Swift-SVD: Theoretical Optimality Meets Practical Efficiency in Low-Rank LLM Compression

Ruoling Qi*[1 2], Yirui Liu*‡ [1], Xuaner Wu[1], Xiangyu Wang[1], Ming Li[3], Chen Chen[2], Jian Chen† [4 5], Yin Chen[1], Qizhen Weng[1]

[1]Institute of Artificial Intelligence (TeleAI), China Telecom, [2]Shanghai Jiao Tong University, [3]University of Maryland, [4]University at Buffalo, [5]Dolby Laboratories

code available

1. Motivation & Problem Setup

LLM inference is bottlenecked by memory:

Memory *Cost* = *Model Weights* + *KV Cache*.

While model weights are static, KV cache grows with sequence length, making long-context inference memory- and bandwidth-intensive.

Deployment bottleneck

- Static weights dominate model memory.
- KV caches grow with sequence length.

Low-rank compression reduces this cost by approximating

$$W \approx W_k = A_k B_k, \mathrm{rank}(W_k) = k.$$

However, conventional SVD minimizes

$$\| W - W_k \|_F,$$

which ignores input activations X. To better preserve model behavior, we instead minimize the activation-aware output error:

$$W_k^* = \arg \min_{\mathrm{rank}(W_k)=k} \| XW - XW_k \|_F.$$

Core problem:
How to compute the activation-aware optimal low-rank approximation **efficiently, stably, and without retraining**?

2. Preliminary

2.1 Low-Rank Compression Reduces Weights and KV Cache

Given activations $X \in \mathbb{R}^{l \times m}$and weight $W \in \mathbb{R}^{m \times n}$,we approximate

$$W \approx W_k = A_k B_k, A_k \in \mathbb{R}^{m \times k},\ B_k \in \mathbb{R}^{k \times n}.$$

Thus the original linear layer is replaced by two smaller projections:

$$XW \ \Rightarrow\ XA_kB_k.$$

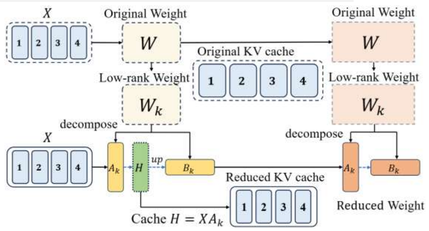
This brings two memory benefits:

Weights: $mn \to k(m+n)$

KV cache: $XW \in \mathbb{R}^{l \times n} \to XA_k \in \mathbb{R}^{l \times k}$

Compression is achieved when

$$k(m+n) < mn,$$
$$k < n.$$



2.2 Activation-Aware Compression Objective

Naive SVD minimizes the weight reconstruction error:

$$\min_{\mathrm{rank}(W_k)=k} \| W - W_k \|_F.$$

However, this ignores the input activations X, and may not preserve the layer output.

We instead minimize the activation-aware reconstruction loss:

$$W_k^* = \arg \min_{\mathrm{rank}(W_k)=k} \| XW - XW_k \|_F,$$
$$\epsilon_k^* = \| XW - XW_k^* \|_F.$$

This objective directly preserves the original output

$$Y = XW.$$

3. Proposed Method

3.1 Optimal Activation-Aware Low-Rank Compression

Theorem 3.1.
Given input activations X and weight matrix W, let $\mathcal{V}$ and Σ denote the right singular vectors and singular values of $Y = XW$, respectively. For any $k < \mathrm{rank}(Y)$,the optimal solution to the activation-aware compression problem is

$$W_k^* = W\mathcal{V}_k\mathcal{V}_k^T, \quad \forall k < \mathrm{rank}(Y)$$

$$\epsilon_k^* = \left(\sum_{j=k+1}^{\mathrm{rank}(Y)} \sigma_j^2 \right)^{\frac{1}{2}}, \quad \forall k < \mathrm{rank}(Y)$$

where $\mathcal{V}_k \in \mathbb{R}^{n \times k}$consists of the top-$k$ right singular vectors corresponding to the k largest singular values.

$$Y^TY \overset{T1}{=} \left(\mathcal{U}\Sigma\mathcal{V}^T\right)^T\mathcal{U}\Sigma\mathcal{V}^T = \mathcal{V}\Sigma^T\mathcal{U}^T\mathcal{U}\Sigma\mathcal{V}^T \overset{T2}{=} \mathcal{V}\Sigma^2\mathcal{V}^T$$

$T1$: substitute $Y = U\Sigma V^{\mathrm{T}}$
$T2$: use $U^{\mathrm{T}}U = I$

Highlights

- Decomposition via incremental aggregation.
- Once eigen-decomposition.

Algorithm 1 Incremental Aggregation Algorithm

Input: Input activations $X = [\mathbf{x}_1; ...; \mathbf{x}_l] \in \mathbb{R}^{l \times m}$; LLM model weights $W \in \mathbb{R}^{m \times n}$
Output: singular values Σ and right singular vectors $\mathcal{V}$ of $Y = XW \in \mathbb{R}^{l \times n}$

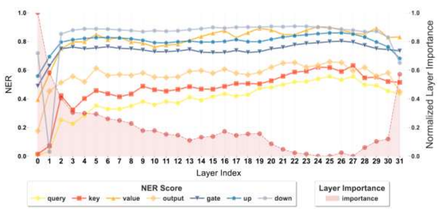
$C \leftarrow$ zero matrix of size $(n \times n)$
for $t = 1, \ldots, l$ **do**
  $\mathbf{y}_t \leftarrow \mathbf{x}_t W$
  $C \leftarrow C + \mathbf{y}_t^T \mathbf{y}_t$
end for
$\mathcal{V}, \Sigma^2, \mathcal{V}^T \leftarrow$ eigen-decomposition(C)
return $\Sigma, \mathcal{V}$

3.2 Dynamic Compression

We employ the normalized effective rank (**NER**) as a quantitative measure of local compressibility.

$$\mathrm{erank}(\Sigma) = \exp\left(-\sum_{i=1}^{r} p_i \ln p_i\right).$$

where $p_i = \sigma_i / \sum_{j=1}^{r} \sigma_j$.
For end-to-end compressibility, we adopt the standard layer importance metric from prior work.



$$s_i = (\beta_i)^\alpha \cdot \left(\log\left(e + \epsilon_{k,i}^*\right)\right)^{1-\alpha}$$

Algorithm 2 Dynamic Rank Allocation Strategy

Input: Frobenius loss ϵ, Layer importance β, Compression ratio ρ, Scaling factor α, Retained ratio δ, Size of original weights m, n
Output: Rank allocation k

$\beta \leftarrow (\beta - \min(\beta))/(\max(\beta) - \min(\beta)) + 1$
$\bar{k} = \frac{m \times n}{m+n} \times \rho$ ▷ *Uniform rank*
for all layer $i \in \{1 \ldots L\}$ **do**
  $k_i \leftarrow \bar{k} \cdot \delta$ ▷ *Guaranteed minimal rank*
  $s_i \leftarrow (\beta_i)^\alpha \cdot (\log(e + \epsilon_{\bar{k},i}))^{1-\alpha}$
end for
$b \leftarrow \bar{k} \times L - \sum_{i=1}^{L} k_i$ ▷ *Flexible rank pool*
for all layer $i \in \{1 \ldots L\}$ **do**
  $k_i \leftarrow k_i + [b \cdot (s_i / \sum_{j=1}^{L} s_j)]$
end for
return k

Highlights

- Compressibility analysis.
- Grid search over candidate rank allocations.

4. Experiments & Efficiency

We evaluate Swift-SVD on 6 LLMs with language modeling and zero-shot QA benchmarks, memory, throughput, and reconstruction error.

4.1 Performance

Ratio(MEM.)	Method	PPL (↓)		Accuracy (↑)						
		WikiText-2	C4	ARC_e	PIQA	Openb.	WinoG.	HellaS.	MathQA	Avg.
1.0(12.6GB)	Baseline	5.68	7.34	0.76	0.79	0.34	0.70	0.57	0.27	0.57
0.8 (10.1 GB)	FWSVD	1727	1511	0.11	0.10	0.09	0.05	0.08	0.05	0.08
	ASVD	11.14	15.93	0.53	0.68	0.29	0.64	0.41	0.17	0.45
	SVD-LLM(W)	7.94	15.84	0.62	0.71	**0.31**	0.61	0.45	0.21	0.49
	Dobi-SVD(w/o)	8.87	10.91	-	-	-	-	-	-	-
	Dobi-SVD(w)	8.54	**10.01**	0.63	0.72	0.30	0.62	0.46	0.20	0.49
	Swift-SVD	7.91	11.42	0.64	0.73	0.26	0.68	0.47	0.23	0.50
	Swift-SVD*	**7.84**	11.15	**0.65**	**0.73**	0.27	**0.68**	**0.48**	**0.23**	**0.51**
0.6 (7.7 GB)	FWSVD	18156	12847	0.05	0.05	0.06	0.02	0.00	0.03	0.04
	ASVD	1407	1109	0.11	0.13	0.08	0.09	0.08	0.08	0.10
	SVD-LLM(W)	13.73	75.42	0.33	0.63	**0.25**	0.55	**0.40**	0.12	0.38
	Dobi-SVD(w/o)	14.96	24.60	-	-	-	-	-	-	-
	Dobi-SVD(w)	13.54	23.54	0.45	0.64	0.22	0.58	0.36	0.18	0.41
	Swift-SVD	13.42	23.32	0.49	0.66	0.21	0.62	0.37	0.22	0.43
	Swift-SVD*	**13.29**	**21.92**	**0.51**	**0.67**	0.23	**0.62**	0.38	**0.22**	**0.44**
0.4 (5.3 GB)	FWSVD	32194	29292	0.02	0.02	0.06	0.01	0.01	0.03	0.03
	ASVD	57057	43036	0.04	0.08	0.05	0.06	0.09	0.05	0.06
	SVD-LLM(W)	66.62	471.83	0.05	0.21	0.10	0.17	0.10	0.04	0.11
	Dobi-SVD(w/o)	58.02	145.41	-	-	-	-	-	-	-
	Dobi-SVD(w)	**46.18**	190.62	0.25	0.51	0.14	0.48	0.24	0.15	0.30
	Swift-SVD	64.16	143.74	0.29	0.56	0.16	0.52	0.27	0.21	0.34
	Swift-SVD*	62.32	**137.77**	**0.30**	**0.57**	**0.16**	**0.53**	**0.28**	**0.21**	**0.34**

Table 1. Performance comparison of LLaMA-7B on language modeling and zero-shot tasks.

Method	OPT-6.7B			LLAMA 2-7B			MISTRAL-7B		
	Perplexity↓		Acc↑	Perplexity↓		Acc↑	Perplexity↓		Acc↑
	Wiki2	C4	Avg.	Wiki2	C4	Avg.	Wiki2	C4	Avg.
Original	10.86	12.52	0.52	5.47	9.30	0.57	5.25	9.28	0.61
ASVD	82.04	102	0.32	10.10	24.02	0.36	13.72	23.34	0.32
SVD-LLM (W)	16.04	21.27	0.41	8.50	12.69	0.53	10.21	13.17	0.42
Swift-SVD	12.12	17.93	0.50	8.41	12.54	0.56	7.40	12.80	0.54
Swift-SVD*	**11.65**	**13.68**	**0.51**	**8.27**	**12.35**	**0.56**	**6.63**	**11.08**	**0.55**

Table 2. Cross-model compression performance under 0.8 compression ratio.

nsamples = 256				nsamples = 2048			
Ratio	Datasets	Original	PPL	Ratio	Datasets	Original	PPL
0.8	C4	11.42	11.42	0.8	C4	11.34	11.34
	WikiText 2	7.86	11.33		WikiText 2	7.81	11.31
	Alpaca	8.49	10.51		Alpaca	7.84	10.38
0.6	C4	23.17	23.17	0.6	C4	22.37	22.37
	WikiText 2	13.42	37.00		WikiText 2	13.37	35.02
	Alpaca	12.31	16.40		Alpaca	9.90	16.82
0.4	C4	137.01	137.01	0.4	C4	136.21	136.21
	WikiText 2	64.16	285.87		WikiText 2	63.01	267.98
	Alpaca	33.97	78.27		Alpaca	24.18	77.48

Table 3. Cross-domain PPL of LLaMA-7B. *Original* uses the evaluation set for calibration; *PPL* uses C4-only.

4.2 Efficiency

Samples	Method	$\rho = 0.8$	$\rho = 0.6$	$\rho = 0.4$	Total (s)	Speedup
16	Dobi-SVD(w/o)	960	650	373	1,983	1.0×
	SVD-LLM (W)	542	489	503	1,534	1.3×
	Swift-SVD	**342**	**146**	**133**	**621**	**3.2×**
64	Dobi-SVD(w/o)	3,823	2,551	1,508	7,882	1.0×
	SVD-LLM (W)	555	530	565	1,650	4.8×
	Swift-SVD	**346**	**158**	**150**	**654**	**12.1×**
256	Dobi-SVD(w/o)	15,269	10,468	5,966	31,703	1.0×
	SVD-LLM (W)	757	734	722	2,213	14.3×
	Swift-SVD	**453**	**152**	**148**	**753**	**42.1×**
512	Dobi-SVD(w/o)	30,862	20,984	11,795	63,641	1.0×
	SVD-LLM (W)	1,080	1,069	1,063	3,212	19.8×
	Swift-SVD	**540**	**157**	**130**	**827**	**76.9×**

Table 4. End-to-end compression latency (in seconds) evaluated on the C4 dataset.

Swift-SVD delivers **3–70$\boldsymbol{X}$** speedups in end-to-end compression time.

4.3 Numerical Stability

Swift-SVD achieves **near-perfect** alignment with the theoretical optimum across all scales.

Input × Weight	$[128 \times 128]^2$	$[1024 \times 1024]^2$	Input × Weight	$[2048 \times 2048]^2$	$[4096 \times 4096]^2$
Minimum	126.2506	841.1812	Minimum	1657.4801	3308.6428
SVD-LLM	126.7102 (+0.4596)	854.4129 (+13.2317)	SVD-LLM	1686.1804 (+28.7003)	3365.6499 (+57.0071)
Dobi-SVD	128.5441 (+2.2935)	874.7711 (+33.5899)	Dobi-SVD	1724.2129 (+66.7328)	3442.5242 (+133.8814)
Swift-SVD	126.2506 (+0.0000)	841.1812 (+0.0000)	Swift-SVD	1657.4801 (+0.0000)	3308.6428 (+0.0000)

Table 5. Reconstruction loss and absolute error for randomly generated matrices of varying shapes under ratio of 0.6 (FP32).