OASIS: Online Activation Subspace Learning for Memory-Efficient Training

Sakshi Choudhary [1] **Utkarsh Saxena** [2 *] **Kaushik Roy** [1]

Abstract

Training large language models (LLMs) is constrained by memory requirements, with activations accounting for a substantial fraction of the total footprint. Existing approaches reduce memory using low-rank weight parameterizations or low-rank gradient subspaces for optimizer states, while activation memory is addressed through architectural modifications or compression schemes based on periodically updated projections. We propose OASIS, an online activation subspace learning algorithm for memory-efficient training that tracks and continuously updates a low-dimensional activation subspace during training. Intermediate activations are projected onto this evolving subspace, reducing memory without modifying forward-pass computations. The evolving activation subspace induces low-rank gradient representations, enabling both gradients and optimizer states to be maintained directly in this subspace, while a projection-aware optimizer consistently transports optimizer states across subspace updates for stable training. Across various finetuning and pretraining tasks, OASIS achieves up to $2\times$ lower peak memory than full fine-tuning while matching its performance and outperforming prior low-rank methods.

1. Introduction

Training large language models (LLMs) is increasingly constrained by memory rather than compute alone (Rajbhandari et al., 2020; Korthikanti et al., 2022). This trend is further exacerbated in modern long-context and high-throughput training regimes, where increasing sequence length and batch size amplify memory requirements. As a result, memory, not compute, often becomes the primary bottleneck limiting scalability. In addition to model parameters and gradients, training requires storing optimizer states, which typically increase parameter memory by approximately 3× (Kingma & Ba, 2017), as well as intermediate activations for backpropagation. These components scale differently with model and training configuration, and together dominate the total training footprint, as illustrated in Figure 1a.

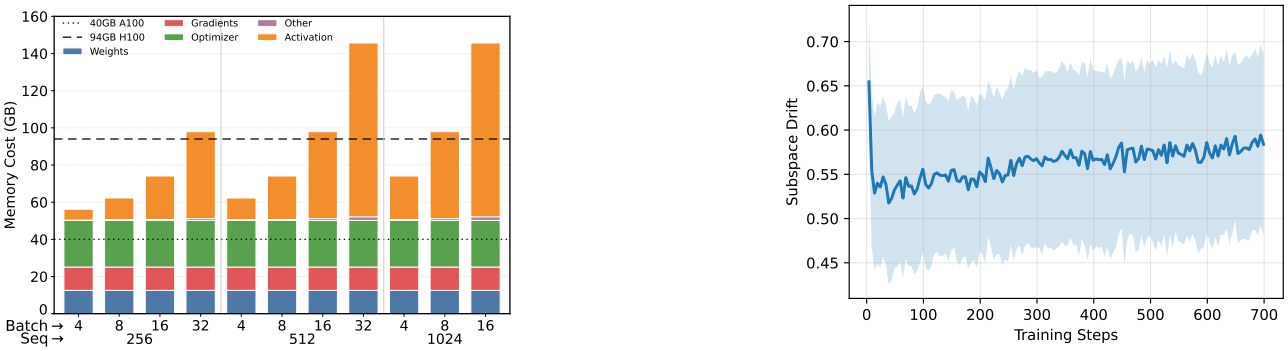
A promising direction is to exploit the low-rank structure of weights, gradients, and activations during training. Prior work has leveraged low-rank parameterizations (Hu et al., 2021) and gradient subspaces (Zhao et al., 2024; Robert et al., 2025) to reduce optimizer memory while maintaining full-parameter updates. However, these approaches do not address the memory required to store activations for backpropagation, which scales with batch size and sequence length and often dominates at scale. Conversely, methods that target activation memory footprint typically require architectural changes (Liu et al., 2025) or introduce approximation errors that can degrade optimization performance (Miles et al., 2024). As a result, activation memory, often the dominant component at scale, remains inadequately addressed.

In this paper, we propose OASIS, an activation-aware low-rank training algorithm based on a simple but powerful observation: for a linear layer, the weight gradients lie in the span of input activations (Zhang et al., 2017). This motivates identifying a low-rank activation subspace that can be used to represent both input activations and the gradients. Gradients inherit the low-rank structure induced by this subspace and can therefore be represented in low rank, reducing gradient memory and naturally inducing low-rank optimizer states. Unlike prior approaches that target individual components of training memory, our method compresses activations, gradients, and optimizer states through a single algorithm. This unified view enables end-to-end memory reduction without modifying the forward pass or introducing additional approximation stages.

In practice, this subspace is not static. As model parameters evolve, the distribution of activations shifts, causing the underlying low-rank subspace to drift over time. To demonstrate this, we measure the average activation subspace drift across all layers between successive training iterations during Llama 2 7B (et al., 2023) finetuning on GSM8K (Cobbe

*Work done while at Purdue University. [1]Department of Electrical and Computer Engineering, Purdue University, USA [2]Advanced Micro Devices, Inc. (AMD). Correspondence to: Sakshi Choudhary <choudh23@purdue.edu>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).



(a) Memory breakdown during training

(b) Activation subspace drift

Figure 1. Memory breakdown and subspace dynamics during training. (a) As batch size and sequence length scale, memory becomes increasingly dominated by activations. (b) These activations lie in subspaces that evolve continuously during training.

et al., 2021), with higher values indicating greater drift. As shown in Figure 1b, the drift remains consistently non-zero throughout training, indicating that the activation subspace continues to evolve rather than stabilizing to a fixed basis. A fixed projection basis therefore becomes progressively stale, failing to capture newly emerging directions of variance in the activation distribution. This mismatch leads to increasingly inaccurate projections, which can degrade optimization and impact both convergence and final performance.

To address this, we develop an online activation subspace learning mechanism based on Oja's rule (Oja, 1997), which incrementally updates the low-rank subspace during training without requiring repeated exact eigen-decomposition. This allows the model to continuously adapt its representation to the evolving activation landscape. As the projection basis evolves, the compressed optimizer states (e.g., momentum and variance) must also be updated consistently. Following LDAdam (Robert et al., 2025), we use projection-aware updates to ensure alignment with the current subspace and maintain stable low-rank optimization.

We evaluate OASIS on both finetuning and pretraining tasks across a range of model scales. Experiments on GSM8K (Cobbe et al., 2021), HumanEval (et al., 2021), and C4 (Dodge et al., 2021) with Llama models demonstrate that OASIS achieves substantial memory savings while maintaining competitive performance. In particular, OASIS reduces peak memory by up to $2\times$ compared to full fine-tuning and by over 30% relative to LDAdam (Robert et al., 2025) at comparable accuracy, while outperforming prior low-rank methods in aggressive compression regimes. These gains persist in pretraining, highlighting the robustness of our approach across model scales and training settings. Importantly, all improvements are achieved without modifying model architectures or training objectives. Our contributions are as follows:

- We introduce OASIS, an online activation subspace learning algorithm for memory-efficient training, which leverages the low-rank structure of activations to jointly compress stored activations, gradients, and optimizer states.
- We develop an online subspace adaptation mechanism based on Oja's rule that incrementally tracks the evolving activation subspace during training without requiring repeated exact eigen decomposition on activation statistics.
- Through extensive experiments on finetuning and pretraining tasks, we show that OASIS achieves up to $2\times$ lower peak memory than full fine-tuning while consistently outperforming prior low-rank methods.

2. Related Works

Low-Rank Parameterization. Low-rank parameterization methods such as LoRA (Hu et al., 2021) and its variants (Zhang et al., 2023; Dettmers et al., 2023; Liu et al., 2024) restrict updates to trainable low-rank adapters while keeping the backbone weights frozen, thus significantly reducing the number of trainable parameters and the memory footprint associated with gradients and optimizer states. However, these approaches restrict optimization to a fixed low-dimensional update space, which can limit their ability to match full-parameter training performance (Zhao et al., 2024; Biderman et al., 2024). To address this limitation, subsequent methods increase the effective rank of updates by aggregating multiple low-rank updates over time (Lialin et al., 2023; Xia et al., 2024). However, they typically require an initial warm-up phase with full-parameter training (Lialin et al., 2023), which undermines the benefits of low-rank optimization.

Low-Rank Optimization via Gradient Subspaces. A complementary line of work performs full-parameter optimization while compressing optimizer states by exploiting the

low-rank structure of gradients (Zhao et al., 2024; Robert et al., 2025; Modoranu et al., 2024; Rajabi et al., 2025; Liang et al., 2024). GaLore (Zhao et al., 2024) periodically applies Singular Value Decomposition (SVD) to identify dominant gradient directions and construct projection matrices defining a low-dimensional gradient subspace. Gradients are projected onto this subspace to update optimizer states, reducing memory while preserving full-parameter updates. LDAdam (Robert et al., 2025) improves upon this by introducing error-feedback and projection-aware state updates to maintain consistency as the subspace evolves. Subsequent works reduce the overhead of periodic SVD through randomized projections (Hao et al., 2024) or streaming subspace estimation via online updates (Liang et al., 2024; Rajabi et al., 2025). However, these methods focus on reducing optimizer-state memory and do not scale to large-scale training settings where activation memory dominates the model footprint (Figure 1a).

Activation Compression during Training. A common approach to reducing activation memory is to avoid storing intermediate activations and instead recompute them on-the-fly during the backward pass (Chen et al., 2016). While effective, this shifts the burden from memory to compute, resulting in increased training time. VeLoRA (Miles et al., 2024) reduces activation memory by splitting tokens into sub-tokens and compressing them into a one-dimensional subspace. However, it requires projecting activations back to full space for gradient computation and is used only for a subset of layers, limiting its generality. CompAct (Shamshoum et al., 2025) applies low-rank compression to stored activations using random projections, which can slightly reduce the computational cost of GaLore-style optimization, but at the expense of accuracy, underperforming both GaLore and full-parameter training. Similarly, LANCE (Apolinario & Roy, 2026) utilizes SVD to obtain low-rank activation subspace for efficient on-device continual learning setup and is not directly applicable to standard training. Taking a different approach, CoLA (Liu et al., 2025) modifies the model architecture by replacing linear and projection layers with autoencoders to induce low-rank activations, but this restricts its applicability to fine-tuning existing LLMs. Overall, these approaches either require invasive architectural modifications or achieve memory savings at the cost of performance, limiting their effectiveness for general-purpose training.

Activation Compression for Inference-Time Efficiency. Recent work has shown that transformer activations lie in low-dimensional subspaces and exploits this structure to improve inference efficiency in LLMs. Several approaches leverage activation subspaces for model compression and pruning, including SliceGPT (Ashkboos et al., 2024), which reduces hidden dimensionality via subspace projection, ASVD (Yuan et al., 2025), which performs activation-aware low-rank decomposition of model weights, and ModeGPT (Lin et al., 2025), which uses activation-guided structure for pruning. Another line of work focuses on reducing KV-cache memory by exploiting low-rank activation structure, projecting keys and values into lower-dimensional spaces during decoding (Saxena et al., 2024; Chang et al., 2025; Zhang et al., 2024; Singhania et al., 2024). Activation-aware quantization methods further leverage this structure to compress activations with minimal accuracy loss (Saxena et al., 2025). Collectively, these methods rely on the observation that activation covariance is approximately low-rank, enabling efficient compression, pruning, and acceleration at inference time. In contrast, we leverage activation subspace structure during training as a foundation for low-rank optimization.

3. Methodology

OASIS reduces the memory footprint of training large models by performing optimization in a dynamically evolving low-dimensional subspace of activations. The key idea is to compress the activations stored for the backward pass while keeping the forward pass exact, enabling memory savings without altering forward computations. This design is motivated by the observation that activations often lie on a low-dimensional manifold, making them amenable to compression without significant loss of information. By restricting optimization to this subspace while preserving exact forward computations, OASIS achieves memory savings without sacrificing model fidelity or requiring architectural changes.

Consider a linear layer with input activations $X_t \in \mathbb{R}^{N\times d}$ at iteration t, and output gradients $G_t^{\text{out}} \in \mathbb{R}^{N\times m}$. The gradient with respect to the weight matrix can be written as

$$\nabla_W = X_t^\top G_t^{\text{out}}, \tag{1}$$

which lies in the span of the input activations. This enables low-rank optimization by projecting activations onto a rank-r subspace using an orthonormal basis $U_t \in \mathbb{R}^{d\times r}$:

$$\widetilde{X}_t = X_t U_t. \tag{2}$$

The resulting low-rank gradient is

$$\widetilde{G}_t = \widetilde{X}_t^\top G_t^{\text{out}}, \tag{3}$$

corresponding to optimal rank-r approximation of the gradient within the activation span.

This formulation avoids materializing full-rank gradients and allows both gradients and optimizer states to be maintained directly in the low-rank subspace. For optimizers such as Adam, the first- and second-moment estimates are computed and stored in $\mathbb{R}^{r\times m}$ using $\widetilde{G}_t$, rather than $\mathbb{R}^{d\times m}$. Parameter updates computed using $\widetilde{G}_t$ are projected back to

the full space via U_t. This reduces activation storage from $O(Nd)$ to $O(Nr)$ and both gradient and optimizer state storage from $O(dm)$ to $O(rm)$ per layer.

To remain effective, this approach requires maintaining a subspace U_t that captures the dominant directions of the activation distribution throughout training. We initialize U_0 using principal components of early activations and update it online using an efficient streaming method described next. Algorithm 1 summarizes the overall procedure.

3.1. Online Activation Subspace Learning

A key challenge in subspace-based optimization is maintaining a basis U_t that tracks the evolving activation subspace during training. As model parameters are updated, the activation distribution changes, and the optimal low-rank subspace is inherently time-varying. This requires a method that can adapt the subspace continuously and efficiently.

Existing approaches typically rely on periodic subspace updates via Singular Value Decomposition (SVD) (Zhao et al., 2024) or on per-iteration approximations computed from individual mini-batches (Robert et al., 2025). Periodic updates can lead to stale subspaces between updates, while batch-level methods are often noisy and fail to capture the temporal evolution of the activation distribution. As a result, both approaches can yield suboptimal projections and degrade optimization performance.

To address this, we adopt an online subspace update based on Oja's rule (Oja, 1997), a classical streaming PCA method designed to estimate principal components from sequential data. This makes it well-suited for training settings, where data arrive in mini-batches and the underlying subspace evolves over time. The update incrementally steers the basis toward directions of high variance while removing components already captured by the current subspace:

$$U_t \leftarrow U_{t-1} + \gamma(I - U_{t-1}U_{t-1}^\top)C_tU_{t-1}, \tag{4}$$

where $C_t = \frac{1}{N}X_t^\top X_t$ denotes the activation covariance at iteration t.

Here, C_tU_{t-1} biases the basis toward the principal directions of the activation distribution, while the projection $(I - U_{t-1}U_{t-1}^\top)$ removes components already captured by the current subspace. This yields an efficient incremental approximation to principal component analysis without requiring explicit eigendecomposition.

However, directly applying the update described in Equation 4 introduces two challenges. First, the update does not preserve orthonormality, and repeated iterations can cause the basis vectors to become correlated. Maintaining an orthonormal basis is essential to ensure that projections correspond to least-squares optimal approximations. Second, the effective update magnitude depends on the scale of C_t, which varies across layers and training iterations. A fixed step size can therefore lead to instability when activation magnitudes are large, or slow adaptation when they are small.

To address these issues, we make two modifications. We explicitly re-orthonormalize U_t after each update to maintain an orthonormal basis, incurring a cost of $O(dr^2)$, which is significantly cheaper than full eigendecomposition. We also normalize the step size using

$$\gamma_t = \frac{\gamma}{\|C_t\|}, \tag{5}$$

making the update adaptive to activation scale and improving stability across layers and training stages. Additionally, we adopt a projection-aware adaptive optimizer inspired by Robert et al. (2025) to ensure consistency under subspace evolution. Together, these modifications enable stable and adaptive tracking of the activation subspace throughout training, avoiding the staleness of periodic methods and the noise of batch-level approximations, while incurring minimal computational overhead.

3.2. Projection Aware Optimizer

We maintain optimizer states in a low-dimensional activation subspace that evolves during training. As the subspace changes, optimizer states are transported to remain aligned with the current basis. Let U_{t-1} and U_t denote the orthonormal bases at iterations $t-1$ and t. The subspace transition matrix $T_t = U_t^\top U_{t-1}$. defines the change of coordinates between subspaces and is used to transport optimizer states from one low-rank subspace to another.

Following prior work (Robert et al., 2025), we view Adam's states as coordinate-wise estimates of gradient moments. Under this view, the first moment M_t can be directly projected between subspaces using the linear transformation induced by T_t (Algorithm 1, Line 7). The second moment V_t, however, cannot be directly projected between subspaces, as it depends on cross-coordinate interactions not captured by coordinate-wise estimates. We approximate it in the new subspace using projected variance and mean-squared terms derived from the transported moments, avoiding explicit covariance estimation (Algorithm 1, Line 8).

4. Experiments

4.1. Experimental Setup

Models and Datasets. We evaluate OASIS across both finetuning and pretraining using language models of varying scales. For downstream tasks, we use Llama-2 7B (et al., 2023) and Llama-3.2 1B (et al., 2024), finetuned on the GSM8K dataset (Cobbe et al., 2021). For code generation, we finetune Llama-2 7B on CodeAlpaca (Chaudhary, 2023) and evaluate on HumanEval (et al., 2021). To study the be-

Algorithm 1 OASIS

Notation: Step size η_t, decay rates β_1, β_2, Oja step size γ, rank r. $W \in \mathbb{R}^{d\times m}$ denotes the weight matrix of a given layer. $X_t \in \mathbb{R}^{N\times d}$ and $G_t^{\text{out}} \in \mathbb{R}^{N\times m}$ are the corresponding input activations and output gradients, where $N = bn$, with b the batch size and n the sequence length. $T_t = U_t^\top U_{t-1}$ represents the subspace transition matrix

Initialization: Use X_0 to form $C_0 = \frac{1}{N} X_0^\top X_0$ and initialize $U_0 \in \mathbb{R}^{d\times r}$ as the top-r principal components. Set $M_0 = 0$ and $V_0 = 0$. $|\cdot|$ denotes element-wise absolute value.

for $t = 1, 2, \ldots, T$ **do**
  $C_t \leftarrow \frac{1}{N} X_t^\top X_t$ //Activation covariance
  $U_t \leftarrow U_{t-1} + \frac{\gamma}{\|C_t\|}(I - U_{t-1}U_{t-1}^\top)C_t U_{t-1}$ //Online subspace update (Oja-based)
  $U_t \leftarrow \text{orth}(U_t)$ //Re-orthonormalize basis
  $\widetilde{X}_t \leftarrow X_t U_t$
  $\widetilde{G}_t \leftarrow \widetilde{X}_t^\top G_t^{\text{out}}$
  $M_t \leftarrow \beta_1(T_t M_{t-1}) + (1-\beta_1)\widetilde{G}_t$ //Projection aware optimizer
  $V_t \leftarrow \beta_2\Big[(1-\beta_2^{t-1})\Big|(T_t)^2 \odot (V_{t-1} - M_{t-1}^2) + (T_t M_{t-1})^2\Big|\Big] + (1-\beta_2)\widetilde{G}_t^2$
  $\widehat{M}_t \leftarrow M_t/(1-\beta_1^t)$; $\widehat{V}_t \leftarrow V_t/(1-\beta_2^t)$
  $\widetilde{N}_t \leftarrow \widehat{M}_t/(\sqrt{\widehat{V}_t} + \epsilon)$
  $\widehat{G}_t \leftarrow U_t \widetilde{N}_t$ // Map update to full space
  $W_{t+1} \leftarrow W_t - \eta_t \widehat{G}_t$ // Parameter update
end for

havior of OASIS in the pretraining regime, we train Llama-130M and Llama-350M from scratch on the C4 dataset (Dodge et al., 2021).

Training Details. All experiments are conducted using bfloat16 (bf16) precision on NVIDIA H200 GPUs. All models are trained using the Adam optimizer. We report task-specific performance metrics: accuracy on GSM8K, pass@10 on HumanEval, and validation loss on C4. In addition, we measure peak memory usage during training to quantify the efficiency gains of OASIS. Please refer to Appendix A for detailed hyperparameters

Baselines. We compare OASIS against diverse training strategies: (i) **Full fine-tuning** with Adam, which serves as the standard upper bound, (ii) **LoRA** (Hu et al., 2021), a parameter-efficient finetuning method that updates low-rank adapters while freezing the base model, (iii) **GaLore** (Zhao et al., 2024), which reduces the memory footprint of optimizer states by projecting gradients into a low-rank subspace, and (iv) **LDAdam** (Robert et al., 2025), which similarly operates in a low-dimensional gradient subspace while incorporating error feedback and projection-aware optimizer to improve performance.

4.2. Main Results

Finetuning. Table 1 compares OASIS with full fine-tuning and prior low-rank methods on GSM8K. We evaluate two ranks to capture the trade-off between compression and performance: a lower rank ($r = 32$) corresponding to an aggressive memory reduction regime, and a higher rank ($r = 128$) chosen such that OASIS closely matches full fine-tuning performance. For Llama-2 7B, OASIS matches the performance of full fine-tuning while reducing peak memory by nearly $2\times$. Across both ranks, it consistently outperforms prior low-rank methods. Notably, even in the low-rank regime ($r = 32$), OASIS outperforms all baselines while using substantially less memory, achieving over 30% lower memory usage compared to the closest performing baseline, LDAdam. Similar trends are observed for the Llama-3.2 1B model. At a lower rank, OASIS remains competitive with prior methods while achieving the lowest memory usage, and at a higher rank, it closely approaches full fine-tuning performance. To better understand the source of memory savings, we present a breakdown of peak memory across different training components in Figure 2 (at rank $r = 32$). Activation memory constitutes the largest portion of the total footprint, followed by optimizer states.

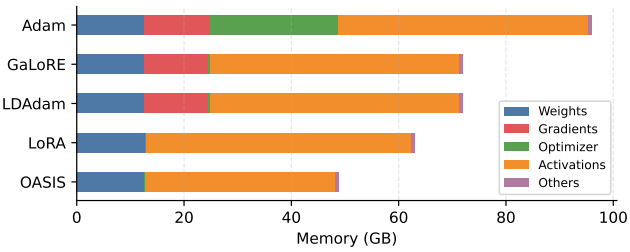


Figure 2. Memory breakdown across training components for different methods on LLaMA-2 7B GSM8K finetuning at rank 32. OASIS reduces memory across activations, gradients, and optimizer states, leading to the lowest overall memory footprint.

Table 1. Finetuning results on GSM8K for Llama-2 7B and Llama-3 1B. For Llama-2 7B, OASIS matches the performance of Adam while reducing peak memory by $\sim 2\times$.

Model	Rank	Method	Accuracy (%)	Peak Memory (GB)
Llama-2 7B	–	Adam	39.37 ± 0.39	95.18
	32	LoRA	37.13 ± 0.64	64.15
		GaLore	35.96 ± 1.18	71.30
		LDAdam	38.74 ± 0.11	71.48
		OASIS	39.14 ± 0.72	48.21
	128	LoRA	37.27 ± 0.36	66.00
		GaLore	37.49 ± 0.56	72.03
		LDAdam	38.17 ± 0.86	72.21
		OASIS	39.10 ± 1.27	49.60
Llama-3 1B	–	Adam	27.09 ± 0.58	52.54
	32	LoRA	22.51 ± 0.46	48.06
		GaLore	21.64 ± 0.36	47.17
		LDAdam	24.55 ± 0.91	49.01
		OASIS	23.78 ± 0.26	41.03
	128	LoRA	26.61 ± 0.62	48.62
		GaLore	25.51 ± 0.61	47.40
		LDAdam	25.72 ± 1.07	49.23
		OASIS	26.88 ± 0.49	41.59

Table 2. Finetuning results on HumanEval for Llama-2 7B (trained on CodeAlpaca). OASIS achieves the best performance among low-rank methods at aggressive compression, and remains competitive with the strongest baseline at higher rank, while consistently using the lowest memory.

Rank	Method	Accuracy (%)	Peak Memory (GB)
–	Adam	34.01 ± 1.29	93.26
4	LoRA	28.34 ± 2.79	61.70
	GaLore	27.42 ± 1.22	69.00
	LDAdam	27.93 ± 1.41	68.99
	OASIS	29.28 ± 0.61	46.16
32	LoRA	31.84 ± 1.74	62.08
	GaLore	31.24 ± 2.74	69.20
	LDAdam	34.20 ± 1.43	69.37
	OASIS	33.50 ± 2.13	46.35

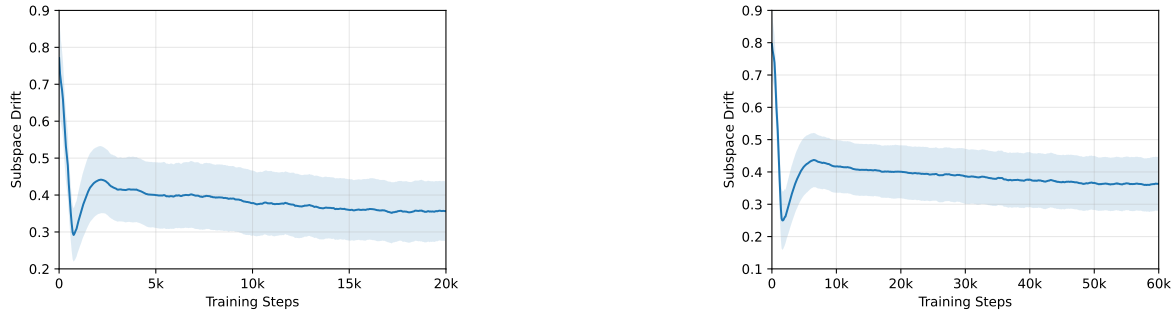
LDAdam and GaLore primarily reduce optimizer memory, while LoRA reduces gradient and optimizer states' memory while leading to a larger activation footprint. In contrast, OASIS reduces memory across activations, gradients, and optimizer states, resulting in the lowest memory footprint.

Table 2 shows results on HumanEval for Llama-2 7B. At lower rank ($r = 4$), OASIS achieves the best accuracy among all low-rank methods while using substantially less memory, highlighting its effectiveness in aggressive compression regimes. At higher rank ($r = 32$), OASIS remains competitive with the strongest baseline (LDAdam) while reducing memory usage by over 30%. These results further demonstrate that OASIS provides strong performance–memory trade-offs, particularly in low-rank settings.

Table 3. Pretraining results on C4 across Llama model scales. OASIS achieves comparable validation loss to Adam while consistently reducing memory usage.

Method	Llama-130M		Llama-350M	
	Val Loss	Mem	Val Loss	Mem
Adam	3.21	41.00	3.03	89.31
GaLore	3.55	40.73	3.39	88.22
LDAdam	3.23	40.73	2.86	88.22
OASIS	3.28	37.10	3.02	79.46

Pretraining. Table 3 shows pretraining results on the C4 dataset for Llama-130M and Llama-350M. OASIS achieves competitive validation loss compared to full training while consistently reducing memory usage across model scales. For Llama-350M, OASIS closely matches the performance of Adam while reducing peak memory by over 10%. For the smaller Llama-130M model, OASIS incurs only a minor increase in validation loss while still providing meaningful memory savings. Overall, these results demonstrate that OASIS scales effectively to pretraining settings, maintaining



(a) Llama 130M

(b) Llama 350M

Figure 3. Subspace drift during pretraining on C4 for Llama-130M and Llama-350M. Both models exhibit a transient phase followed by stabilization at a non-zero drift, indicating that the activation subspace evolves throughout training.

strong performance while reducing memory consumption.

4.3. Ablation Study

How does the activation subspace evolve during training? We analyze the evolution of the activation subspace using a Frobenius-based distance between subspaces. Let $U_{t-1}, U_t \in \mathbb{R}^{d\times r}$ denote orthonormal bases of the rank-r activation subspace at successive steps, and $T_t = U_t^\top U_{t-1}$ denote the subspace transition matrix. We define the subspace drift as:

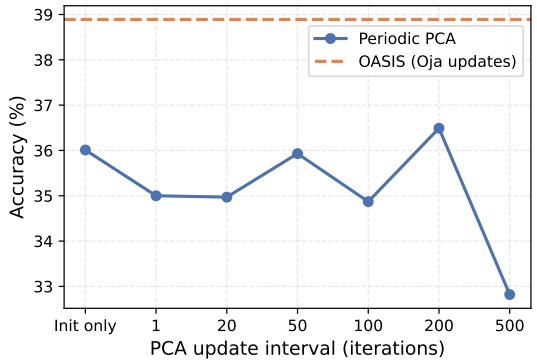
$$\mathrm{Drift}(t) = \sqrt{1 - \frac{\|T_t\|_F^2}{r}}. \tag{6}$$

This metric is bounded in $[0, 1]$, where 0 indicates identical subspaces and 1 indicates orthogonal subspaces. Figure 1b shows the subspace drift during finetuning, while Figure 3 presents the corresponding behavior during pretraining on C4 with Llama-130M and LlaMA-350M. In both settings, the drift remains non-zero throughout training, indicating that the activation subspace does not converge to a fixed basis. Notably, pretraining exhibits a more pronounced transient phase: the drift decreases initially and subsequently settles at a non-zero level, indicating continued evolution of the subspace. These observations suggest that the activation subspace is inherently non-stationary. Consequently, methods that rely on a fixed subspace estimated at initialization or infrequent updates based on batch statistics may fail to capture the evolving structure of activations, motivating the need for continuously updating the subspace.

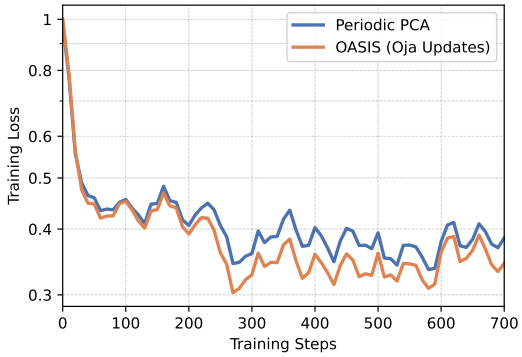
Is online subspace tracking really necessary? A natural question that arises is whether online subspace tracking is necessary, or if similar performance can be achieved by periodically recomputing the subspace using PCA with a carefully tuned update interval. To answer this, we compare OASIS with periodic PCA across a range of update intervals, where smaller intervals correspond to more frequent updates. As shown in Figure 4a, even after tuning the update interval, periodic PCA consistently underperforms OASIS. In addition, Figure 4b shows that OASIS converges faster and achieves lower training loss compared to periodic PCA.These results highlight that relying on periodic recomputation using noisy batch statistics is fundamentally less effective than continuously tracking activation subspaces, underscoring the importance of online subspace learning for memory-efficient training.

How does the performance vary with rank? As shown in Figure 5, performance improves with increasing rank for both periodic PCA and OASIS, reflecting the increased expressivity of higher-dimensional subspaces. However, OASIS consistently outperforms periodic PCA across all ranks. Notably, OASIS achieves substantially higher accuracy even at low ranks, indicating that online subspace tracking yields more effective representations than periodic recomputation. Performance gains begin to saturate beyond moderate ranks, indicating higher-dimensional subspaces are easier to adapt to than more constrained low-rank subspaces undergoing rapid evolution.These results indicate that OASIS effectively tracks the evolving subspace and provides strong performance across a wide range of ranks.

How does the subspace learning rate affect performance? As shown in Figure 6, very small learning rates lead to slow adaptation of the subspace and result in lower accuracy. Increasing the learning rate improves performance, with accuracy peaking at around 0.1. Larger learning rates degrade performance, likely due to instability in the subspace updates. Overall, these results highlight the importance of selecting an appropriate subspace learning rate that balances stability and responsiveness in subspace adaptation. In practice, we find that a moderate value (e.g., 0.1) consistently provides strong performance across tasks.



(a) Effect of PCA update interval



(b) Training loss vs iterations

Figure 4. Comparison of periodic PCA and OASIS during finetuning on GSM8K with Llama-2 7B. (a) Periodic PCA consistently underperforms OASIS across all tested update intervals. (b) OASIS also converges faster, achieving lower training loss compared to periodic PCA.

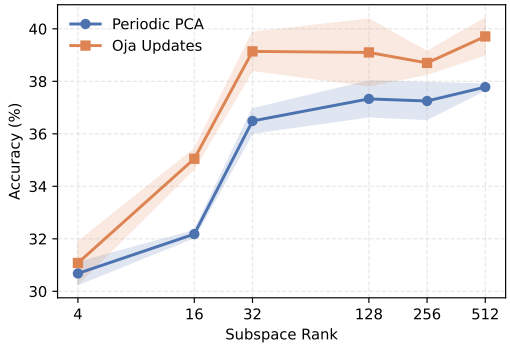


Figure 5. Effect of subspace rank on finetuning performance on GSM8K with Llama-2 7B. OASIS consistently outperforms periodic PCA across ranks.

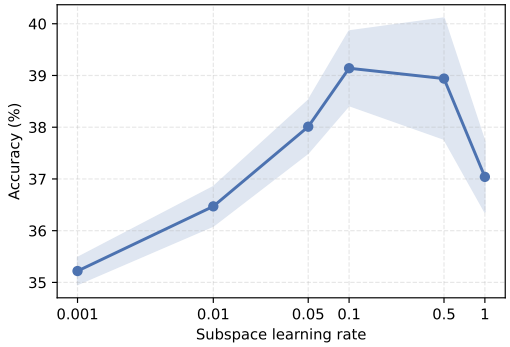


Figure 6. Effect of subspace learning rate on finetuning performance on GSM8K with Llama-2 7B. Best accuracy is achieved at 0.1 LR.

5. Conclusion

In this paper, we present OASIS, an online activation subspace learning algorithm for memory-efficient training that continuously tracks low-dimensional activation subspaces through an iterative update rule. Across both pretraining and finetuning settings, OASIS achieves strong performance while reducing memory footprint compared to prior low-rank training techniques. However, realizing these gains requires tuning the subspace learning rate (LR), which governs the dynamics of the online updates. While we observe a reasonably stable range of effective LR values, performance can degrade outside this range, making it an important hyperparameter. A natural direction for future work is to develop adaptive strategies for controlling this learning rate based on subspace dynamics, for example by increasing it when the subspace is evolving rapidly and decreasing it as it stabilizes. More broadly, our work opens up avenues for exploring adaptive and data-driven mechanisms for subspace tracking in low-rank training.

Impact Statement

This work focuses on improving the memory efficiency of training large language models by reducing the storage requirements of activations, gradients, and optimizer states. By enabling training in a compressed representation without modifying model architectures, OASIS can lower the hardware barriers associated with large-scale model development and make advanced training techniques more accessible to a broader range of practitioners. It may also contribute to reducing the computational and energy costs of training, particularly in long-context and large-batch regimes.

References

Apolinario, M. P. E. and Roy, K. Lance: Low rank activation compression for efficient on-device continual learning, 2026. URL https://arxiv.org/abs/2509.21617.

Ashkboos, S., Croci, M. L., do Nascimento, M. G., Hoefler, T., and Hensman, J. SliceGPT: Compress large language models by deleting rows and columns. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=vXxardq6db.

Biderman, D., Portes, J., Ortiz, J. J. G., Paul, M., Greengard, P., Jennings, C., King, D., Havens, S., Chiley, V., Frankle, J., et al. Lora learns less and forgets less. *arXiv preprint arXiv:2405.09673*, 2024.

Chang, C.-C., Lin, W.-C., Lin, C.-Y., Chen, C.-Y., Hu, Y.-F., Wang, P.-S., Huang, N.-C., Ceze, L., Abdelfattah, M. S., and Wu, K.-C. Palu: KV-cache compression with low-rank projection. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=LWMS4pk2vK.

Chaudhary, S. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca, 2023.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms, 2023. URL https://arxiv.org/abs/2305.14314.

Dodge, J., Sap, M., Marasović, A., Agnew, W., Ilharco, G., Groeneveld, D., Mitchell, M., and Gardner, M. Documenting large webtext corpora: A case study on the colossal clean crawled corpus, 2021. URL https://arxiv.org/abs/2104.08758.

et al., A. G. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

et al., H. T. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

et al., M. C. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Hao, Y., Cao, Y., and Mou, L. Flora: Low-rank adapters are secretly gradient compressors, 2024. URL https://arxiv.org/abs/2402.03293.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Korthikanti, V., Casper, J., Lym, S., McAfee, L., Andersch, M., Shoeybi, M., and Catanzaro, B. Reducing activation recomputation in large transformer models, 2022. URL https://arxiv.org/abs/2205.05198.

Lialin, V., Shivagunde, N., Muckatira, S., and Rumshisky, A. Relora: High-rank training through low-rank updates, 2023. URL https://arxiv.org/abs/2307.05695.

Liang, K., Liu, B., Chen, L., and Liu, Q. Memory-efficient llm training with online subspace descent, 2024. URL https://arxiv.org/abs/2408.12857.

Lin, C.-H., Gao, S., Smith, J. S., Patel, A., Tuli, S., Shen, Y., Jin, H., and Hsu, Y.-C. ModeGPT: Modular decomposition for large language model compression. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=8EfxjTCg2k.

Liu, S.-Y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., and Chen, M.-H. Dora: Weight-decomposed low-rank adaptation, 2024. URL https://arxiv.org/abs/2402.09353.

Liu, Z., Zhang, R., Wang, Z., Yan, M., Yang, Z., Hovland, P., Nicolae, B., Cappello, F., Tang, S., and Zhang, Z. Cola: Compute-efficient pre-training of llms via low-rank activation, 2025. URL https://arxiv.org/abs/2502.10940.

Miles, R., Reddy, P., Elezi, I., and Deng, J. Velora: Memory efficient training using rank-1 sub-token projections, 2024. URL https://arxiv.org/abs/2405.17991.

Modoranu, I.-V., Safaryan, M., Malinovsky, G., Kurtic, E., Robert, T., Richtarik, P., and Alistarh, D. Microadam: Accurate adaptive optimization with low space overhead and provable convergence, 2024. URL https://arxiv.org/abs/2405.15593.

Oja, E. The nonlinear pca learning rule in independent component analysis. *Neurocomputing*, 17(1):25–45, 1997.

Rajabi, S., Nonta, N., and Rambhatla, S. Subtrack++ : Gradient subspace tracking for scalable llm training, 2025. URL https://arxiv.org/abs/2502.01586.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: Memory optimizations toward training trillion parameter models, 2020. URL https://arxiv.org/abs/1910.02054.

Robert, T., Safaryan, M., Modoranu, I.-V., and Alistarh, D. Ldadam: Adaptive optimization from low-dimensional gradient statistics, 2025. URL https://arxiv.org/abs/2410.16103.

Saxena, U., Saha, G., Choudhary, S., and Roy, K. Eigen attention: Attention in low-rank space for KV cache compression. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 15332–15344, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.899. URL https://aclanthology.org/2024.findings-emnlp.899/.

Saxena, U., Sharify, S., Roy, K., and Wang, X. Resq: Mixed-precision quantization of large language models with low-rank residuals. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=4qIP1sXcR1.

Shamshoum, Y., Hodos, N., Sieradzki, Y., and Schuster, A. CompAct: Compressed activations for memory-efficient LLM training. In Chiruzzo, L., Ritter, A., and Wang, L. (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1511–1524, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.71. URL https://aclanthology.org/2025.naacl-long.71/.

Singhania, P., Singh, S., He, S., Feizi, S., and Bhatele, A. Loki: Low-rank keys for efficient sparse attention, 2024. URL https://arxiv.org/abs/2406.02542.

Xia, W., Qin, C., and Hazan, E. Chain of lora: Efficient fine-tuning of language models via residual learning, 2024. URL https://arxiv.org/abs/2401.04151.

Yuan, Z., Shang, Y., Song, Y., Yang, D., Wu, Q., Yan, Y., and Sun, G. ASVD: Activation-aware singular value decomposition for compressing large language models, 2025. URL https://openreview.net/forum?id=HyPofygOCT.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization, 2017. URL https://arxiv.org/abs/1611.03530.

Zhang, Q., Chen, M., Bukharin, A., Karampatziakis, N., He, P., Cheng, Y., Chen, W., and Zhao, T. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning, 2023. URL https://arxiv.org/abs/2303.10512.

Zhang, R., Wang, K., Liu, L., Wang, S., Cheng, H., Zhang, C., and Shen, Y. Lorc: Low-rank compression for llms kv cache with a progressive compression strategy, 2024. URL https://arxiv.org/abs/2410.03111.

Zhao, J., Zhang, Z., Chen, B., Wang, Z., Anandkumar, A., and Tian, Y. Galore: Memory-efficient llm training by gradient low-rank projection, 2024. URL https://arxiv.org/abs/2403.03507.

A. Hyperparameters

Table 4. Hyperparameters for finetuning experiments on GSM8K and HumanEval.

Hyperparameter	**GSM8K (Table 1)**	**HumanEval (Table 2)**
Batch size	32	16
Sequence length	512	1024
Learning rate scheduler	Cosine	Cosine
Learning rate (LR)	{2e-4, 5e-4, 9e-4, 1e-3, 1e-2}	{2e-4, 5e-4, 9e-4, 1e-3, 1e-2}
Warmup steps	5%	5%
Epochs	3	3
OASIS Subspace LR (γ)	$\{0.001, 0.01, 0.1\}$	$\{0.001, 0.01, 0.05, 0.1\}$

Table 5. Hyperparameters for pretraining experiments on C4 presented in Table 3.

Hyperparameter	**Llama-130M**	**Llama-350M**
Batch size	512	512
Sequence length	256	256
Training iterations	20k	60k
Learning rate (LR)	{5e-4, 9e-4, 1e-3, 5e-3, 1e-2}	{5e-4, 9e-4, 1e-3, 5e-3, 1e-2}
Learning rate scheduler	Cosine	Cosine
Warmup	10%	10%
OASIS subspace LR (γ)	$\{0.001, 0.01, 0.1\}$	$\{0.001, 0.01, 0.1\}$

Table 6. Model configurations for pretraining experiments.

Model	Hidden	Intermediate	Heads	Layers
Llama-130M	768	2048	12	12
Llama-350M	1024	2736	16	24

We summarize the hyperparameters for finetuning and pretraining experiments in Tables 4 and 5, respectively. For finetuning on GSM8K and HumanEval, we use a cosine learning rate schedule with 5% warmup and train for 3 epochs, tuning the learning rate and OASIS subspace learning rate across a fixed grid.

For pretraining on C4, we use a shared configuration across model scales, including batch size, sequence length, optimizer settings, and learning rate schedules. The primary difference lies in the number of training iterations, where LLaMA-130M and LLaMA-350M are trained for 20k and 60k steps, respectively. These iteration counts are chosen based on Chinchilla scaling laws to ensure compute-efficient training across model sizes (Hoffmann et al., 2022). Please refer to Table 6 for details on model architectures.

Hyperparameters for baseline methods are selected based on the configurations reported in their respective papers (Hu et al., 2021; Zhao et al., 2024; Robert et al., 2025).