts training speed, improves model performance, and reduces memory footprint.

6 Limitations

Although we have done a comprehensive experiments to show the effectiveness of Lotus, there are still some aspects that requires improvements.

Hyper-parameter Illustration In our experiments, we mainly set threshold around 0.01 and verifying gap as 50. There might be better combination for the threshold γ, verifying gap η, learning rate, batch size, etc. We mainly inherit and tuning the hyper-parameter from GaLore, but it is lack of illustration.

Stable Subspace Switching Our work inherits the limitation of low-rank gradient projection methods: performance degradation during subspace switching due to abrupt gradient changes. While exponential moving average mitigates this issue, its effectiveness varies across tasks. The instability from gradient norm surges remains an open challenge in this research line.

Experiments on Scaling Models Lotus should be more friendly in FSDP environment for the reducing computational and memory budget by randomized SVD. We consider that the hyper-parameter should also work on the scaling models and distributed environments. Due to the limited resources, we leave this work in the future.

References

Paul Albert, Frederic Z Zhang, Hemanth Saratchandran, Cristian Rodriguez-Opazo, Anton van den Hengel, and Ehsan Abbasnejad. 2025. Randlora: Full-rank parameter-efficient fine-tuning of large models. *arXiv preprint arXiv:2502.00987*.

Huandong Chang, Zicheng Ma, Mingyuan Ma, Zhenting Qi, Andrew Sabot, Hong Jiang, and HT Kung. 2025. Elalora: Elastic & learnable low-rank adaptation for efficient model fine-tuning. *arXiv preprint arXiv:2504.00254*.

Guanduo Chen, Yutong He, Yipeng Hu, Kun Yuan, and Binhang Yuan. 2025. Ce-lora: Computation-efficient lora fine-tuning for language models. *arXiv preprint arXiv:2502.01378*.

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*.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 2021. 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*.

Wenbo Gong, Meyer Scetbon, Chao Ma, and Edward Meeds. 2025. Towards efficient optimizer design for llm via structured fisher approximation with a low-rank extension. *arXiv preprint arXiv:2502.07752*.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288.

Yongchang Hao, Yanshuai Cao, and Lili Mou. 2024. Flora: Low-rank adapters are secretly gradient compressors. *arXiv preprint arXiv:2402.03293*.

Haonan He, Peng Ye, Yuchen Ren, Yuan Yuan, and Lei Chen. 2025. Gora: Gradient-driven adaptive low rank adaptation. *arXiv preprint arXiv:2502.12171*.

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

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

Weihao Huang, Zhenyu Zhang, Yushun Zhang, Zhi-Quan Luo, Ruoyu Sun, and Zhangyang Wang. Galore-mini: Low rank gradient learning with fewer learning rates. In *NeurIPS 2024 Workshop on Fine-Tuning in Modern Machine Learning: Principles and Scalability*.

Ajay Jaiswal, Lu Yin, Zhenyu Zhang, Shiwei Liu, Jiawei Zhao, Yuandong Tian, and Zhangyang Wang. 2024. From galore to welore: How low-rank weights non-uniformly emerge from low-rank gradients. *arXiv preprint arXiv:2407.11239*.

Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5:341–353.

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

Kaizhao Liang, Bo Liu, Lizhang Chen, and Qiang Liu. 2024. Memory-efficient llm training with online subspace descent. *arXiv preprint arXiv:2408.12857*.

Vijay Chandra Lingam, Atula Neerkaje, Aditya Vavre, Aneesh Shetty, Gautham Krishna Gudur, Joydeep Ghosh, Eunsol Choi, Alex Dimakis, Aleksandar Bojchevski, and Sujay Sanghavi. 2024. Svft: Parameter-efficient fine-tuning with singular vectors. *Advances in Neural Information Processing Systems*, 37:41425–41446.

Liming Liu, Zhenghao Xu, Zixuan Zhang, Hao Kang, Zichong Li, Chen Liang, Weizhu Chen, and Tuo Zhao. 2025. Cosmos: A hybrid adaptive optimizer for memory-efficient training of llms. *arXiv preprint arXiv:2502.17410*.

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

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. 2024. Pissa: Principal singular values and singular vectors adaptation of large language models. *Advances in Neural Information Processing Systems*, 37:121038–121072.

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. *Journal of machine learning research*, 21(140):1–67.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Yehonathan Refael, Jonathan Svirsky, Boris Shustin, Wasim Huleihel, and Ofir Lindenbaum. 2024. Adarankgrad: Adaptive gradient-rank and moments for memory-efficient llms training and fine-tuning. *arXiv preprint arXiv:2410.17881*.

Steffen Schotthöfer, Emanuele Zangrando, Jonas Kusch, Gianluca Ceruti, and Francesco Tudisco. 2022. Low-rank lottery tickets: finding efficient low-rank neural networks via matrix differential equations. *Advances in Neural Information Processing Systems*, 35:20051–20063.

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

Chengwei Sun, Jiwei Wei, Yujia Wu, Yiming Shi, Shiyuan He, Zeyu Ma, Ning Xie, and Yang Yang. 2024. Svfit: Parameter-efficient fine-tuning of large pre-trained models using singular values. *arXiv preprint arXiv:2409.05926*.

Pengwei Tang, Yong Liu, Dongjie Zhang, Xing Wu, and Debing Zhang. 2025. Lora-null: Low-rank adaptation via null space for large language models. *arXiv preprint arXiv:2503.02659*.

Lucas Torroba-Hennigen, Hunter Lang, Han Guo, and Yoon Kim. 2025. On the duality between gradient transformations and adapters. *arXiv preprint arXiv:2502.13811*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Qibin Wang, Xiaolin Hu, Weikai Xu, Wei Liu, Jian Luan, and Bin Wang. 2024. Pmss: Pretrained matrices skeleton selection for llm fine-tuning. *arXiv preprint arXiv:2409.16722*.

Haochen Zhang, Junze Yin, Guanchu Wang, Zirui Liu, Tianyi Zhang, Anshumali Shrivastava, Lin Yang, and Vladimir Braverman. 2025. I3s: Importance sampling subspace selection for low-rank optimization in llm pretraining. *arXiv preprint arXiv:2502.05790*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*.

Zhenyu Zhang, Ajay Jaiswal, Lu Yin, Shiwei Liu, Jiawei Zhao, Yuandong Tian, and Zhangyang Wang. 2024. Q-galore: Quantized galore with int4 projection and layer-adaptive low-rank gradients. *arXiv preprint arXiv:2407.08296*.

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*.

Pengxiang Zhao, Ping Li, Yingjie Gu, Yi Zheng, Stephan Ludger Kölker, Zhefeng Wang, and Xiaoming Yuan. 2024b. Adapprox: Adaptive approximation in adam optimization via randomized low-rank matrices. *arXiv preprint arXiv:2403.14958*.

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

A Proofs

A.1 Subspace-Approximation Error of Randomized SVD

The two–stage Halko–Martinsson–Tropp scheme

$$\underbrace{Y}_{m\times(r+p)} = G\Omega,$$
$$[P,\_\,] = \mathrm{qr}(Y), P \in \mathbb{R}^{m\times(r+p)},$$

Exact SVD in the reduced space decomposes $B = P^\top G$ and lifts its left singular vectors U_B by $U = PU_B$

$$\left\|G - PP^\top G\right\|_2 \leq$$
$$(1+C_1)\,\sigma_{r+1}(G) + C_2\left(\sum_{j>r}\sigma_j^2(G)\right)^{1/2}$$

with constants $C_1, C_2 = O(\sqrt{r}/p)$. If the singular values decay polynomially or exponentially—an empirical fact for deep-network gradients—the right-hand side is negligible. Replacing Y by $\left(GG^\top\right)^q G\Omega$ rescales the singular values to σ_j^{2q+1} and tightens the bound to $\sigma_{r+1}^{1/(2q+1)}$.

Let $(P_\star, Q_\star)$ be the exact rank- r singular subspaces and (P, Q) their randomized counterparts. Davis-Kahan perturbation theory implies

$$\|P - P_\star R\|_2$$

$$\|Q - Q_\star S\|_2 \leq \delta := \frac{\left\|G - PP^\top G\right\|_2}{\sigma_r(G)}$$

Because Adam, Adafactor, and similar optimizers are coordinate-wise 1-Lipschitz,

$$\left\|P^\top G - P_\star^\top G\right\|_2 \leq \delta\|G\|_2$$

$$\|G_{\mathrm{upd}} - G_{\mathrm{upd},\star}\|_2 = O(\delta)\|G\|_2$$

As GaLore's linear convergence guarantee $\|R_t\|_F \leq (1-\eta\kappa)\,\|R_{t-1}\|_F$ is now perturbed by an additive term $\varepsilon_t = O\left(\delta\,\|G_t\|_2\right)$. Standard noisy-gradient analysis yields

$$\|R_t\|_F \leq (1-\eta\kappa)^t\,\|R_0\|_F + \frac{\eta}{\kappa}\sup_t \varepsilon_t$$

so convergence is retained provided $\delta \ll \kappa/\,\|G_t\|_2$

A.2 Proof of Lemma 3.1

Based on the gradient-Lipschitz property

$$\mathcal{L}\left(w_{t+1}\right) \leq \mathcal{L}\left(w_t\right) + g_t^\top\left(w_{t+1} - w_t\right) + \frac{L}{2}\left\|w_{t+1} - w_t\right\|_2^2$$

applied to the GaLore projected update

$$w_{t+1} = w_t - \eta P_k g_t^{\text{proj}}, \quad g_t^{\text{proj}} := P_k^\top g_t$$

a single step descent

$$\begin{aligned} g_t^\top\left(w_{t+1} - w_t\right) &= -\eta g_t^\top P_k g_t^{\text{proj}} \\ &= -\eta\left\|P_k g_t^{\text{proj}}\right\|_2^2 \\ &= -\eta\rho_t^2\left\|g_t\right\|_2^2 \end{aligned}$$

$$\begin{aligned} \left\|w_{t+1} - w_t\right\|_2^2 &= \eta^2\left\|P_k g_t^{\text{proj}}\right\|_2^2 \\ &= \eta^2\rho_t^2\left\|g_t\right\|_2^2 \\ &\leq \eta^2\left\|g_t\right\|_2^2 \end{aligned}$$

where the last inequality simply drops the factor $\rho_t^2 \leq 1$ so that the expression matches the unified form $\frac{1}{2}\eta^2 L\left\|g_t\right\|_2^2$, then Lemma 3.1 is proved.

Fix any $\rho_t \geq \rho > 0$. Replacing ρ_t by its lower bound ρ yields the projected-descent inequality. Whenever the stepsize satisfies $\eta < \frac{2\rho^2}{L}$ the negative term $-\eta\rho^2\left\|g_t\right\|_2^2$ dominates the curvature penalty $\frac{1}{2}\eta^2 L\left\|g_t\right\|_2^2$, and the iteration achieves strict single-step reduction in the objective.

B Details of the Pre-Training Experiment

Table 7: Hyperparameters of pre-training different size of LLaMA models on C4 datasets. LR refers to the learning rate, Min LR ratio refers to the minimal learning rate ratio of learning rate.

	60M	**130M**	**350M**	**1B**
LR	0.02	0.02	0.02	0.025
Lotus α	0.3	0.3	0.3	0.3
Min LR ratio	0.2	0.2	0.2	0.1
Threshold γ	0.009	0.009	0.02	0.02

C Details of the Fine-Tuning Experiment

Table 8 shows the hyperparameters we used to fine-tune RoBERTa-Base model on the GLUE benchmark. We also show the different combinations of threshold γ and verifying gap η in fine-tuning RoBERTa-Base on 4 GLUE tasks with previous hyper-parameters. The average results of baseline are the best.

Table 8: Hyperparameters of fine-tuning RoBERTa-Base for Lotus.

	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	2E-05	3E-05	1E-05	1E-05	1E-05	1E-05
Lotus α	4							
Rank Config.	$r = 4$							
Max Seq. Len.	512							
	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
Batch Size	16	16	16	32	16	16	32	16
# Epochs	30	30	30	30	30	30	30	30
Learning Rate	1E-05	2E-05	5E-05	3E-05	1E-05	2E-05	2E-05	1E-05
Lotus α	2	2	2	4	2	2	4	4
Rank Config.	$r = 8$							
Max Seq. Len.	512							

Table 9: Ablation study to the combination of hyper-parameter settings to threshold γ and verifying gap η. The gray lines represents the baseline in main body section.

	γ	η	**CoLA**	**STS-B**	**MRPC**	**RTE**	**Avg**
Lotus(rank=4)	0.01	50	64.02	90.79	**93.14**	**83.39**	82.83
	0.005	25	64.10	90.70	91.95	79.42	81.54
	0.005	50	**64.27**	90.44	92.22	82.31	82.31
	0.005	100	62.57	90.63	92.66	79.42	81.32
Lotus(rank=4)	0.01	25	63.31	90.67	92.73	78.70	81.35
	0.01	100	64.22	90.50	92.72	78.70	81.53
	0.02	25	62.90	**90.80**	92.16	77.97	80.95
	0.02	50	63.84	90.73	92.25	79.06	81.47
	0.02	100	64.22	90.50	92.72	79.78	81.80
Lotus(rank=8)	0.01	50	63.44	**91.06**	93.35	81.58	82.35
	0.005	25	60.92	90.93	92.60	82.67	81.78
	0.005	50	62.43	90.87	92.76	**83.03**	82.27
	0.005	100	**63.74**	90.89	92.33	81.58	82.13
Lotus(rank=8)	0.01	25	63.26	90.79	92.74	81.58	82.09
	0.01	100	62.92	90.91	92.49	82.31	82.15
	0.02	25	61.73	90.90	**93.47**	81.58	81.92
	0.02	50	63.10	90.73	93.14	80.50	81.86
	0.02	100	62.92	90.91	92.49	82.31	82.15