SALR: Sparsity-Aware Low-Rank Representation for Efficient Fine-Tuning of Large Language Models

Longteng Zhang[1], Sen Wu[3], Shuai Hou[3], Zhengyu Qing[3], Zhuo Zheng[4], Danning Ke[4], Qihong Lin[4], Qiang Wang[3*], Shaohuai Shi[3], Xiaowen Chu[1,2*]

[1]The Hong Kong University of Science and Technology (GuangZhou)
[2]The Hong Kong University of Science and Technology
[3]Harbin Institute of Technology, Shenzhen
[4]Huawei Technologies
lzhang330@connect.hkust-gz.edu.cn, {24S151068, houshuai, 210110609}@stu.hit.edu.cn
{zhengzhuo2, kedanning, linqihong}@huawei.com, {qiang.wang, shaohuais}@hit.edu.cn, xwchu@ust.hk

Abstract

Adapting large pre-trained language models to downstream tasks often entails fine-tuning millions of parameters or deploying costly dense weight updates, which hinders their use in resource-constrained environments. Low-rank Adaptation (LoRA) reduces trainable parameters by factorizing weight updates, yet the underlying dense weights still impose high storage and computation costs. Magnitude-based pruning can yield sparse models but typically degrades LoRA's performance when applied naively. In this paper, we introduce SALR (Sparsity-Aware Low-Rank Representation), a novel fine-tuning paradigm that unifies low-rank adaptation with sparse pruning under a rigorous mean-squared-error framework. We prove that statically pruning only the frozen base weights minimizes the pruning error bound, and we recover the discarded residual information via a truncated-SVD low-rank adapter, which provably reduces per-entry MSE by a factor of $(1 - r/\min(d, k))$. To maximize hardware efficiency, we fuse multiple low-rank adapters into a single concatenated GEMM, and we adopt a bitmap-based encoding with a two-stage pipelined decoding+GEMM design to achieve true model compression and speedup. Empirically, SALR attains 50% sparsity on various LLMs while matching the performance of LoRA on GSM8K and MMLU, reduces model size by $2\times$, and delivers up to a $1.7\times$ inference speedup.

Introduction

The rapid growth of large-scale neural networks has driven remarkable advances across natural language processing, computer vision, and other machine learning domains (Meta 2025; Dubey and et al. 2024; Yang et al. 2025; DeepSeek-AI 2025; OpenAI 2023). However, adapting these massive models to downstream tasks often requires fine-tuning millions or even billions of parameters, posing significant challenges for deployment in resource-constrained environments (Mangrulkar et al. 2022; Lester, Al-Rfou, and Constant 2021; Li and Liang 2021). Low-Rank Adaptation (LoRA) (Hu et al. 2022) has emerged as a lightweight fine-tuning paradigm, representing weight updates as a product of two much smaller matrices. While LoRA drastically reduces the number of trainable parameters, the underlying dense structure of the original weights still incurs substantial memory and computation overhead (Dettmers et al. 2023; Zhang et al. 2023a).

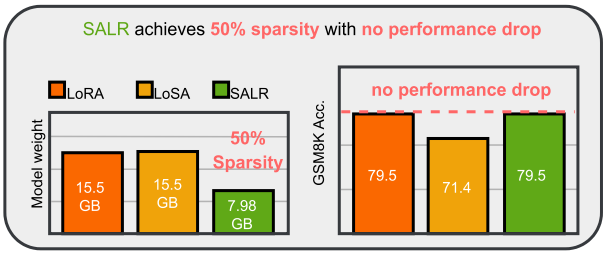


Figure 1: Memory-accuracy trade-off on the GSM8K (Cobbe et al. 2021) benchmark for Llama3-8B (Dubey and et al. 2024), fine-tuned on MetaMath (Yu et al. 2023). At 50% sparsity, SALR maintains the dense LoRA (Hu et al. 2022) baseline accuracy (79.5%) while reducing model size from 15.5 GB to 7.98 GB. In contrast, LoSA (Huang et al. 2025) at 50% sparsity suffers a drop to 71.4% accuracy.

Model pruning offers a complementary approach by eliminating redundant weights to yield sparse networks with reduced storage and faster inference (Han, Mao, and Dally 2016; Frantar and Alistarh 2023; Sun et al. 2024; Guo et al. 2023). Ideally, magnitude-based pruning applied to a LoRA-fine-tuned model should combine the benefits of sparsity and low-rank adaptation. However, pruning can disrupt the carefully learned low-rank subspace, and existing dynamic masking strategies often lead to significant performance degradation (Huang et al. 2025). To guide effective pruning in the LoRA setting, we develop a unified theoretical framework that quantifies the mean-squared error (MSE) induced by different pruning schemes. Our analysis reveals that a static mask applied solely to the frozen base weights achieves the lowest error bound among both static and dynamic approaches, laying the foundation for our Sparsity-Aware Low-Rank Representation fine-tuning (SALR) method.

Although static pruning of the base weights minimizes theoretical error, it discards valuable information residing in

*Corresponding author

the pruned elements. To address this loss, SALR introduces a sparsity preservation pruning strategy: rather than permanently zeroing out pruned entries, we capture their residual in an auxiliary low-rank adapter. By applying a truncated singular value decomposition (SVD) to the sparse residual, we have a compact rank-r correction that retains essential information without inflating the parameter count. We further show that this low-rank correction provably reduces the per-entry MSE by a factor of $\big(1 - \frac{r}{\min(d,k)}\big)$, where d and k are the input and output dimensions. However, in this setting, SALR requires the deployment of two or more low-rank adapters, one corresponding to the LoRA fine-tuning module and others serving as sparsity preservation modules. Performing the small matrix multiplications for each adapter sequentially results in under-utilization of modern hardware accelerators. To address this inefficiency, SALR adopts an adapter concatenation scheme: by stacking the low-rank matrices along their rank dimension, all adapter updates are consolidated into a single, larger GEMM (general matrix-matrix multiplication) operation on a shared input. This fusion reduces kernel-launch overhead and maximizes computational throughput when multiple adapters operate on the same input vector.

In recent LoRA-based pruning studies, although most works claim to achieve a specified sparsity ratio and the associated speedup, almost none address the challenge of actual model size compression. Thus, the achieved sparsity does not necessarily translate into reduced model size (Huang et al. 2025; Khaki et al. 2025; Agarwalla et al. 2024; Kurtic et al. 2023). This limitation is a major reason why sparse methods lag behind quantization in model compression (Fan et al. 2025; Xia et al. 2023). Therefore, to enable effective model compression via pruning, SALR integrates a bitmap-based encoding for the pruned base weights along with a two-stage pipelined design. By employing bitmap encoding, SALR efficiently stores the dense elements of sparse weights and enables high-performance bitmap decoding. The two-stage pipeline further alleviates decoding overhead: in the first stage, byte-level masks and lookup tables reconstruct sparse submatrices efficiently, while the LoRA module participates in GEMM computation; concurrently, the second stage feeds these reconstructed blocks into high-performance GEMM kernels. In this manner, the two-stage pipeline sustains compute-bound density throughout all computation phases.

In summary, our contributions are:

- We develop a unified, MSE-based theoretical framework for pruning in LoRA-fine-tuned models and prove that applying a static mask to the frozen base weights yields the lowest error bound.
- We introduce SALR, a sparsity-preservation pruning method that captures pruned-weight residuals via a truncated-SVD low-rank adapter and provably reduces per-entry MSE.
- We propose an adapter concatenation scheme that stacks all low-rank adapters into a single GEMM operation, minimizing kernel-launch overhead and maximizing hardware accelerator utilization.
- We design a practical compression and deployment pipeline using bitmap encoding for actual model-size reduction and a two-stage, pipelined decoding+GEMM approach to sustain compute-bound throughput.

Preliminary

In this section, we present the essential background and theoretical underpinnings necessary to understand the challenges and methodologies associated with Sparsity-Aware Low-Rank Representation fine-tuning (SALR).

Let the weight matrix be denoted as $W \in \mathbb{R}^{d\times k}$, where d represents the number of input features and k denotes the number of output classes. The magnitude-based pruning operation can be formulated as

$$\hat{W}_{ij} = \begin{cases} W_{ij}, & |W_{ij}| > T_p, \\ 0, & |W_{ij}| \le T_p, \end{cases}$$

where T_p is the pruning threshold determined by the desired sparsity ratio p. In this manner, pruning can significantly reduce the memory footprint and computational cost of the model, thereby enabling the deployment of large-scale models in resource-constrained environments. In the following, we analyze the error introduced by pruning and discuss how it can be mitigated through low-rank adaptation techniques.

Theorem 1. *Let $W \sim \mathcal{N}(0,\sigma^2)$ and for a given pruning ratio $p \in [0,1)$ we choose the threshold T_p such that*

$$P\big(|W| \le T_p\big) = p, \quad \Longrightarrow \quad T_p = \sigma\Phi^{-1}\Big(\tfrac{1+p}{2}\Big).$$

Then the mean-squared error (MSE) of this pruning is

$$\mathrm{MSE}(p) = \mathbb{E}\big[(W-\hat{W})^2\big] = 2\sigma^2\Big[\Phi(t_p) - \tfrac{1}{2} - t_p\varphi(t_p)\Big],$$

where $t_p = \Phi^{-1}\big(\frac{1+p}{2}\big)$ and $\varphi(t) = \frac{1}{\sqrt{2\pi}}e^{-t^2/2}$ is the standard normal PDF.

Based on the theorem above, if we prune 50% of the entries in the weight matrix, we have

$$t_{0.5} = \Phi^{-1}\Big(\tfrac{1+0.5}{2}\Big) = \Phi^{-1}(0.75) \approx 0.674,$$

and the MSE is given by

$$\mathrm{MSE}(0.5) = 2\sigma^2\Big[\Phi(0.674) - \tfrac{1}{2} - 0.674\varphi(0.674)\Big].$$

Since $\Phi(0.674) = 0.75$ and $\varphi(0.674) \approx 0.318$, we find

$$\begin{aligned}\mathrm{MSE}(0.5) &= 2\sigma^2\big[0.75 - 0.5 - 0.674\cdot 0.318\big] \\ &\approx 2\sigma^2(0.25 - 0.214) \approx 0.072\sigma^2.\end{aligned}$$

This result demonstrates that pruning a substantial proportion of model parameters leads to only a modest increase in MSE, thereby establishing the foundational performance guarantees for applying pruning as an effective strategy for large model compression.

However, the pruning operation in LoRA fine-tuning differs in several respects. Due to the introduction of LoRA, the model parameters are represented as a linear combination of low-rank matrices and the original weights, i.e.,

$W = W_0 + AB$, which implies that pruning may have to account for the effects of the low-rank structure. Specifically, the pruning procedure should aim to reduce the number of parameters while preserving the integrity of the low-rank structure as much as possible. A typical approach is to prune both W and the subspaces of the low-rank matrices A and B (Huang et al. 2025). This strategy ensures that the sparse, low-rank adapter is effectively integrated into the pruned model without altering the model's sparsity after training. However, because pruning is applied to the low-rank subspace, which is designed to compensate for the loss of parameters in the original model, this approach often results in significant performance degradation compared to LoRA.

In addition to pruning both A and B simultaneously, a simpler alternative is to prune only W_0, that is, the frozen original weights. The advantage of this approach is that it minimizes the impact of pruning on the low-rank matrices A and B, thereby better preserving the low-rank structure of the model. Furthermore, the above methods can be combined, such as deriving a pruning mask from AB and subsequently applying it to W_0, which allows for compression of the base model weights while taking the low-rank structure into consideration.

Consider a single pruning step during LoRA fine-tuning, where the objective is to minimize the distance between W and $\hat{W}$, with W and $\hat{W}$ denoting the weights before and after pruning, respectively. In the context of LoRA fine-tuning, $W = W_0 + AB$, where W_0 is the frozen original weight matrix and A and B are low-rank matrices. Given the objective, we derive the pruning error bound for such various methods as follows.

Theorem 2. *Let $W_{0,ij} \sim \mathcal{N}(0, \sigma^2)$ and $\Delta_{ij} = (A^*B^*)_{ij} \sim \mathcal{N}(0, \tau^2)$ be independent, where A^* and B^* be the optimal values of A and B; $U_{ij} = W_{0,ij} + \Delta_{ij} \sim \mathcal{N}(0, V^2)$ with $V^2 = \sigma^2 + \tau^2$; the global prune rate is $p \in [0, 1)$, and define the prune operation as the same as above. Let $E_1(p), E_2(p), E_3(p)$ be the per-entry MSEs of*

- ***Method 1 (static mask on W_0)**: prune the smallest entries of $|W_0|$ at rate p.*
- ***Method 2 (dynamic mask driven by U, but prune only W_0)**.*
- ***Method 3 (dynamic mask on the full $U = W_0 + \Delta$)**.*

Then the MSEs are

$$E_1(p) = 2\sigma^2 \mathcal{Q}(t_p), \tag{1}$$

$$E_2(p) = \frac{\sigma^2\tau^2}{\sigma^2 + \tau^2} p + 2\frac{\sigma^4}{\sigma^2 + \tau^2} \mathcal{Q}(t_p) \tag{2}$$

$$E_3(p) = 2(\sigma^2 + \tau^2)\mathcal{Q}(t_p) \tag{3}$$

where $\mathcal{Q}(t)$ is defined as

$$\mathcal{Q}(t) \coloneqq \Phi(t) - \tfrac{1}{2} - t\varphi(t).$$

Moreover, for every p,

$$E_1(p) \le E_3(p) \le E_2(p).$$

Method	Performance	Model	Speedup
LoSA (ICLR2025)	Low	**Sparse**	**Y**
SparseLoRA (ICML2025)	**High**	Dense	N
SALR (ours)	**High**	**Sparse**	**Y**

Table 1: Comparison of LoSA, SparseLoRA and SALR across three criteria: *Performance* indicates end-task accuracy relative to a dense LoRA baseline; *Model* denotes whether the fine-tuned weights remain dense or become sparse; and *Speedup* reports whether true throughput gains for inference are realized. LoSA applies static pruning to achieve sparsity and speedup at the cost of performance; SparseLoRA retains a dense model to match LoRA accuracy but cannot compress or accelerate inference; SALR combines sparse pruning with low-rank residual adapters to deliver both high accuracy and genuine inference speedup.

System Efficiency Limitations in Existing Work

Table 1 compares the latest pruning methods leveraging low-rank adaptation to ours. LoSA and SparseLoRA diverge significantly in both their algorithms and system-level design. LoSA utilizes the third pruning approach described above: it simultaneously prunes both W and the low-rank modules, ensuring the final merged model maintains global sparsity. However, as indicated by our earlier analysis, LoSA's error bounds are relatively high, resulting in only marginal performance retention compared to the original, unpruned model.

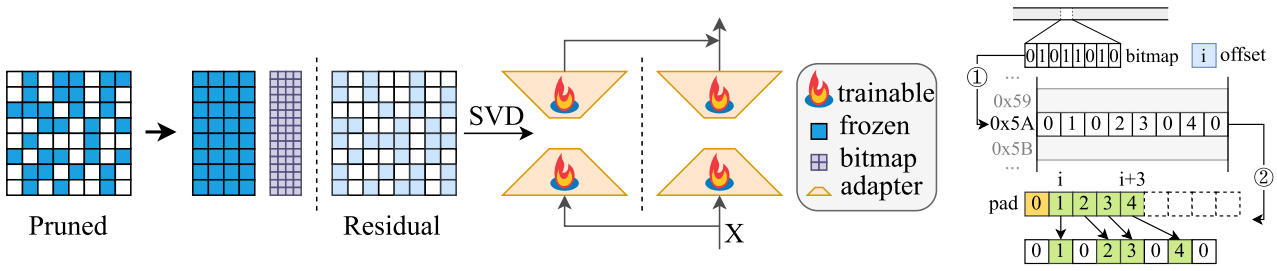
In contrast, SparseLoRA uses a dynamic pruning scheme, selecting which elements of W to compute in response to each input during training. This leads to faster fine-tuning, and by not pruning the low-rank subspace, its accuracy matches that of standard LoRA. However, SparseLoRA does not actually prune the deployed model: at inference time, the model remains dense and thus offers no speedup or compression benefits during deployment, i.e., the improvements exist only during training.

Methodology

In this section, we detail the design of our SALR. Based on the error analysis presented in the previous section, we adopt method 1 as our pruning strategy, which achieves the lowest error bound by applying a static mask solely to W_0. However, although this approach reduces the error bound, it does not take into account the characteristics of the low-rank structure during pruning. Moreover, it still introduces errors, as pruned elements are typically discarded and their information is lost. To address this limitation, we propose a sparsity preservation pruning strategy, in which the information of the pruned elements is retained. Specifically, during pruning, the pruned elements are stored in an additional low-rank adapter, allowing this information to be leveraged in subsequent training stages. In this manner, we are able to preserve information pertinent to the low-rank structure while pruning, thereby enhancing overall model performance.

Sparsity Preservation Pruning

Decomposing the full-parameter weight matrix W into low-rank pairs is an effective approach to reducing the number



(a) The overall pipeline of SALR (b) Bitmap decoding

Figure 2: Overview of SALR. (a) SALR first prunes the base model, resulting in a pruned module and a corresponding residual matrix. Only the bitmap and the dense entries of the pruned module are stored. Subsequently, SALR decomposes the residual into a single low-rank pair using the optimal rank-r approximation. (b) Bitmap decoding of sparse weights.

of parameters while maintaining model performance(Li et al. 2025; Meng, Wang, and Zhang 2024; Liu et al. 2024; Zhang et al. 2023b; Büyükakyüz 2024). Given the pruned weight matrix $\hat{W}$, the residual matrix can be defined as $E = W - \hat{W}$. The objective is to obtain a low-rank approximation of E that preserves essential residual information. This is achieved by applying a truncated singular value decomposition (SVD) to E and retaining only the top r singular values. Since the residual matrix E is sparse and often contains negligible information, such decomposition can substantially reduce the parameter count while still capturing the essential information required for model performance. Furthermore, we demonstrate that applying a rank-r low-rank correction to the pruned residual matrix E can effectively bound the mean squared error (MSE) per entry.

Theorem 3. *Under the same assumptions as Theorem 1, let the residual matrix $E = W - \hat{W}$. Then*

$$\mathbb{E}\|E\|_F^2 = dk\mathrm{MSE}(p).$$

Now let E_r be the best rank-r approximation to E (i.e. its truncated SVD), $q = \min(d, k)$ and the singular values of E as $\sigma_1 \geq \cdots \geq \sigma_q$, the Eckart–Young theorem (Eckart and Young 1936) gives

$$\|E - E_r\|_F^2 = \sum_{i=r+1}^{q} \sigma_i^2.$$

In the worst case (uniformly distributed spectrum) one has $\sum_{i=r+1}^{q} \sigma_i^2 \leq \frac{q-r}{q} \sum_{i=1}^{q} \sigma_i^2$. Taking expectations,

$$\mathbb{E}\|E - E_r\|_F^2 \leq \left(1 - \frac{r}{q}\right)\mathbb{E}\|E\|_F^2 = (dk)\left(1 - \frac{r}{\min(d,k)}\right)\mathrm{MSE}(p).$$

Dividing by dk yields the per-entry bound

$$\mathrm{MSE}_{\text{prune+SVD}}(p, r) \leq \left(1 - \frac{r}{\min(d,k)}\right)\mathrm{MSE}(p).$$

According to Theorem 3, this SVD residual can closely approximate the pruning error. However, if this residual is fixed during training and only the LoRA adapters A and B are fine-tuned, then the subspaces of A and B must compensate for at least two aspects: *the useful structure that was pruned from W*, in addition to *any further task-specific adaptation required*. This burden may be excessive for a small adapter. It is important to note that the subspace of the SVD residual encompasses not only the residual itself; by appropriately adjusting the residual, we can compensate the pruned $\hat{W}$ with global information without affecting its sparsity. Thus, it is necessary to fine-tune not only A and B to enhance task-specific performance, but also to adjust the SVD residual in order to control task-specific sparsity. Finally, we derive the optimal learning rate for updating the SVD-residual matrix during fine-tuning, as shown in Theorem 4.

Theorem 4. *Fix LoRA adapters A, B and let the residual update subproblem be*

$$L(M) = \frac{1}{2}\|XM - R\|_F^2, R = Y - X\left(\hat{W} + AB\right),$$

where X and Y be the input and target output respectively, M represents the optimization variable for the residual.

Then L is convex with

$$\nabla_M L = X^\top(XM - R), \mathrm{Hess}_M L = I_k \otimes (X^\top X),$$

and its gradient is Lipschitz with constant

$$L_{\text{SVD}} = \lambda_{\max}(I_k \otimes X^\top X) = \sigma_{\max}(X)^2.$$

Hence, gradient descent converges for any step-size

$$0 < \eta < \frac{2}{L_{\text{SVD}}} = \frac{2}{\sigma_{\max}(X)^2},$$

and the choice minimizing the worst-case contraction factor is

$$\eta_{\text{SVD}}^* = \frac{1}{\lambda_{\max}(X^\top X)} = \frac{1}{\sigma_{\max}(X)^2}.$$

In practice we estimate $\sigma_{\max}(X)$ by a few power-iterations on a representative mini-batch every epoch, and then set $\eta_{\text{SVD}} \approx 1/\left(\sigma_{\max}(X)^2\right)$ (or more conservatively, half this value) to guarantee stable and efficient convergence of the SVD-residual updates.

Concatenating Multi-LoRA adapters

By concatenating all adapters into A_{cat} and B_{cat}, we replace $2n$ small matrix multiplications with two larger ones, thereby reducing kernel-launch overhead and improving hardware

utilization whenever the same x is fed to multiple LoRA adapters.

Specifically, let the input $x \in \mathbb{R}^{d_{\text{in}}}$, and let the i-th LoRA Adapter be defined as

$$A_i \in \mathbb{R}^{d_{\text{in}} \times r}, B_i \in \mathbb{R}^{r \times d_{\text{out}}}.$$

The incremental computation for a single adapter is given by $\Delta y_i = xA_iB_i = (xA_i)B_i \in \mathbb{R}^{1 \times d_{\text{out}}}$. If the updates are accumulated sequentially, this requires $2n$ small matrix multiplications. To improve computational efficiency, we define the concatenated matrices as

$$A_{\text{cat}} = \begin{bmatrix} A_1^\top \\ A_2^\top \\ \vdots \\ A_n^\top \end{bmatrix}^\top \in \mathbb{R}^{d_{\text{in}} \times (nr)}, B_{\text{cat}} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} \in \mathbb{R}^{(nr) \times d_{\text{out}}}.$$

With this construction, only two matrix multiplications are required to obtain the cumulative update from all adapters:

$$\Delta y = \sum_{i=1}^{n} (xA_i)B_i \in \mathbb{R}^{1 \times d_{\text{out}}}.$$

The final output is then given by $y = xW + \Delta y$.

Mapping Sparse Weights and Pipeline Design

Traditional CSR-format sparse representations incur significant indexing overhead. In contrast, bitmap encoding largely eliminates this cost, though it requires an efficient decoding mechanism to avoid throughput overhead. We show that this decoding process can be highly parallelized.

Mapping Sparse Weights. We define the bitmap as

$$B \in \{0,1\}^{d_{\text{in}} \times d_{\text{out}}}, \quad B_{ij} = \begin{cases} 1, & \hat{W}_{ij} \neq 0, \\ 0, & \hat{W}_{ij} = 0. \end{cases}$$

All nonzero elements are stored in a compact array $v \in \mathbb{R}^{\text{nnz}(\hat{W})}$ in row-major order, where $\text{nnz}(\hat{W}) = \sum_{i,j} B_{ij}$.

To accelerate sparse matrix reconstruction, we partition each row into several byte blocks, with every 8 columns forming one group. For the b-th byte block of row i, we define the bitmask as $mask_{i,b} = \sum_{t=0}^{7} B_{i,8b+t} 2^t \in \{0, \ldots, 255\}$. Let $\text{bit}_t(m)$ denote the t-th binary digit of integer m:

$$\text{bit}_t(m) = \left\lfloor (m \div 2^t) \right\rfloor \bmod 2, \quad t = 0, \ldots, 7,$$

then $mask_{i,b} = \sum_{t=0}^{7} \text{bit}_t\big(B_{i,8b+t}\big) 2^t$. We further define the popcount function as

$$\begin{aligned} \text{popcount}(m) &= \sum_{t=0}^{7} \text{bit}_t(m) \\ &= \#\big\{t : 0 \leq t \leq 7, \text{bit}_t(m) = 1\big\}, \end{aligned}$$

which returns the number of 1's in the binary representation of m. Thus, the number of nonzero elements in the b-th byte block of row i is

$$k_{i,b} = \text{popcount}(mask_{i,b}),$$

corresponding to a segment in the compact array, $v_{i,b} = (v_{i,b}^0, v_{i,b}^1, \ldots, v_{i,b}^{k_{i,b}-1})$.

We precompute a lookup table $\text{LUT} : \{0, \ldots, 255\} \to \{-1, 0, 1, \ldots, 7\}^8$, such that for any $mask$, if its t-th bit is 1, then $\text{LUT}(mask)_t$ gives the corresponding index in the nonzero segment $v_{i,b}$; otherwise, it is -1. Let

$$\text{LUT}(mask) = (\ell_0, \ldots, \ell_7),$$

then the sparse reconstruction rule can be expressed as

$$\forall t = 0, \ldots, 7 : \quad \hat{W}_{i,8b+t} = \begin{cases} v_{i,b}^{\ell_t}, & \ell_t \geq 0, \\ 0, & \ell_t = -1. \end{cases}$$

Pipeline Design. To fully exploit hardware capabilities, we decouple sparse bitmap decoding from GEMM computation and construct a two-stage pipeline. During the decoding stage, CUDA cores sequentially read the bitmap B and compact array v by byte block, and reconstruct contiguous sparse submatrix blocks using the precomputed lookup table LUT. In the computation stage, the recovered submatrix blocks are processed by invoking Tensor Core to perform dense–dense multiplication. The two stages are connected via a ring buffer: while the decoding stage processes block $b + 1$, the computation stage can simultaneously execute the multiplication for block b. This design eliminates blocking due to sparse format conversion during the main computation, maximizes hardware throughput, and significantly reduces memory access and scheduling overhead.

Experimental Results

We conduct extensive experiments to evaluate the effectiveness of our SALR across a range of benchmarks and models. First, we focus on supervised fine-tuning (SFT) of state-of-the-art LLMs in the MATH and multidisciplinary domains, comparing the performance of SALR against its competitors, i.e., LoRA-based pruning methods, on MMLU and GSM8K benchmarks. Additionally, we examine the system efficiency of various methods during fine-tuning, demonstrating that SALR significantly reduces model size while maintaining fine-tuning throughput. The models, datasets, metrics, and baselines used in our experiments are detailed below, with further experimental settings provided in supplementary due to space constraints.

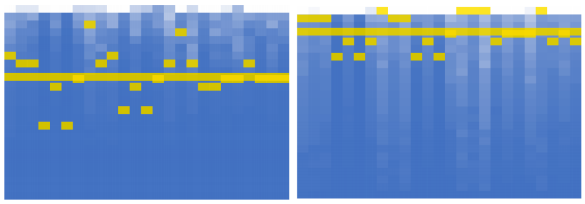
Models. We fine-tune a diverse selection of LLMs, including Llama2-7B (Touvron et al. 2023), Llama3-8B (Dubey and et al. 2024), Mixtral-8x7B (Jiang et al. 2024), DeepSeek-V2-Lite (DeepSeek-AI 2025).

Datasets. In alignment with prior studies (Wang, Yu, and Li 2024; Wang et al. 2024), our experiments span a variety of datasets tailored to specific task types. We utilize MetaMath (Yu et al. 2023) for the MATH domain, auxiliary multiple-choice training questions from ARC, MC-TEST, OBQA, RACE, etc., for the multidisciplinary domain.

Evaluation Metrics. Following the evaluation protocols of prior works such as QLoRA (Dettmers et al. 2023), LLM-Adapters (Hu et al. 2023), and LoRA-Pro (Wang et al. 2024), we primarily assess the zero-shot performance of fine-tuned LLMs across various benchmarks. for MMLU we report 5-shot accuracy, and for GSM8K we report zero-shot accuracy.

Model	Llama2-7B			Llama3-8B			Mixtral-8x7B		
Dataset	MMLU	GSM8K	Sparsity	MMLU	GSM8K	Sparsity	MMLU	GSM8K	Sparsity
Pretrained	45.6	35.5	-	66.0	72.0	-	70.6	58.4	-
LoRA	56.0	56.8	-	69.2	79.5	-	71.0	79.2	-
LoSA (Huang et al. 2025)	45.0	34.2	50%	64.4	71.4	50%	69.2	57.9	50%
SparseLoRA (Khaki et al. 2025)	**56.0**	37.6	-	**69.0**	72.0	-	70.9	78.0	-
DeepSparse (Agarwalla et al. 2024)	45.1	36.5	50%	60.4	47.9	50%	70.0	59.0	50%
Ours	**56.0**	**56.7**	50%	68.2	**79.5**	50%	**71.4**	**79.1**	50%

Table 2: Performance comparison on the MMLU and GSM8K benchmarks. We utilize a global 50% sparsity for all methods except Pretrained and LoRA. The rank is set to 64. "-" denotes N/A.



(a) LoSA (b) SALR

Figure 3: Normalized cumulative singular-value energy spectra of the residual correction matrices for LoSA and SALR on Llama3-8B after fine-tuning on MetaMath.

Baselines. We compare the performance of SALR against pretrained baseline, the standard LoRA, and several recent LoRA-based pruning methods, including LoSA (Huang et al. 2025), SparseLoRA (Khaki et al. 2025), DeepSparse (Kurtic et al. 2023). Please refer to supplementary for detailed introduction of baselines.

Performance on Domain-Specific Tasks

Across all evaluated models and benchmarks, our method consistently outperforms traditional pruning techniques such as LoSA, SparseLoRA, and DeepSparse. For instance, on Llama2-7B, SALR achieves 56.0 on MMLU and 56.7 on GSM8K, which is notably higher than LoSA (45.0/34.2), SparseLoRA (56.0/37.6), and DeepSparse (45.1/36.5). Similarly, for Llama3-8B, our approach yields 68.2 on MMLU and 79.5 on GSM8K, closely aligning with the performance of LoRA (69.2/79.5), and surpassing other pruning baselines by a significant margin.

Notably, while LoRA remains a strong baseline for parameter-efficient fine-tuning, our method reaches comparable or even superior performance, particularly on Mixtral-8x7B, where SALR scores 71.4 on MMLU and 79.1 on GSM8K, effectively matching or exceeding the LoRA results while providing significant model compression. These results highlight the capability of our approach to maintain high performance while substantially reducing model size. The alignment with LoRA on key benchmarks further demonstrates that our pruning technique does not compromise the model's ability to generalize across diverse tasks.

Figure 3 shows the normalized singular-value energy spectra of the weight updates for LoSA and SALR. Concretely, if $(\sigma_i)_{i=1}^{q}$ are the singular values of the residual correction matrix E, we show

$$\frac{\sum_{j=1}^{i}\sigma_j^2}{\sum_{j=1}^{q}\sigma_j^2} \quad \text{where} \quad i = 1, 2, \ldots, q,$$

and we mark the cube (with yellow) with the smallest index $i_{0.99}$ such that $\sum_{j=1}^{i_{0.99}}\sigma_j^2 / \sum_{j=1}^{q}\sigma_j^2 \geq 0.99$. $i_{0.99}^{\text{LoSA}} \ll i_{0.99}^{\text{SALR}}$ is reported, indicating LoSA requires far fewer singular values to reach 99% energy, whereas SALR retains a much larger tail of the spectrum. In other words, SALR preserves substantially more of the residual's singular-value energy, resulting in a denser effective update. This finding directly corroborates the Theorem 3. By retaining a larger number of singular values (r), SALR drives down the RHS of this bound, preserving more of the original update energy and delivering higher accuracy.

System Efficiency

Model Compression and Throughput in Fine-Tuning. Table 3 reports the fine-tuning memory footprint and sustained throughput for LoSA and SALR. LoSA consumes 27.1 GB of GPU memory and achieves 74.5 TFLOPS, whereas SALR reduces the memory to 19.2 GB and boosts throughput to 89.2 TFLOPS. Remarkably, given sparsity at 50%, our method achieves a 2x reduction in model size for Llama3-8B which matches with LoSA, demonstrating the effectiveness of our bitmap decoding strategy in compressing large language models. The key inefficiency in LoSA arises from its two-stage dense update: $\Delta W = AB$ and then $\Delta Y = X\Delta W$. They are two compute-intensive GEMM operations and contribute to the activation update after applying the sparsity mask. In contrast, since SALR prunes only W, thus it can align the computation with LoRA so that only low-rank products are formed: $U = XA, \Delta Y = UB$, with $U \in \mathbb{R}^{N\times r}$ and $r \ll \min(d_{\text{in}}, d_{\text{out}})$. This strategy replaces the expensive full-rank GEMM by two smaller GEMMs of complexity $O(Nd_{\text{in}}r)$ and $O(Nrd_{\text{out}})$. Moreover, by concatenating all adapters along the rank dimension, SALR fuses multiple updates into a single GEMM, further reducing kernel-launch overhead. Overall, SALR's fused low-rank adapter kernel and sparsity decoding yield a $\sim 30\%$ reduction in fine-tuning memory and a $\sim 20\%$ increase in TFLOPS relative to LoSA.

Inference Speedup. To assess the end-to-end benefits of SALR at inference time, we follow the N:M sparsity protocol of (Huang et al. 2025) and measure the throughput and

Method	Sparsity	FT mem (GB)↓	FT TFLOPS↑	# Comp
LoRA	-	26.7	91.9	-
LoSA	50%	27.1	74.5	2.0 x
SALR	50%	19.2	89.2	2.0 x

Table 3: Fine-tuning GPU memory footprint and sustained throughput (in TFLOPS) on Llama3-8B for LoRA (dense), LoSA (50% sparsity) and SALR (50% sparsity). # Comp denotes model compression rate.

Method (Sparsity)	GSM8K↑	Throughput (tokens/s)↑	Speedup
LoRA (N/A)	79.5	60.1	1.0 x
SparseLoRA (N/A)	72	60.1	1.0 x
LoSA (2:4)	69.4	113.5	1.9 x
SALR (2:4)	78.9	104.9	1.7 x

Table 4: GSM8K accuracy, inference throughput (tokens/s) and inference speedup of Llama3-8B. Dense LoRA and SparseLoRA (no sparsity) serve as baselines, LoSA and SALR are fine-tuned and evaluated under 2:4 semi-structured pattern. Results are collected on 1x RTX4090.

speedup on GSM8K of Llama3-8B. We include dense LoRA and SparseLoRA, which incur no model compression, as well as LoSA and SALR under 2:4 sparsity. Results are shown in Table 4. Despite operating at 50% sparsity, SALR retains nearly the same GSM8K accuracy as dense LoRA while delivering a 1.7× speedup. Compared to LoSA, which suffers a larger accuracy drop under 2:4 sparsity, SALR achieves significantly higher end-task performance (78.9% vs. 69.4%) at only a modest reduction in throughput.

Ablation Study

Train Residual to Boost Performance. To quantify the impact of updating the sparsity-preservation residual during fine-tuning, we compare three variants: standard LoRA, SALR with the residual kept frozen, and SALR with the residual trainable. Results on Llama2-7B and Llama3-8B are summarized in Table 5. When the SVD-derived residual is held fixed, SALR suffers a notable accuracy drop (-1.8 on Llama2-7B, -2.4 on Llama3-8B), indicating that the pruned-weight correction learned from the pretrained model is not fully aligned with the downstream task. By making the residual trainable alongside the LoRA adapters, SALR recovers nearly all of this lost performance, matching LoRA on Llama2-7B and reducing the gap to just 1.0 on Llama3-8B. This demonstrates that fine-tuning the sparsity-preservation residual is crucial for adapting the preserved information to the task.

Coupled with Quantization. To further shrink the memory footprint of SALR for ultra-large models, we combine a 20% static sparsity mask with NF4 quantization, yielding Quantized SALR (QSALR). Table 6 reports end-task accuracy and peak memory for DeepSeek-V2 and Mixtral-8x7B. By applying NF4 quantization on top of 20% sparse SALR, QSALR achieves a ∼5× reduction in model size for both models, dropping from 31.8 GB to 6.5 GB on DeepSeek-V2 and from 93.9 GB to 19.2 GB on Mixtral-8x7B. The performance degradation is minimal (-0.6 on DeepSeek-V2 and none on Mixtral-8x7B), confirming that the structured sparsity and low-precision quantization are highly complementary. For the Huawei NPU deployment of Mixtral-8x7B, QSALR demonstrates similar efficiency and accuracy as on GPU, showing that the quantized sparse representation generalizes well across heterogeneous hardware. This combination enables efficient deployment of very large LLMs in constrained environments without sacrificing task performance.

Method	MMLU↑	
	Llama2-7B	Llama3-8B
LoRA	56	69.2
SALR w/ frozen residual	54.2	66.8
SALR w/ trainable residual	56	68.2

Table 5: Ablation of residual update on MMLU accuracy.

Model	DeepSeek-V2-Lite		Mixtral-8x7B		Mixtral-8x7B(NPU)	
	Acc.↑	Size (GB)↓	Acc.↑	Size (GB)↓	Acc.↑	Size (GB)↓
LoRA	71	31.8	79.2	93.9	79.2	94.0
QSALR	70.4	6.5	79.2	19.2	78.0	19.2

Table 6: GSM8K accuracy and model size of LoRA and SALR with quantization (20% sparsity + NF4).

Method (Sparsity)	Llama3-8B GSM8K↑
LoRA (N/A)	79.5
SALR (10%)	79.5
SALR (30%)	80.1
SALR (50%)	79.5

Table 7: Effect of sparsity level on GSM8K accuracy.

Sparsity–Accuracy Trade-off. We evaluate how varying the global sparsity level affects SALR's performance. Table 7 reports GSM8K accuracy on Llama3-8B for sparsity levels from 10% to 50%. Across all sparsity levels up to 50%, SALR maintains accuracy nearly identical to dense LoRA. Notably, the 30%-sparse model even slightly surpasses the baseline (80.1% vs. 79.5%), suggesting that moderate sparsity can act as a regularizer. Furthermore, SALR can aggressively prune up to half of the weights without compromising the accuracy.

Conclusion

We have presented SALR, a novel fine-tuning paradigm that unifies magnitude-based pruning with low-rank adaptation under a principled MSE framework. Our analysis shows that statically pruning only the frozen base weights minimizes the pruning error bound, and that recovering the discarded residual via a truncated-SVD low-rank adapter provably reduces per-entry MSE. To maximize hardware efficiency, we fuse multiple adapters into a single concatenated GEMM and employ bitmap encoding with a two-stage decoding + GEMM pipeline, yielding true model compression and inference speedup. Empirically, SALR achieves 50% sparsity on various LLMs with no loss in GSM8K and MMLU accuracy, halves model size, and delivers up to 1.7× inference speedup. Our work paves the way for resource-efficient deployment of large language models in constrained environments.

Acknowledgments

This work was partially supported by the Guangzhou Municipal Joint Funding Project with Universities and Enterprises under Grant No. 2024A03J0616, and Hong Kong CRF grants under Grant C6015-23G.

References

Agarwalla, A.; Gupta, A.; Marques, A.; Pandit, S.; Goin, M.; Kurtic, E.; Leong, K.; Nguyen, T.; Salem, M.; Alistarh, D.; Lie, S.; and Kurtz, M. 2024. Enabling High-Sparsity Foundational Llama Models with Efficient Pretraining and Deployment. arXiv:2405.03594.

Büyükakyüz, K. 2024. OLoRA: Orthonormal Low-Rank Adaptation of Large Language Models. arXiv:2406.01775.

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. 2021. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*.

DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948.

Dettmers, T.; Pagnoni, A.; Holtzman, A.; and Zettlemoyer, L. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. *arXiv preprint arXiv:2305.14314*.

Dubey, A.; and et al., A. J. 2024. The Llama 3 Herd of Models. arXiv:2407.21783.

Eckart, C.; and Young, G. 1936. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3): 211–218.

Fan, R.; Yu, X.; Dong, P.; Li, Z.; Gong, G.; Wang, Q.; Wang, W.; and Chu, X. 2025. SpInfer: Leveraging Low-Level Sparsity for Efficient Large Language Model Inference on GPUs. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, 243–260. New York, NY, USA: Association for Computing Machinery. ISBN 9798400711961.

Frantar, E.; and Alistarh, D. 2023. SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot. *arXiv preprint arXiv:2301.00774*.

Guo, S.; Xu, J.; Zhang, L. L.; and Yang, M. 2023. Compresso: Structured Pruning with Collaborative Prompting Learns Compact Large Language Models. arXiv:2310.05015.

Han, S.; Mao, H.; and Dally, W. J. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. arXiv:1510.00149.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*.

Hu, Z.; Lan, Y.; Wang, L.; Xu, W.; Lim, E.-P.; Lee, R. K.-W.; Bing, L.; and Poria, S. 2023. LLM-Adapters: An Adapter Family for Parameter-Efficient Fine-Tuning of Large Language Models. *arXiv preprint arXiv:2304.01933*.

Huang, W.; Zhang, Y.; Zheng, X.; Liu, Y.; Lin, J.; Yao, Y.; and Ji, R. 2025. Dynamic Low-Rank Sparse Adaptation for Large Language Models. arXiv:2502.14816.

Jiang, A. Q.; Sablayrolles, A.; Roux, A.; Mensch, A.; Savary, B.; Bamford, C.; Chaplot, D. S.; de las Casas, D.; Hanna, E. B.; Bressand, F.; Lengyel, G.; Bour, G.; Lample, G.; Lavaud, L. R.; Saulnier, L.; Lachaux, M.-A.; Stock, P.; Subramanian, S.; Yang, S.; Antoniak, S.; Scao, T. L.; Gervet, T.; Lavril, T.; Wang, T.; Lacroix, T.; and Sayed, W. E. 2024. Mixtral of Experts. arXiv:2401.04088.

Khaki, S.; Li, X.; Guo, J.; Zhu, L.; Xu, C.; Plataniotis, K. N.; Yazdanbakhsh, A.; Keutzer, K.; Han, S.; and Liu, Z. 2025. SparseLoRA: Accelerating LLM Fine-Tuning with Contextual Sparsity. arXiv:2506.16500.

Kurtic, E.; Kuznedelev, D.; Frantar, E.; Goin, M.; and Alistarh, D. 2023. Sparse Fine-tuning for Inference Acceleration of Large Language Models. arXiv:2310.06927.

Lester, B.; Al-Rfou, R.; and Constant, N. 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. arXiv:2104.08691.

Li, M.; Si, W. M.; Backes, M.; Zhang, Y.; and Wang, Y. 2025. SaLoRA: Safety-Alignment Preserved Low-Rank Adaptation. arXiv:2501.01765.

Li, X. L.; and Liang, P. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 4582–4597.

Liu, S.-Y.; Wang, C.-Y.; Yin, H.; Molchanov, P.; Wang, Y.-C. F.; Cheng, K.-T.; and Chen, M.-H. 2024. DoRA: Weight-Decomposed Low-Rank Adaptation. *arXiv preprint arXiv:2402.09353*.

Mangrulkar, S.; Gugger, S.; Debut, L.; Belkada, Y.; and Paul, S. 2022. PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods. https://github.com/huggingface/peft.

Meng, F.; Wang, Z.; and Zhang, M. 2024. PiSSA: Principal Singular Values and Singular Vectors Adaptation of Large Language Models. arXiv:2404.02948.

Meta. 2025. The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation. https://ai.meta.com/blog/llama-4-multimodal-intelligence/ [Accessed: 2025-04-05].

OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774.

Sun, M.; Liu, Z.; Bair, A.; and Kolter, J. Z. 2024. A Simple and Effective Pruning Approach for Large Language Models. arXiv:2306.11695.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; and et al., S. B. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288.

Wang, S.; Yu, L.; and Li, J. 2024. LoRA-GA: Low-Rank Adaptation with Gradient Approximation. arXiv:2407.05000.

Wang, Z.; Liang, J.; He, R.; Wang, Z.; and Tan, T. 2024. LoRA-Pro: Are Low-Rank Adapters Properly Optimized? arXiv:2407.18242.

Xia, H.; Zheng, Z.; Li, Y.; Zhuang, D.; Zhou, Z.; Qiu, X.; Li, Y.; Lin, W.; and Song, S. L. 2023. Flash-LLM: Enabling Cost-Effective and Highly-Efficient Large Generative Model Inference with Unstructured Sparsity. arXiv:2309.10285.

Yang, A.; Li, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Gao, C.; Huang, C.; Lv, C.; Zheng, C.; Liu, D.; Zhou, F.; Huang, F.; Hu, F.; Ge, H.; Wei, H.; Lin, H.; Tang, J.; Yang, J.; Tu, J.; Zhang, J.; Yang, J.; Yang, J.; Zhou, J.; Zhou, J.; Lin, J.; Dang, K.; Bao, K.; Yang, K.; Yu, L.; Deng, L.; Li, M.; Xue, M.; Li, M.; Zhang, P.; Wang, P.; Zhu, Q.; Men, R.; Gao, R.; Liu, S.; Luo, S.; Li, T.; Tang, T.; Yin, W.; Ren, X.; Wang, X.; Zhang, X.; Ren, X.; Fan, Y.; Su, Y.; Zhang, Y.; Zhang, Y.; Wan, Y.; Liu, Y.; Wang, Z.; Cui, Z.; Zhang, Z.; Zhou, Z.; and Qiu, Z. 2025. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*.

Yu, L.; Jiang, W.; Shi, H.; Yu, J.; Liu, Z.; Zhang, Y.; Kwok, J. T.; Li, Z.; Weller, A.; and Liu, W. 2023. MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models. *arXiv preprint arXiv:2309.12284*.

Zhang, L.; Zhang, L.; Shi, S.; Chu, X.; and Li, B. 2023a. LoRA-FA: Memory-efficient Low-rank Adaptation for Large Language Models Fine-tuning. arXiv:2308.03303.

Zhang, Q.; Chen, M.; Bukharin, A.; Karampatziakis, N.; He, P.; Cheng, Y.; Chen, W.; and Zhao, T. 2023b. AdaLoRA: Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning. arXiv:2303.10512.