SVD-LLM: Truncation-aware Singular Value Decomposition for Large Language Model Compression

Xin Wang[1] **Yu Zheng**[2] **Zhongwei Wan**[1] **Mi Zhang**[1]*
[1]The Ohio State University [2]Michigan State University
https://github.com/AIoT-MLSys-Lab/SVD-LLM

Abstract

The advancements in Large Language Models (LLMs) have been hindered by their substantial sizes, which necessitate LLM compression methods for practical deployment. Singular Value Decomposition (SVD) offers a promising solution for LLM compression. However, state-of-the-art SVD-based LLM compression methods have two key limitations: truncating smaller singular values may lead to higher compression loss, and the lack of update on the compressed weight after SVD truncation. In this work, we propose SVD-LLM, a new SVD-based LLM compression method that addresses the limitations of existing methods. SVD-LLM incorporates a truncation-aware data whitening strategy to ensure a direct mapping between singular values and compression loss. Moreover, SVD-LLM adopts a layer-wise closed-form model parameter update strategy to compensate for accuracy degradation under high compression ratios. We evaluate SVD-LLM on a total of 10 datasets and eight models from three different LLM families at four different scales. Our results demonstrate the superiority of SVD-LLM over state-of-the-arts, especially at high model compression ratios.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in a wide range of tasks such as natural language understanding and language generation [31, 9]. Despite such capabilities, the democratization of LLMs is primarily restricted by their substantial resource demands [25, 26]. One of the most effective techniques to reduce the resource demands of LLMs is model compression [32]. Compression techniques based on quantization [6, 15, 27], parameter pruning [16, 5], and knowledge distillation [10, 11] specifically designed for LLMs have been intensively studied. Regardless of their success, these techniques have their own constraints, such as hardware dependency and the need for expensive retraining. Compared to those techniques, compression techniques based on low-rank approximation, such as Singular Value Decomposition (SVD) are not limited by those constraints.

Despite these advantages, the potential of SVD for LLM compression has not been thoroughly explored. A few SVD-based LLM compression methods such as ASVD [28] and FWSVD [12] have recently been proposed. However, these methods exhibit severe performance degradation when the model compression ratio[2] is high. Such limitation can be attributed to two fundamental issues involved in their approaches: ❶ **Imprecise Data Preprocessing**: although the data preprocessing strategy proposed by ASVD reduces the negative impact of activation outliers, it does not establish a direct relationship between singular values and the model compression loss. As a consequence, truncating smaller singular values in SVD could lead to significant compression loss. ❷ **Lack of**

*Corresponding author. Email: mizhang.1@osu.edu

[2]The compression ratio refers to the percentage of parameter reduction achieved through compression.

Preprint. Under review.

Model Parameter Update after SVD Truncation: as the model compression ratio increases, the number of singular values that need to be truncated in SVD increases as well. To compensate for the accuracy degradation caused by truncating a large number of singular values, it is required to update the remaining parameters in the compressed model. Unfortunately, existing SVD-based LLM compression methods do not take such update into account, and thus fail to compensate for the accuracy degradation under high model compression ratios.

In this paper, we propose a new SVD-based LLM compression method named `SVD-LLM` that effectively addresses the two fundamental issues of the existing methods. `SVD-LLM` differs from existing SVD-based LLM compression methods in two key aspects: ❶ **Truncation-Aware Data Whitening:** Supported by the theoretical proof, `SVD-LLM` incorporates a truncation-aware data whitening technique that ensures a direct mapping between singular values and model compression loss. In doing so, the proposed truncation-aware data whitening technique is able to identify which singular values should be truncated to incur minimal model compression loss. ❷ **Layer-Wise Closed-Form Model Parameter Update:** to compensate for accuracy degradation under high compression ratios, `SVD-LLM` incorporates a layer-wise closed-form model parameter update strategy to progressively update the compressed weights layer by layer.

We compare `SVD-LLM` with three SVD-based methods for LLM compression, including vanilla SVD as well as state-of-the-art methods FWSVD and ASVD. To demonstrate the generability of `SVD-LLM`, we conduct our evaluation on a total of 10 datasets and eight models from three different LLM families (LLaMA, OPT, and Mistral) at four different scales (7B, 13B, 30B, and 65B). We highlight six of our findings: (1) `SVD-LLM` consistently outperforms vanilla SVD, FWSVD, and ASVD across all 10 datasets, three different LLM families, and four different scales and even exhibits significant advantages under high compression ratios from 30% to 60%. (2) `SVD-LLM` also exhibits superiority over the state-of-the-art in terms of compression speed. Specifically, when compressing LLaMA-7B under 20% compression ratio on an A100 GPU, ASVD takes about 5.5 hours whereas `SVD-LLM` completes the compression process in 15 minutes. (3) The independent performance of either of the two key components of `SVD-LLM` still consistently surpasses the performance of the current state-of-the-art SVD compression method under different compression ratios. (4) `SVD-LLM` can benefit other LLM compression methods. Our evaluation results show that `SVD-LLM` is able to further enhance the compression performance of well-recognized quantization (GPTQ [6]) and parameter pruning-based (LLM-Pruner [16]) LLM compression methods. (5) `SVD-LLM` can ensure inference speedup on both GPU and CPU. It is able to achieve at most 1.7x speedup on GPU and 1.5x speedup on CPU under the 40% compression ratio. (6) Lastly, `SVD-LLM` brings additional benefit beyond compressing the sizes of LLMs, and is also able to reduce the footprint of KV cache during inference at runtime.

2 Related Work

Large Language Model Compression: LLMs in general contain billion-scale parameters. Applying conventional model compression methods for LLMs is not feasible as they necessitate retraining. To avoid retraining, post-training methods that do not involve retraining LLMs in the compression process have been developed. In general, these methods can be grouped into four categories: unstructured pruning, structured pruning, quantization, and low-rank approximation. Specifically, unstructured pruning methods set the individual weights' elements to zero without changing its shape. A notable contribution is SparseGPT [5] which prunes the least important weight elements with the inversion of the Hessian matrix. However, the irregular sparsification of unstructured pruning is difficult to achieve the desired speedup or memory saving and can only demonstrate its best efficiency on certain hardware architecture such as NVIDIA Ampere GPU. Unlike unstructured pruning, structured pruning methods directly remove entire channels or other structured components from LLMs, making them easier to implement on hardware. For example, LLM-Pruner [16] utilizes a small amount of data to obtain the weight, parameter, and group importance of the coupled structure for pruning with LoRA to recover precision. However, due to the great modification of the weight matrix in LLM, it suffers from a great accuracy degradation, especially under high compression ratios. Quantization methods, on the other hand, achieve model compression by reducing the precision of weight matrices of an LLM. For example, GPTQ [6] uses layer-wise quantization and updates the weights with inverse Hessian information. However, quantization has the drawback of only providing a limited range of compression options, typically ranging from 3 to 8 bits. This limited range could prevent full utilization of the available memory budget.

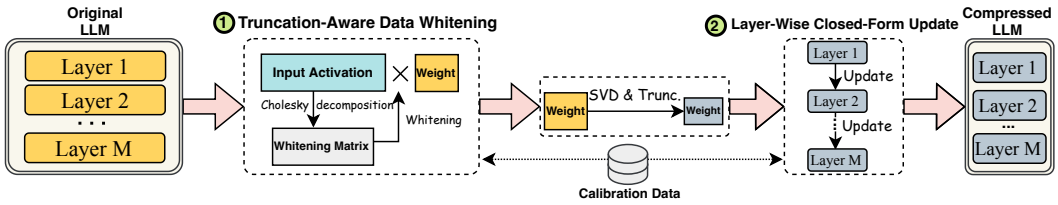


Figure 1: Overview of SVD-LLM.

SVD for LLM Compression: Singular Value Decomposition (SVD) is a widely used technique to reduce matrix size by approximating a matrix with two smaller low-ranking matrices [8]. In the context of LLM compression, only a few SVD-based LLM compression methods have been proposed. Specifically, vanilla SVD only focuses on the compression of the original weight matrix without considering the importance of the parameters, potentially giving a larger compression error. To address this problem, [12] propose FWSVD, which introduces Fisher information to weigh the importance of parameters. However, FWSVD requires a complex gradient calculation that demands substantial resources for LLM compression. Another problem of vanilla SVD is the distribution of activation can affect the compression error. To address this issue, [28] propose ASVD, which scales the weight matrix by a diagonal matrix that represents the impact of input channels on the weights. However, both FWSVD and ASVD do not establish a direct relationship between singular values and compression loss. As a result, truncating the smaller singular values may lead to higher compression loss. Moreover, as the compression ratio increases, it is necessary to update the compressed weight due to truncating a great number of singular values. However, existing methods have no design for this update and thus incur severe accuracy degradation under high compression ratios.

3 SVD-LLM

Figure 1 provides an overview of SVD-LLM. At a high level, SVD-LLM is a SVD-based post-training LLM compression method. Specifically, following the standard procedure of post-training LLM compression methods [5, 28, 27], SVD-LLM uses a random set of sentences as calibration data to generate activation for truncation-aware data whitening and layer-wise closed-form update for model compression. SVD-LLM whitens the activation through Cholesky decomposition, and performs SVD to truncate the weight matrices to compress the LLM. Under high model compression ratios, SVD-LLM performs a layer-wise closed-form update to progressively update the remaining weights layer by layer after compression. In the following, we describe both truncation-aware data whitening and layer-wise closed-form update in detail. The pseudocode is provided in Appendix A.2.

3.1 Truncation-Aware Data Whitening

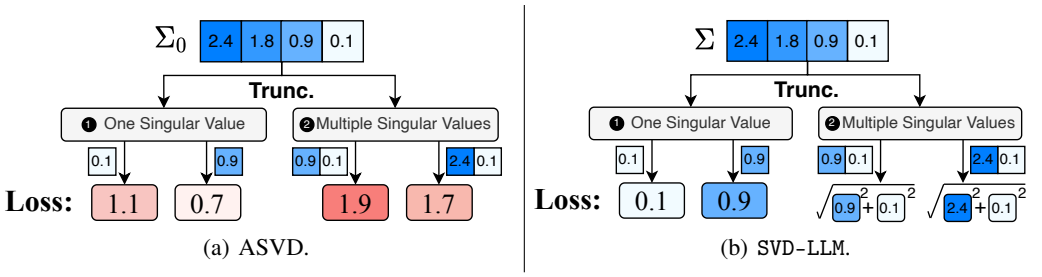
Motivation: Due to high variance of the input activation, simply applying vanilla SVD for LLM compression leads to severe accuracy degradation [28]. To address this issue, ASVD [28] formulates LLM compression as an optimization problem with the following optimization objective:

$$O = \min(||WX - W'X||_F) \tag{1}$$

where W is the weight matrix of the original LLM, X is the activation of W given an input, W' is the compressed weight matrix, and $L = ||WX - W'X||_F$ is the compression loss in the form of Frobenius loss.

Specifically, ASVD extracts a diagonal matrix S_0 from X where each element in the diagonal is the absolute mean value of each channel. It then uses S_0 to normalize X and converts WX into $(WS_0)(S_0^{-1}X)$. Subsequently, SVD is performed on WS_0 to obtain the decomposed matrices U_0, Σ_0, and V_0. Lastly, ASVD truncates the smallest singular values in Σ_0 to obtain the compressed weight matrix $W_0' = U_0 \times \text{Trunc.}(\Sigma_0) \times V_0 \times S_0^{-1}$.

Although normalizing the activation improves the performance, ASVD does not establish a direct relationship between singular values and compression loss (a detailed proof is included in Appendix A.1). To better illustrate this point, we show two concrete examples in Figure 2(a). In the first example ❶ where only one singular value is truncated, truncating the smallest singular value 0.1 results in a higher compression loss (loss $= 1.1$) than truncating the second smallest singular value 0.9 (loss $=$ 0.7). In the second example ❷ where multiple singular values are truncated, truncating the smallest



(a) ASVD. (b) SVD-LLM.

Figure 2: Comparison of data whitening methods between ASVD and SVD-LLM.

two singular values 0.9 and 0.1 also leads to a higher loss (loss = 1.9) than truncating 2.4 and 0.1 (loss = 1.7). Hence, truncating the smallest singular values does not lead to minimal loss.

Key Design: The key idea of SVD-LLM is to incorporate a truncation-aware data whitening technique that ensures a direct mapping between singular values and compression loss. To achieve this, SVD-LLM enforces the whitened activation $S^{-1}X$ to be orthogonal such that each channel is independent of each other, i.e., $(S^{-1}X)(S^{-1}X)^T = S^{-1}XX^T(S^{-1})^T = I$, where S is derived through Cholesky decomposition [19]. Then we perform SVD on WS to obtain the decomposed matrices U, Σ, V, where $U = [u_1, u_2, u_3, ..., u_r]$, $\Sigma = \text{diag}(\sigma_1, \sigma_2, \sigma_3, \cdots, \sigma_r)$, and $V = [v_1, v_2, v_3, ..., v_r]$. Lastly, the smallest singular values in Σ are truncated to obtain the compressed weight matrix $W' = U \times \text{Trunc.}(\Sigma) \times V^T \times S^{-1}$.

Figure 2(b) illustrates the effect of the proposed truncation-aware data whitening technique. In the first example ❶ where only one singular value is truncated, the compression loss is equal to the truncated singular value. In the second example ❷, the compression loss of truncating multiple singular values is equal to the square root of the sum of their squares. As such, under the proposed truncation-aware data whitening technique, truncating the smallest singular values leads to minimal compression loss.

Below, we provide a theoretical proof on why the proposed truncation-aware data whitening technique ensures a direct mapping between singular values and compression loss in the case of one singular value (Theorem 3.2) and multiple singular values (Corollary 3.3).

Lemma 3.1. *The Frobenius norm of matrix A with dimension $m \times n$ can be deduced into the square root of the trace of its gram matrix, which is:*

$$\|A\|_F \triangleq \left(\sum_{j=1}^{n} \sum_{i=1}^{m} |a_{ij}|^2 \right)^{\frac{1}{2}} = \left[\text{trace}\left(A^T A\right) \right]^{\frac{1}{2}} \tag{2}$$

Using Lemma 3.1, we obtain the compression loss L_i when truncating the i^{th} singular value of $S^{-1}X$ to reduce its rank for compression:

$$L_i = ||(W - W')X||_F = ||\sigma_i u_i v_i^T S^{-1} X||_F = \sigma_i \, trace(u_i v_i^T S^{-1} X X^T (S^{-1})^T v_i u_i^T)^{\frac{1}{2}} \tag{3}$$

Since both $U = [u_1, u_2, u_3, ..., u_r]$ and $V = [v_1, v_2, v_3, ..., v_r]$ are orthogonal matrices, we have:

$$v_i^T v_i = u_i^T u_i = I; v_i^T v_j = u_i^T u_j = 0, \forall i \neq j; trace(v_i v_i^T) = trace(u_i u_i^T) = 1 \tag{4}$$

Theorem 3.2. *If S is the Cholesky decomposition of XX^T, the compression loss L_i equals to σ_i.*

Proof. Since the whitening matrix S is the Cholesky decomposition of XX^T, we have $SS^T = XX^T$. We can further infer Equation (3) to obtain:

$$L_i = ||\sigma_i u_i v_i^T S^{-1} X||_F = \sigma_i \, trace(u_i v_i^T S^{-1} X X^T (S^{-1})^T v_i u_i^T)^{\frac{1}{2}} = \sigma_i \, trace(u_i v_i^T v_i u_i^T)^{\frac{1}{2}} = \sigma_i \tag{5}$$

Therefore, L_i of truncating σ_i equals to the singular value σ_i itself. □

Corollary 3.3. *If S is the Cholesky decomposition of XX^T, truncating the smallest singular values leads to the lowest loss L compared to truncating others.*

Proof. If we truncate $\sigma_{m+1}, \sigma_{m+2}, \sigma_{m+3}, ..., \sigma_r$ in Σ for compression, the square of the loss L is:

$$
\begin{aligned}
L^2 &= \left|\left| \sum_{i=m+1}^{r} \sigma_i u_i v_i^T S^{-1} X \right|\right|_F^2 = \sum_{j=m+1}^{r} \sum_{i=m+1}^{r} \sigma_i \sigma_j \, trace(u_i v_i^T S^{-1} X X^T (S^{-1})^T v_j u_j^T) \\
&= \sum_{i=m+1}^{r} \sigma_i^2 \, trace(u_i v_i^T S^{-1} X X^T (S^{-1})^T v_i u_i^T) = \sum_{i=m+1}^{r} (L_i)^2 = \sum_{i=1}^{k} (\sigma_i)^2
\end{aligned}
\tag{6}
$$

The squared loss L^2 is equal to the sum of the squared singular values. Therefore, truncating the smallest singular values σ_i achieves the lowest compression loss. □

3.2 Layer-Wise Closed-Form Update

Motivation: Given the same calibration data as input, the compressed weight matrix W' generates a new activation X' that is different from X generated by the original weight matrix W. As the compression ratio increases, W needs to truncate a larger number of singular values to obtain W', thus X' deviates further from X. Therefore, it becomes necessary to design a strategy for updating W' to minimize $||WX - W'X'||_F$. However, existing SVD-based LLM compression methods have no design of parameter update after compression, leading to less competitive performance at high compression ratios.

Key Design: The key idea of `SVD-LLM` is to incorporate a layer-wise closed-form strategy to update W' to minimize $||WX' - W'X'||_F$. Figure 3 illustrates the overall process. To perform the update in layer i, `SVD-LLM` uses the new activation X'_{i-1} from the previous layer $i-1$ that has been updated. To perform the update without destroying the low-rank structure of W'_i, `SVD-LLM` only updates the matrix U_i to its closed-form solution U'_i while keeping Trunc.$(\Sigma)_i$ and V_i fixed as:

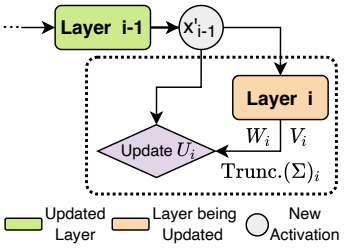


Figure 3: Layer-Wise Closed-Form Update.

$$
\begin{aligned}
U'_i &= \arg\min_{U'_i} ||W_i X'_{i-1} - W'_i X'_{i-1}||_F \\
&= W_i X'_{i-1} D^T (DD^T)^{-1}, D = \text{Trunc.}(\Sigma)_i V_i^T S_i^{-1} X'_{i-1}
\end{aligned}
\tag{7}
$$

4 Experiments

Baselines. We compare `SVD-LLM` against three baselines including vanilla SVD as well as state-of-the-art SVD-based LLM compression methods FWSVD [12] and ASVD [28].

Models and Datasets. To demonstrate the generability of our method, we evaluate the performance of `SVD-LLM` and the baselines on eight models from three different LLM families (LLaMA-7B, 13B, 30B, 65B [23], LLaMA2-7B [24], OPT-6.7B [30], Vicuna-7B [3] and Mistral-7B [14]) and 10 datasets including three language modeling datasets (WikiText-2 [18], PTB [17] and C4 [21]) and seven classification datasets (OpenbookQA [20], WinoGrande [22], HellaSwag [29], PIQA [2], MathQA [1], ARC-e, and ARC-c [4]) in zero-shot setting with LM-Evaluation-Harness framework [7].

Implementation Details. To ensure a fair comparison, we followed ASVD [28] to randomly select 256 samples from WikiText-2 as the calibration data. Since layer-wise closed-form update is intended to mitigate the accuracy drop under higher compression ratios, we only apply it when the compression ratios are at 40% and above. All of our experiments are conducted on Nvidia A100 GPUs.

4.1 Overall Performance

We evaluate the overall performance of `SVD-LLM` from four aspects: (1) performance under different compression ratios, (2) performance on different LLMs, (3) performance on LLMs with larger scales, and (4) performance with LoRA fine-tuning (See Appendix A.3). Some generated contents by the compressed LLM are listed in Appendix A.4 to provide a more straightforward comparison.

Performance under Different Compression Ratios. First, we evaluate the performance of LLaMA-7B compressed by `SVD-LLM` and the baselines under 20% to 60% compression ratios. Table 1

Table 1: Zero-shot performance of LLaMA-7B compressed by SVD-LLM and baselines under 20% to 60% compression ratio on three language modeling datasets (measured by perplexity (↓)) and seven common sense reasoning datasets (measured by both individual and average accuracy (↑)). The best performance is marked in bold. The relative performance gain compared to the best-performing baseline is marked in green color inside bracket.

RATIO	METHOD	WikiText-2↓	PTB↓	C4↓	Openb.	ARC_e	WinoG.	HellaS.	ARC_c	PIQA	MathQA	**Average↑**
0%	Original	5.68	8.35	7.34	0.28	0.67	0.67	0.56	0.38	0.78	0.27	0.52
20%	SVD	20061	20306	18800	0.14	0.27	0.51	0.26	0.21	0.53	0.21	0.31
	FWSVD	1727	2152	1511	0.15	0.31	0.50	0.26	0.23	0.56	0.21	0.32
	ASVD	11.14	16.55	15.93	**0.25**	0.53	**0.64**	0.41	0.27	0.68	0.24	0.43
	SVD-LLM	**7.94** (↓29%)	**16.22** (↓2%)	**15.84** (↓1%)	0.22	**0.58**	0.63	**0.43**	**0.29**	**0.69**	**0.24**	**0.44** (↑2%)
30%	SVD	13103	17210	20871	0.13	0.26	0.51	0.26	0.21	0.54	0.22	0.30
	FWSVD	20127	11058	7240	0.17	0.26	0.49	0.26	0.22	0.51	0.19	0.30
	ASVD	51	70	41	0.18	0.43	0.53	0.37	0.25	0.65	0.21	0.38
	SVD-LLM	**9.56** (↓81%)	**26.39** (↓62%)	**25.11** (↓39%)	**0.20**	**0.48**	**0.59**	**0.37**	**0.26**	**0.65**	**0.22**	**0.40** (↑5%)
40%	SVD	52489	59977	47774	0.15	0.26	0.52	0.26	0.22	0.53	0.20	0.30
	FWSVD	18156	20990	12847	0.16	0.26	0.51	0.26	0.22	0.53	0.21	0.30
	ASVD	1407	3292	1109	0.13	0.28	0.48	0.26	0.22	0.55	0.19	0.30
	SVD-LLM	**13.11** (↓99%)	**63.75** (↓98%)	**49.83** (↓96%)	**0.19**	**0.42**	**0.58**	**0.33**	**0.25**	**0.60**	**0.21**	**0.37** (↑23%)
50%	SVD	131715	87227	79815	0.16	0.26	0.50	0.26	0.23	0.52	0.19	0.30
	FWSVD	24391	28321	23104	0.12	0.26	0.50	0.26	0.23	0.53	0.20	0.30
	ASVD	15358	47690	27925	0.12	0.26	0.51	0.26	0.22	0.52	0.19	0.30
	SVD-LLM	**23.97** (↓99%)	**150.58** (↓99%)	**118.57** (↓99%)	**0.16**	**0.33**	**0.54**	**0.29**	**0.23**	**0.56**	**0.21**	**0.33** (↑10%)
60%	SVD	105474	79905	106976	**0.16**	0.26	0.50	0.26	0.22	0.52	0.21	0.30
	FWSVD	32194	43931	29292	0.15	0.26	0.49	0.26	0.22	0.53	0.18	0.30
	ASVD	57057	45218	43036	0.12	0.26	0.49	0.26	0.21	0.51	0.18	0.29
	SVD-LLM	**53.74** (↓99%)	**438.58** (↓99%)	**345.49** (↓99%)	0.14	**0.28**	**0.50**	**0.27**	**0.22**	**0.55**	**0.21**	**0.31** (↑7%)

Table 2: Perplexity (↓) of four different LLMs including OPT-6.7B, LLaMA 2-7B, Mistral-7B, and Vicuna-7B under 20% compression ratio on WikiText-2. The relative performance gain compared to the best-performing baseline is marked in green color inside bracket.

METHOD	OPT-6.7B	LLaMA 2-7B	Mistral-7B	Vicuna-7B
SVD	66275	18192	159627	18644
FWSVD	14559	2360	6357	2758
ASVD	82	10.10	13.72	16.23
SVD-LLM	**16.04** (↓80%)	**8.50** (↓16%)	**10.21** (↓26%)	**6.78** (↓58%)

Table 3: Perplexity (↓) of LLaMA-7B, 13B, 30B, 65B under 20% compression ratio on WikiText-2. Some baselines' results are not available due to running out of memory (OOM) during model compression. The relative performance gain compared to the best-performing baseline is marked in green color inside bracket.

METHOD	LLaMA-7B	LLaMA-13B	LLaMA-30B	LLaMA-65B
SVD	20061	946.31	54.11	11.27
FWSVD	1630	OOM	OOM	OOM
ASVD	11.14	6.74	22.71	OOM
SVD-LLM	**7.94** (↓29%)	**6.61** (↓2%)	**5.63** (↓75%)	**6.58** (↓42%)

summarizes the results on all 10 datasets. As shown, SVD-LLM consistently outperforms vanilla SVD, FWSVD and ASVD across all of the compression ratios. More importantly, compared to the low compression ratio scenario in Table 1, SVD-LLM exhibits significant advantages over vanilla SVD, FWSVD, and ASVD under high compression ratios. Specifically, under 30% compression ratio, compared to the best-performing baseline (ASVD), SVD-LLM reduces the perplexity on WikiText-2, PTB, and C4 by 81%, 62%, and 39%, respectively; When the compression ratio reaches 40% and above, SVD-LLM reduces the perplexity by more than 96%. These results indicate that SVD-LLM is more effective in compressing LLMs for more resource-constrained devices such as smartphones and IoT devices. On the seven classification datasets, SVD-LLM performs better than the best-performing baseline on most of the datasets and consistently achieves at least 2% higher average accuracy across all the compression ratios.

Performance on Different LLMs. To examine the generability of SVD-LLM across different LLMs, we compare the performance between SVD-LLM and the baselines on four different models, including OPT-6.7B, LLaMA 2-7B, Mistral-7B, and Vicuna-7B under 20% compression ratio on WikiText-2 and the common sense reasoning datasets (See Appendix A.5). The result on WikiText-2 is shown in Table 2, SVD-LLM consistently outperforms vanilla SVD, FWSVD, and ASVD across all four LLMs. In addition, SVD-LLM exhibits more stable performance on different LLM families, especially compared to vanilla SVD and FWSVD.

Performance on LLMs with Larger Scales. To examine the generability of SVD-LLM on LLMs across different scales, we compare the performance between SVD-LLM and the baselines on LLaMA series at four different scales – 7B, 13B, 30B, and 65B – under 20% compression ratio on WikiText-2. As shown in Table 3, SVD-LLM consistently outperforms vanilla SVD, FWSVD, and ASVD across all four model sizes. Moreover, both FWSVD and ASVD demand excessive memory resources,

Table 6: Perplexity (↓) of compressed LLaMA-7B on WikiText-2. `SVD-LLM (W)` denotes the version of `SVD-LLM` with truncation-aware data whitening only; `SVD-LLM (U)` denotes the version of `SVD-LLM` with layer-wise closed-form update only; `SVD-LLM (W+U)` denotes the version of `SVD-LLM` with both truncation-aware data whitening and layer-wise closed-form update. The relative performance gain compared to ASVD is marked in green color.

METHOD	20%	30%	40%	50%	60%
ASVD	11.14	51	1407	15358	57057
`SVD-LLM (W)`	**7.94** (↓29%)	**9.56** (↓81%)	13.73 (↓99%)	26.11 (↓99%)	66.62 (↓99%)
`SVD-LLM (U)`	9.54 (↓14%)	12.98 (↓75%)	24.16 (↓99%)	72.13 (↓99%)	204 (↓99%)
`SVD-LLM (W+U)`	8.25 (↓26%)	9.95 (↓80 %)	**13.11** (↓99%)	**23.97** (↓99%)	**53.74** (↓99%)
`SVD-LLM`	**7.94** (↓29%)	**9.56** (↓81%)	**13.11** (↓99%)	**23.97** (↓99%)	**53.74** (↓99%)

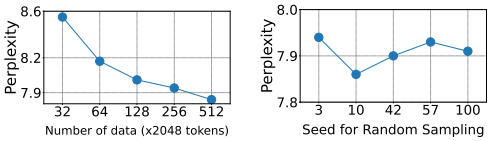
causing out of memory (OOM) when compressing LLMs at larger scales even on an A100 GPU due to memory-intensive operations for estimating the importance of weight matrices. In contrast, `SVD-LLM` does not involve such estimation operations and thus avoids OOM.

4.2 Compression Speed Evaluation

Besides compression performance, we also evaluate the compression speed of `SVD-LLM` and the baselines. Specifically, we measured the GPU hours used for `SVD-LLM` and ASVD when compressing LLaMA-7B under the 20% compression ratio on an A100 GPU. The results are shown in Table 4. As shown, ASVD takes about 5.5 hours whereas `SVD-LLM` completes the compression process in 15 minutes, which is 32 times faster. When breaking down the time, most of the time consumed by ASVD is dedicated to calculating the compression ratio of each weight matrix based on its estimated importance through a search process. In contrast, `SVD-LLM` maintains a consistent compression ratio among all weight matrices and thus gets rid of the time-consuming search process.

Table 4: Compression time of `SVD-LLM` and ASVD on LLaMA-7B under 20% compression ratio. The relative speedup is marked in green color inside bracket.

METRIC	SVD-LLM			ASVD		
	White	Update	**Total**	Normalize	Search	**Total**
TIME	10min	5min	**15min** (↓95%)	5min	5.5h	**5.5h**



(a) Change of Number (b) Change of Seed

Figure 4: Perplexity of LLaMA-7B under 20% compression ratio using calibration data with different number or sampled with different seeds from WikiText-2.

Table 5: Performance of LLaMA-7B compressed by `SVD-LLM` under 20% and 30% compression ratios using calibration data randomly sampled from WikiText-2 (by default in our paper) and C4. The performance on WikiText-2, PTB, and C4 is reported by perplexity (↓), while the performance on OpenbookQA and HellaSwag are reported by accuracy (↑). The relative performance drop (gain) for data sampled from C4 compared to that sampled from WikiText-2 is marked in red (green) color inside bracket.

RATIO	WikiText-2	PTB	C4	Openb.	HellaS.
Calibration data sampled from WikiText-2 (seed=3)					
20%	7.94	16.22	15.84	0.22	0.43
30%	9.56	26.39	25.11	0.20	0.37
Calibration data sampled from C4 (seed=3)					
20%	8.62(↑9%)	14.63(↓9%)	13.77(↓13%)	0.23(↑5%)	0.43
30%	10.67(↑12%)	25.07(↓5%)	22.37(↓10%)	0.22(↑9%)	0.42(↑14%)

4.3 Ablation Study

Modular Sensitivity Study: We conduct ablation studies to evaluate the separate contributions of the two key components (truncation-aware data whitening and layer-wise closed-form update) of `SVD-LLM`. Let `SVD-LLM (W)` denote the version of `SVD-LLM` with truncation-aware data whitening only; `SVD-LLM (U)` denote the version of `SVD-LLM` with layer-wise closed-form update only; and `SVD-LLM (W+U)` denote the version of SVD-LLM with both truncation-aware data whitening and layer-wise closed-form update. The results are shown in Table 6. We have three observations. (1) Both `SVD-LLM (W)` and `SVD-LLM (U)` consistently outperform ASVD across all the compression ratios. Notably, when the compression ratio is at and above 40%, both variants reduce the perplexity by more than 99% compared to ASVD. (2) Under 20% and 30% compression ratios, `SVD-LLM (W)` achieves the lowest perplexity compared to `SVD-LLM (U)` and `SVD-LLM (W+U)`. (3) Under 40%, 50% and 60% compression ratios, `SVD-LLM (W+U)` achieves the lowest perplexity compared to `SVD-LLM (W)` and `SVD-LLM (U)`, highlighting the importance of combining both truncation-aware data whitening and layer-wise closed-form update when compression ratio goes high.

Table 7: Perplexity (↓) of LLaMA-7B compressed by GPTQ w/ and w/o SVD-LLM on WikiText-2. The relative performance gain of combined compression compared to GPTQ-3bit is marked in green color inside bracket.

METRIC	GPTQ-4bit	GPTQ-3bit	SVD-LLM + GPTQ-4bit
Memory	3.9 GB	2.8 GB	2.1 GB
Perplexity	6.21	16.28	13.29 (↓18%)

Table 8: Perplexity (↓) of LLaMA-7B compressed by LLM-Pruner w/ and w/o SVD-LLM on WikiText-2. The relative performance gain of combined compression compared to LLM-Pruner under 40% compression ratio is marked in green color.

METRIC	LLM-Pruner-30%	LLM-Pruner-40%	LLM-Pruner-30% + SVD-LLM
Memory	9.8 GB	8.8 GB	8.8 GB
Perplexity	9.88	12.21	10.58 (↓13%)

Calibration Data Analysis: We next analyze the impact of calibration data used for both truncation-aware data whitening and layer-wise closed-form update on the compression performance. Figure 4 and Table 5 summarize the performance of compressed LLaMA-7B when changing three key characteristics of the calibration data, the number of the calibration data, the seed used to randomly sample the calibration data, and the data set from which the calibration data is sampled. As shown, changing any of the three characteristics only causes a tiny disturbance of less than 15% to the final performance, demonstrating that SVD-LLM is less sensitive to the design of the calibration data to compress LLM.

4.4 Benefits to other LLM Compression Methods

SVD-LLM is orthogonal to other LLM compression methods including quantization and parameter pruning. In this experiment, we combine SVD-LLM with quantization and parameter pruning-based LLM compression methods that are widely recognized by the community to examine how SVD-LLM could further enhance their performance.

Integrate SVD-LLM with Quantization. We select GPTQ [6] as the quantization method. Specifically, we compress LLaMA-7B by GPTQ-4bit combined with SVD-LLM, and compare the compressed model against LLaMA-7B compressed by GPTQ-3bit. As shown in Table 7, combining GPTQ-4bit with SVD-LLM achieves a perplexity that is 18% lower than GPTQ-3bit even with a smaller memory footprint (2.1 GB vs. 2.8 GB). This result demonstrates that compared to directly quantizing using smaller number of bits, GPTQ achieves better compression performance with the help of SVD-LLM.

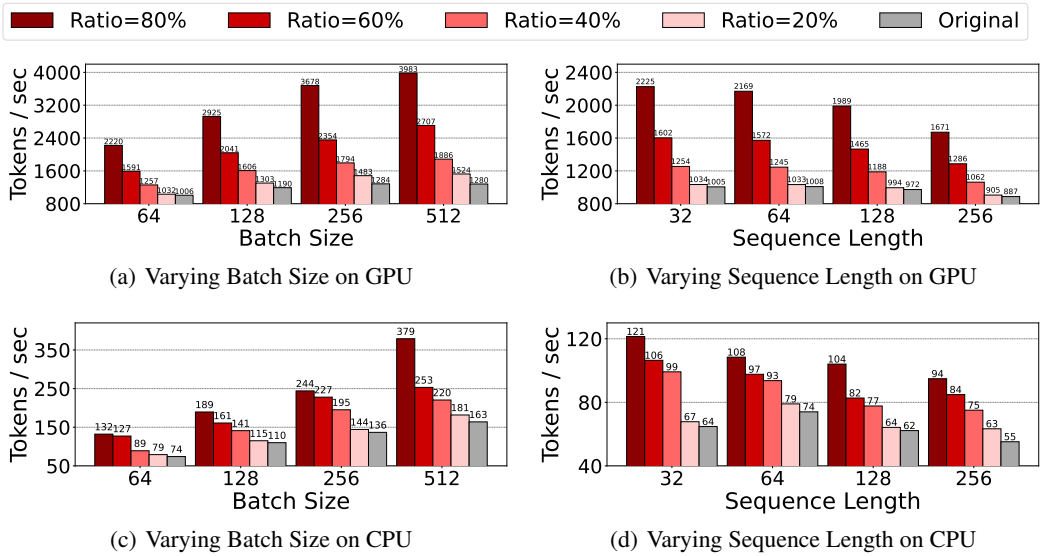
Integrate SVD-LLM with Parameter Pruning. We select LLM-Pruner [16] as the parameter pruning method. Specifically, we compress LLaMA-7B by LLM-Pruner under 30% compression ratio combined with SVD-LLM, and compare the compressed model against LLaMA-7B compressed by LLM-Pruner under 40% compression ratio. As shown in Table 8, LLM-Pruner achieves better compression performance when used in conjunction with SVD-LLM. In particular, with the same memory footprint of 8.8 GB, combining LLM-Pruner under 30% compression ratio with SVD-LLM achieves a perplexity that is 13% lower than LLM-Pruner under 40% compression ratio.

4.5 Benefits of Inference Speedup

SVD-LLM is capable to achieve inference speedup. To demonstrate this advantage, we measure the number of tokens that the original LLaMA-7B and its compressed version by SVD-LLM can generate on average per second with different batch size and sequence length. Figure 5 show the results on the GPU and CPU. As shown, SVD-LLM consistently ensures an acceleration in the generation speed across all the compression ratios illustrated in the figure. More importantly, this enhancement becomes more significant as the batch size increases and the sequence length decreases, resulting in a maximum speedup of 1.7x on GPU and 1.5x on CPU under the 40% compression ratio, where the model performance remains acceptable according to Table 1. These results highlight the effectiveness of SVD-LLM in improving the efficiency of LLM for real-world usage.

4.6 Benefits of KV Cache Compression

SVD-LLM is able to not only compress LLMs but also compress the runtime KV cache *at the same time*. Specifically, instead of keeping the original intermediate state matrix $m = WX$ with shape $M \times L$ inside the KV cache, after decomposing and compressing W into $W_u W_v^T$, SVD-LLM only needs to store $m' = W_v^T X$ with shape $r \times L$. Therefore, the size of the KV cache can be compressed to $\frac{r}{M}$ of the original. Moreover, since W_u is already stored as the weight matrix in the decomposed LLM, the original intermediate state matrix can still be recovered by $m = W_u m'$ without accuracy drop. Therefore, SVD-LLM provides a unified solution that combines model compression and KV cache compression into a single process. This is different from existing quantization or parameter



(a) Varying Batch Size on GPU

(b) Varying Sequence Length on GPU

(c) Varying Batch Size on CPU

(d) Varying Sequence Length on CPU

Figure 5: Throughput (Tokens/sec) of original LLaMA-7B and its compressed version by SVD-LLM under 20%, 40%, 60% and 80% compression ratio on single A100 GPU (Figure (a),(b)) and single AMD EPYC 7643 CPU (Figure (c),(d)). Figure (a),(c) is the comparison with different batch size while sequence length = 32, Figure (b), (d) is the comparison with different sequence length while batch size = 64.

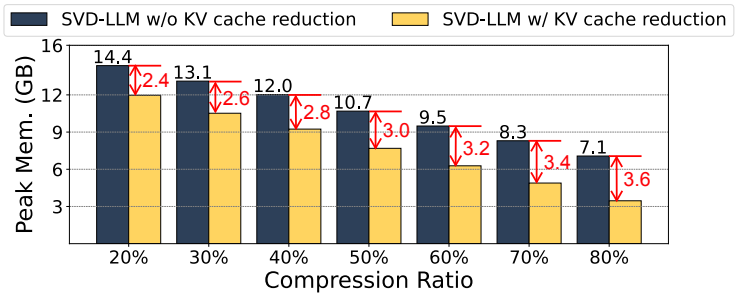


Figure 6: Peak memory to generate 128 tokens with batch size of 32 using LLaMA-7B compressed by SVD-LLM under different compression ratios w/ and w/o KV-cache compression. The difference between the blue and yellow bars marked in red indicates the reduced footprint of the KV cache.

pruning-based LLM compression methods that need to be combined with other techniques for compressing both weights and KV cache..

In our last experiment, we evaluate this benefit on KV cache compression brought by SVD-LLM. This is a new avenue since KV cache compression has not been evaluated in previous LLM compression studies. Specifically, we measure the peak memory footprint during inference when generating 128 tokens with batch size of 32 using LLaMA-7B compressed by SVD-LLM under different compression ratios w/ and w/o considering KV cache compression. The results are illustrated in Figure 6 where the difference between the blue and yellow bars marked in red represents the reduced footprint of the KV cache. As shown, SVD-LLM is able to effectively reduce the footprint of KV cache. Therefore, the peak memory during inference at runtime across all the compression ratios.

5 Conclusion

In this paper, we presented SVD-LLM, a SVD-based LLM compression method. SVD-LLM proposes a novel truncation-aware data whitening strategy to guide which singular values to be truncated with minimal compression loss. It also introduces a layer-wise closed-form model parameter update scheme to compensate for accuracy degradation under high compression ratios. We have demonstrated the effectiveness of SVD-LLM on 10 datasets and seven models from three LLM families at four scales and have shown its superiority over state-of-the-arts. We also show its effectiveness in further enhancing the performance of other LLM compression methods.

References

[1] Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *NAACL-HLT (1)*, pages 2357–2367. Association for Computational Linguistics, 2019.

[2] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: reasoning about physical commonsense in natural language. In *AAAI*, pages 7432–7439. AAAI Press, 2020.

[3] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL `https://lmsys.org/blog/2023-03-30-vicuna/`.

[4] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457, 2018.

[5] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 10323–10337. PMLR, 2023.

[6] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: accurate post-training quantization for generative pre-trained transformers. *CoRR*, abs/2210.17323, 2022.

[7] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL `https://zenodo.org/records/10256836`.

[8] G.H. Golub, Alan Hoffman, and G.W. Stewart. A generalization of the eckart-young-mirsky matrix approximation theorem. *Linear Algebra and its Applications*, 88-89:317–327, 1987. ISSN 0024-3795. doi: https://doi.org/10.1016/0024-3795(87)90114-5. URL `https://www.sciencedirect.com/science/article/pii/0024379587901145`.

[9] Roberto Gozalo-Brizuela and Eduardo C. Garrido-Merchán. A survey of generative AI applications. *CoRR*, abs/2306.02781, 2023.

[10] Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Knowledge distillation of large language models. *CoRR*, abs/2306.08543, 2023.

[11] Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *ACL (Findings)*, pages 8003–8017. Association for Computational Linguistics, 2023.

[12] Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. Language model compression with weighted low-rank factorization. In *ICLR*. OpenReview.net, 2022.

[13] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *ICLR*. OpenReview.net, 2022.

[14] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *CoRR*, abs/2310.06825, 2023.

[15] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. AWQ: activation-aware weight quantization for LLM compression and acceleration. *CoRR*, abs/2306.00978, 2023.

[16] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In *NeurIPS*, 2023.

[17] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Comput. Linguistics*, 19(2):313–330, 1993.

[18] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *ICLR (Poster)*. OpenReview.net, 2017.

[19] Carl Dean Meyer. *Matrix Analysis and Applied Linear Algebra*. SIAM, 2000.

[20] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? A new dataset for open book question answering. In *EMNLP*, pages 2381–2391. Association for Computational Linguistics, 2018.

[21] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020.

[22] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *AAAI*, pages 8732–8740. AAAI Press, 2020.

[23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023.

[24] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288, 2023.

[25] Zhongwei Wan, Xin Wang, et al. Efficient large language models: A survey. *arXiv preprint arXiv:2312.03863*, 2023.

[26] Xin Wang, Zhongwei Wan, Arvin Hekmati, Mingyu Zong, Samiul Alam, Mi Zhang, and Bhaskar Krishnamachari. Iot in the era of generative ai: Vision and challenges. *arXiv preprint arXiv:2401.01923*, 2024.

[27] Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 38087–38099. PMLR, 2023.

[28] Zhihang Yuan, Yuzhang Shang, Yue Song, Qiang Wu, Yan Yan, and Guangyu Sun. ASVD: activation-aware singular value decomposition for compressing large language models. *CoRR*, abs/2312.05821, 2023.

[29] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *ACL (1)*, pages 4791–4800. Association for Computational Linguistics, 2019.

[30] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. OPT: open pre-trained transformer language models. *CoRR*, abs/2205.01068, 2022.

[31] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *CoRR*, abs/2303.18223, 2023.

[32] Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A survey on model compression for large language models. *CoRR*, abs/2308.07633, 2023.

A Appendix.

A.1 The compression error of ASVD

The previous state-of-the-art method ASVD introduced a diagonal scaling matrix S_0 that modifies the weight matrix to reflect the varying significance of different input channels. The linear layer is formulated as $Y = (WS_0)S_0^{-1}X$. The compression is made by keeping the largest m singular value of WS_0:

$$WS_0 \approx \sum_{i=1}^{m} \sigma'_i u'_i {v'}_i^T$$

The resulting activation is expressed as:

$$Y \approx \sum_{i=1}^{m} \sigma'_i u'_i {v'}_i^T S_0^{-1} X \ .$$

The compression error $L = ||(WS_0 - \sum_{i=1}^{m} \sigma'_i u'_i {v'}_i^T)S_0^{-1}X||_F$ is demonstrated below:

$$\begin{aligned}
L^2 =& ||(WS_0 - \sum_{i=1}^{m} \sigma'_i u'_i {v'}_i^T)S_0^{-1}X||_F^2 \\
=& \left|\left| \sum_{i=m+1}^{r} \sigma'_i u'_i {v'}_i^T S_0^{-1} X \right|\right|_F^2 \\
=& \sum_{j=m+1}^{r} \sum_{i=m+1}^{r} \sigma'_i \sigma'_j \ trace(u'_i {v'}_i^T X X^T v'_j {u'}_j^T) \\
=& \sum_{j=m+1}^{r} \sum_{i=m+1}^{r} \sigma'_i \sigma'_j \ trace({u'}_j^T u'_i {v'}_i^T S_0^{-1} X X^T (S_0^{-1})^T v'_j) \\
=& \sum_{i=m+1}^{r} {\sigma'}_i^2 \ trace({v'}_i^T S_0^{-1} X X^T (S_0^{-1})^T v'_i) \\
=& \sum_{i=m+1}^{r} {\sigma'}_i^2 \ ||{v'}_i^T S_0^{-1} X||_F^2 \ ,
\end{aligned}$$

which is still a complex function that involves the activation X, the diagonal matrix S_0, the singular vector v'_i and the singular value σ'_i. As a result, compression error is not directly related to the singular value, and the conventional SVD compression by truncating the smallest singular values may lead to suboptimal compression error.

A.2 Pseudocode for `SVD-LLM`

Algorithm 1 shows the pseudocode of `SVD-LLM`. Before compression, `SVD-LLM` randomly collects a small amount of sentences as the calibration data C, it then runs the truncation-aware data whitening process as shown in Algorithm 2 to obtain the set of whitening matrix Set_S for the weight to compress. After that, it runs the SVD and truncation with Set_S on each weight matrix in the LLM. Instead of directly finishing the whole compression, it stores the decomposed matrices and further utilizes these matrices to run the layer-wise closed-form update as shown in Algorithm 3.

A.3 Performance with LoRA Fine-Tuning.

LoRA [13] is a common fine-tuning technique for LLM. It has been applied with pruning-based LLM compression methods such as LLM-Pruner [16] to mitigate accuracy drop after pruning. LoRA can

Algorithm 1 Pseudocode for SVD-LLM

Input: M: Original LLM
Output: M': Compressed LLM by SVD-LLM
procedure SVD-LLM(M)
    Randomly collect several sentences as the calibration data C
    $\text{Set}_S \leftarrow$ TRUNCATION-AWARE DATA WHITENING(M, C)
    $\text{Set}_{SVD} \leftarrow \emptyset$ ▷ Initialize the set of decomposed matrices for the weight to compress
    $\text{Set}_W \leftarrow M$ ▷ Obtain the set of weights in M to compress
    for W **in** Set_W **do**
        $S \leftarrow \text{Set}_S(W)$ ▷ Extract the whitening matrix of current weight W
        $U, \Sigma, V \leftarrow \text{SVD}(WS)$ ▷ Apply singular value decomposition on W
        $\Sigma_1 \leftarrow \text{Trunc.}(\Sigma)$ ▷ Truncate the smallest singular values in Σ
        $\text{Set}_{SVD} \leftarrow (U, \Sigma_1, V) \cap \text{Set}_{SVD}$ ▷ Store the decomposed matrices of the weight in the set
    end for
    $M' \leftarrow$ LAYER-WISE CLOSED-FORM UPDATE$(M, C, \text{Set}_S, \text{Set}_{SVD})$
    return M'
end procedure

Algorithm 2 Pseudocode for Truncation-Aware Data Whitening

Input: M: Original LLM
Input: C: Calibration Data
Output: Set_S: Set of whitening matrices for the weight to compress in M
procedure TRUNCATION-AWARE DATA WHITENING(M, C)
    $\text{Set}_S \leftarrow \emptyset$ ▷ Initialize the set of whitening matrices
    $\text{Set}_W \leftarrow M$ ▷ Obtain the set of weights in M to compress
    for W **in** Set_W **do**
        $X \leftarrow M(W, D)$ ▷ Obtain the input activation of the weight matrix W
        $S \leftarrow \text{Cholesky\_Decomposition}(XX^T)$ ▷ Apply cholesky decomposition on XX^T
        $\text{Set}_S \leftarrow S \cup \text{Set}_S$ ▷ Store the whitening weight matrix in the set
    end for
    return Set_S
end procedure

also be combined with SVD-based LLM compression methods by modifying the forward pass of a linear layer as:

$$Y = W'_u W'_v X \tag{8}$$

where $W'_u = W_u + B_u A_u$, $W'_v = W_v^T + B_v A_v$, and A_u, B_u, A_v, and B_v are low-rank weights fine-tuned using LoRA.

To examine the performance of SVD-LLM in combination with LoRA, we follow the same configuration used in LLM-Pruner [16] to fine-tune LLaMA-7B compressed by SVD-LLM and ASVD under the compression ratios from 20% to 80% with LoRA. The results are shown in Table 9. We have three observations. (1) Comparing SVD-LLM with SVD-LLM + LoRA, under the compression ratio between 20% and 50%, the accuracy enhancement brought by LoRA is limited; as the compression ratio increases, LoRA plays a more significant role in improving the performance of SVD-LLM. (2) Compared to ASVD + LoRA, SVD-LLM + LoRA consistently achieves better accuracy across all the compression ratios. In particular, under 70% compression ratio, the perplexity of SVD-LLM + LoRA is 75% lower than that of ASVD + LoRA. (3) Finally, even without LoRA, SVD-LLM is able to achieve perplexity comparable to ASVD + LoRA, especially under the compression ratios between 20% and 60%.

A.4 Generated Content from the Compressed Model

We also compare some examples of sentences generated by LLaMA-7B compressed with SVD-LLM and ASVD in Table 10. As shown, the sentences generated by the model compressed by SVD-LLM

Algorithm 3 Pseudocode for Layer-Wise Closed-Form Update

Input: M: Original LLM
Input: C: Calibration Data
Input: Set_S: Set of whitening matrices for the weight to compress in M
Input: Set_{SVD}: Set of decomposed matrices for the weight to compress in M
Output: M': Compressed LLM by `SVD-LLM`
procedure LAYER-WISE CLOSED-FORM UPDATE($M, C, \text{Set}_S, \text{Set}_{SVD}$)
    $M' \leftarrow M$ ▷ Initialize M' with M
    $\text{Set}_L \leftarrow M'$ ▷ Obtain the set of encoder and decoder layers in M'
    $X' \leftarrow M'(C)$ ▷ Obtain the input activation of the first layer in M'
    for L **in** Set_L **do**
        $\text{Set}_W \leftarrow L$ ▷ Obtain the set of weights in L to compress
        for W **in** Set_W **do**
            $S \leftarrow \text{Set}_S(W)$ ▷ Extract the whitening matrix of current weight W
            $U, \Sigma_1, V \leftarrow \text{Set}_{SVD}(W)$ ▷ Obtain the decomposed matrices of W from Set_{SVD}
            $U' = WX'X'^T(S^{-1})^T V\Sigma_1(\Sigma_1 V^T S^{-1} X' X'^T (S^{-1})^T V \Sigma_1)^{-1}$ ▷ Closed-form
            $W_u \leftarrow U'(\Sigma_1)^{1/2}, W_v \leftarrow S^{-1}V(\Sigma_1)^{1/2}$ ▷ Obtain two low-rank matrices
            $L(W) \leftarrow L(W_u, W_v)$ ▷ Replace W with W_u and W_v in L
        end for
        $X' \leftarrow L(X')$ ▷ Use the compressed layer to calculate the new input activation X' for the next layer
    end for
    return M'
end procedure

Table 9: Perplexity of LLaMA-7B compressed by `SVD-LLM` and ASVD (w/ and w/o LoRA) on WikiText-2.

METHOD	20%	30%	40%	50%	60%
ASVD	11.14	51	1407	15358	57057
ASVD + LoRA	7.37	10.16	14.86	21.83	44.81
`SVD-LLM`	7.94	9.56	13.11	23.97	53.74
`SVD-LLM` + LoRA	8.28	9.14	10.65	13.26	17.93

exhibit better fluency, relevance, and informativeness compared to that compressed by ASVD. More importantly, when the compression ratio is increased to 30%, the previous state-of-the-art method ASVD completely loses its generation ability. In contrast, even when the compression ratio is up to 40%, `SVD-LLM` is still capable of generating complete sentences.

A.5 More Experiments on compressing different LLMs

We also evaluate the performance of different LLMs, including OPT-6.7B, LLaMA 2-7B, Mistral-7B, and Vicuna-7B under 20% compression ratio on seven common sense reasoning datasets. The results are shown in Table 11. `SVD-LLM` performs better than the best-performing baseline in most of the datasets across different LLMs and even achieves 32% higher average accuracy on OPT-6.7B.

Table 10: An example of contents generated by the compressed LLaMA-7B by SVD-LLM and and under different compression ratios. The input is marked in bold and the normal texts are the generated sentences.

RATIO	SVD-LLM	ASVD
Original	**Building a website can be done in 10 simple steps:** Step 1: Register an domain name; Step 2: Get a hosting package; Step 3: Develop a website; Step 4: Design the website; Step 5: Make the website live; Step 6: Perform on-page SEO; Step 7: Promote the website; Step 8: Evaluate the results; Step 9: Improve the website; Step 10: Build a website repeat ...	
20%	**Building a website can be done in 10 simple steps:** 1) Identifying an appropriate theme. 2) Determining which types of Web site would be of interest and which could be implemented with minimal overhead and resources. 3) Identify the audience 4) Choose Web site software. 5) Select Web site developers. 6) Design Web site 7) Identify Web site technology 8) Design Web site layout 9) Define Web site aesthetics and 10) Define Web site access controls ...	**Building a website can be done in 10 simple steps:** 1, Build up your website. You will need to select a name for your website. As a beginner, it is a good idea to select your company name or product name as your website name. You can use your real company or product name or you can create an unusual business name. This will make the name of your business name consistent throughout your website. 2, Decide on your business name. You will be choosing the official name of your business for all activities that represent your business. This includes your business name, products, your website, and any ...
30%	**Building a website can be done in 10 simple steps:** 1. Preliminaries and planning 2. Designing 3. Prototyping 4. Production 5. Operation...	**Building a website can be done in 10 simple steps:** To: With A Web You Are Into 2 Zim 40 Wh I M T If Y As An Onl You Orwice There W Os You Hcimp ...
40%	**Building a website can be done in 10 simple steps:** 2 days of website plan. If you want to set a web site. This is the 1st step. In the 2 days . It wants the company to develop a website to help the investors and investors. He will have another investor. He wants to help others. The company is working to grow, and create a name for some investors and investors. 100 million US dollars It is used to build a hotel and a hotel. In order to obtain their cooperation, investors, investors, investors in .	**Building a website can be done in 10 simple steps:** Front 1222ED825333 3333333333333333etennenn eenneenneSecondinjustanyanojunatajin ...

Table 11: Zero-shot performance comparison of OPT-6.7B, LLaMA 2-7B, Vicuna-7B, Mistral-7B compressed by SVD-LLM and baselines under 20% compression ratio on seven common sense reasoning datasets (measured by both individual and average accuracy (↑)). The best performance is marked in bold. The relative performance gain compared to the best-performing baseline is marked in green color inside bracket.

LLM	METHOD	Openb.	ARC_e	WinoG.	HellaS.	ARC_c	PIQA	MathQA	**Average↑**
OPT-6.7B	SVD	0.14	0.27	0.51	0.25	0.23	0.54	0.21	0.31
	FWSVD	0.15	0.26	0.49	0.26	0.20	0.52	0.21	0.30
	ASVD	0.14	0.40	0.51	0.30	0.20	0.59	0.22	0.34
	SVD-LLM	**0.24**	**0.60**	**0.60**	**0.45**	**0.28**	**0.73**	**0.24**	**0.45** (↑32%)
LLaMA 2-7B	SVD	0.15	0.27	0.49	0.26	0.23	0.52	0.20	0.30
	FWSVD	0.12	0.25	0.49	0.25	0.22	0.52	0.21	0.30
	ASVD	0.25	0.31	0.60	0.41	**0.32**	**0.72**	0.23	**0.41**
	SVD-LLM	**0.26**	**0.50**	**0.60**	**0.41**	0.26	0.66	**0.23**	**0.41** (↑0%)
Mistral-7B	SVD	0.14	0.25	0.52	0.26	0.23	0.54	0.20	0.30
	FWSVD	0.15	0.28	0.52	0.26	0.21	0.53	0.21	0.31
	ASVD	**0.21**	0.51	0.58	**0.42**	0.25	0.50	**0.26**	0.39
	SVD-LLM	0.17	**0.55**	**0.58**	0.36	**0.25**	**0.67**	0.21	**0.40** (↑3%)
Vicuna-7B	SVD	0.15	0.26	0.50	0.26	0.23	0.52	0.20	0.30
	FWSVD	0.14	0.27	0.49	0.26	0.22	0.53	0.20	0.30
	ASVD	0.21	0.53	0.55	0.39	**0.30**	0.41	**0.23**	0.37
	SVD-LLM	**0.23**	**0.51**	**0.58**	**0.40**	0.28	**0.67**	0.22	**0.41** (↑11%)