# SVD-LLM: TRUNCATION-AWARE SINGULAR VALUE DECOMPOSITION FOR LARGE LANGUAGE MODEL COMPRESSION

**Anonymous authors**
Paper under double-blind review

## 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 parameter update with sequential low-rank approximation to compensate for the accuracy degradation after compression. We evaluate `SVD-LLM` on 10 datasets and seven models from three different LLM families at three 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 generation (Zhao et al., 2023; Gozalo-Brizuela and Garrido-Merchán, 2023). Despite such capabilities, the democratization of LLMs is primarily restricted by their substantial resource demands (Wan et al., 2023; Wang et al., 2024; Zhou et al., 2024). One of the most effective techniques to reduce resource demands of LLMs is model compression (Zhu et al., 2023). To avoid resource-intensive retraining, LLM compression is often conducted in a post-training manner, and methods based on quantization (Yuan et al., 2024; Huang et al., 2024), unstructured pruning (Frantar and Alistarh, 2023), and structured pruning (Ma et al., 2023; Ashkboos et al., 2024; Zhong et al., 2024) specifically designed for LLMs have been intensively studied. Regardless of their success, these techniques have their own constraints, such as hardware dependency and low inference speedup. Compared to those techniques, compression techniques based on low-rank approximation, such as Singular Value Decomposition (SVD) are not limited by those constraints. Moreover, the KV cache of LLMs compressed via SVD at runtime can also be reduced.

Despite these advantages, the potential of SVD for LLM compression has not been thoroughly explored. A few SVD-based LLM compression methods such as FWSVD (Hsu et al., 2022) and ASVD (Yuan et al., 2023) have recently been proposed. However, these methods exhibit severe performance degradation when model compression ratio[1] increases. Such limitation can be attributed to two fundamental issues involved in their approaches: ❶ **Misalignment between SVD Truncation and Compression Loss**[2]: All existing methods fail to 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 Model Parameter Update after SVD Truncation**: as model compression ratio increases, the number of singular values that need to be

[1]Model compression ratio refers to the percentage of parameter reduction achieved through compression.

[2]Given input activation $X$, original weight matrix $W$ and its compressed version $W'$ in the LLM, the SVD compression loss (Yuan et al., 2023; Hsu et al., 2022) refers to $||WX - W'X||_F$ in the form of Frobenius loss.

truncated in SVD increases as well. To compensate for the accuracy degradation caused by truncating a larger number of singular values, it becomes necessary to update the remaining parameters of 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 SVD-based post-training LLM compression method, which we refer to as SVD-LLM, that effectively addresses the two fundamental issues of 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 **Parameter Update with Sequential Low-rank Approximation:** to compensate for accuracy degradation after compression, SVD-LLM sequentially fine-tunes the decomposed low-ranking matrices for a global accuracy recovery.

We compare SVD-LLM with both state-of-the-art SVD-based LLM compression methods as well as pruning and quantization-based LLM compression methods. To demonstrate the generability of SVD-LLM, we conduct our evaluation on a total of 10 datasets and seven models from three different LLM families (LLaMA, OPT, and Mistral) at three different scales (7B, 13B, 30B), and evaluate the performance of SVD-LLM on both GPU and CPU. We highlight three of our findings:

- SVD-LLM consistently outperforms state-of-the-art SVD-based LLM compression methods across all 10 datasets, three LLM families at three scales by a large margin.
- SVD-LLM outperforms state-of-the-art structured pruning-based LLM compression methods with up to 57% lower perplexity under 7GB memory budget. It also outperforms state-of-the-art 1-bit post-training quantization-based LLM compression methods. More importantly, when combined with 2-bit post-training quantization, SVD-LLM outperforms state-of-the-art 1-bit training-required quantization-based LLM compression method, presenting a new way to achieve state-of-the-art compression performance without incurring expensive retraining.
- LLMs compressed by SVD-LLM are able to achieve inference speedup and memory reduction when deployed on real hardware. In particular, compared to the original LLMs, models compressed by SVD-LLM are able to achieve 3.1 times higher throughput on A100 GPU and 2.3 times higher throughput on AMD EPYC 7643 CPU. Moreover, the weight memory saving brought by SVD-LLM are near linear to the compression ratio. At the same time, SVD-LLM is able to reduce runtime KV cache memory without additional accuracy drop.

## 2 Related Work

**Large Language Model Compression:** LLMs in general contain billion-scale parameters. Applying conventional model compression methods for LLMs is unfeasible as they necessitate resource-intensive retraining. Given that, post-training methods that avoid retraining 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 of an LLM to zero without changing its shape. 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. A notable contribution is LLM-Pruner (Ma et al., 2023), which 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, it suffers from accuracy degradation especially under high compression ratios, and many follow-up works such as SliceGPT (Ashkboos et al., 2024) and BlockPruner (Zhong et al., 2024) have been proposed with better compression performance. Quantization methods, on the other hand, achieve model compression by reducing the precision of weight matrices of an LLM. However, quantization is not only difficult to achieve the desired inference speedup (Lin et al., 2024), but also has the drawback of only providing a limited range of compression options, typically ranging from 2 to 8 bits. This limited range prevents full utilization of the available memory budget. Recent studies, including

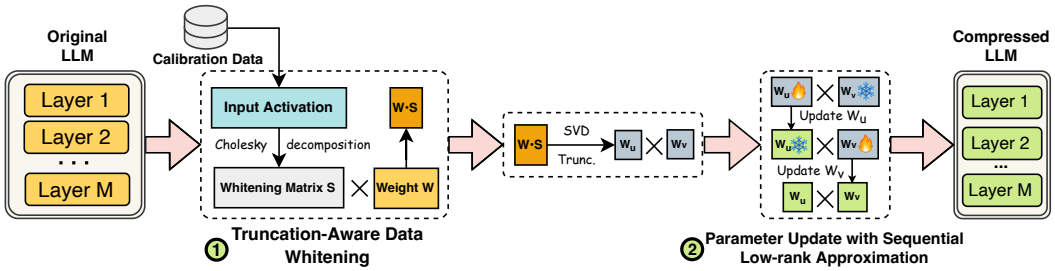



Figure 1: Overview of SVD-LLM.

PB-LLM (Yuan et al., 2024), and BiLLM (Huang et al., 2024) have focused on 1-bit post-training quantization. Nevertheless, these approaches often lead to severe accuracy degradation, indicating that 1-bit quantization remains a challenging aspect of LLM compression.

**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 (Golub et al., 1987). SVD is commonly used to compress models. Previous work Drone (Chen et al., 2021) has successfully compressed the Bert model via SVD. In LLM compression, for example, AAFM (Yu and Wu, 2023) dynamically identifies the compressed model architecture and selectively compresses the output features of individual linear layers instead of the model weights. LoSparse (Li et al., 2023) compresses the weight matrix by the sum of a low-rank matrix and a sparse matrix. ARS (Gao et al., 2024) dynamically allocates the rank of the SVD compression based on the importance of weights in the LLM. Directly applying SVD on the weight matrix without considering the importance of the weights leads to a large LLM compression error. To address this issue, Hsu et al. (2022) propose FWSVD, which introduces Fisher information to weigh the importance of parameters. To make up for the lack of motivation to apply SVD in the context of LLM compresion, the authors in FWSVD also provide an analysis of the impact of SVD compression to the final compression performance. However, FWSVD requires a complex gradient calculation that demands substantial computing and memory resources for LLM compression. Another problem of vanilla SVD is the distribution of activation can affect the compression error. To address this issue, Yuan et al. (2023) propose ASVD, which scales the weight matrix by a diagonal matrix that normalizes the impact of input channels on the weights. However, all of the SVD-based LLM compression methods, particularly including 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 weights 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.

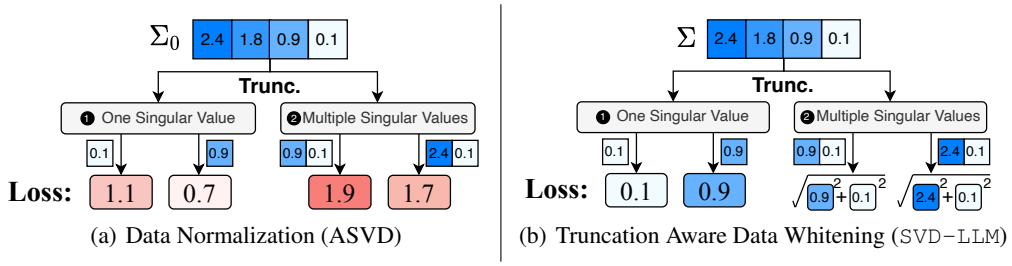
## 3 SVD-LLM

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 (Frantar and Alistarh, 2023; Yuan et al., 2023; Xiao et al., 2023), SVD-LLM uses a random set of sentences as calibration data to generate activation for truncation-aware data whitening. Given the generated activation, SVD-LLM calculates the whitening matrix $S$ through Cholesky decomposition, and then performs SVD to truncate the multiplication of weight matrices $W$ and whitening matrix $S$ to compress the LLM. After truncation, SVD-LLM updates the remaining model parameters with sequential low-rank approximation to recover accuracy. In the following, we describe both truncation-aware data whitening and parameter update with sequential low-rank approximation in detail. The pseudocode of SVD-LLM is provided in Appendix A.8.

### 3.1 Truncation-Aware Data Whitening

**Motivation:** Due to high variance of input activation, simply applying vanilla SVD for LLM compression leads to severe accuracy degradation (Yuan et al., 2023). To address this issue, existing methods (Yuan et al., 2023; Hsu et al., 2022) formulate LLM compression as an optimization problem with the following objective:

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




(a) Data Normalization (ASVD) (b) Truncation Aware Data Whitening (SVD-LLM)

Figure 2: Compression loss ($L = ||WX - W'X||_F$) of different data preprocessing methods.

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

Although existing methods attempt to reduce this compression loss during their SVD truncation, they all fail to establish a direct relationship between singular values and compression loss. As a consequence, truncating smaller singular values in SVD could lead to significant compression loss. Taking ASVD (Yuan et al., 2023) as an example, 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 $U_0$, $\Sigma_0$, and $V_0$. Lastly, ASVD truncates the smallest singular values in $\Sigma_0$ 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 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 orthonormal 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 (Meyer, 2000). Then we perform SVD on $WS$ to obtain the decomposed matrices $U, \Sigma, V$, 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 $\Sigma$ are truncated to obtain the compressed weight matrix $W' = U \times \text{Trunc.}(\Sigma) \times V^T \times S^{-1}$. To save the model memory, SVD-LLM replace the original matrix $W$ with the two smaller and low-ranking ones $W_u = U \times [\text{Trunc.}(\Sigma)]^{\frac{1}{2}}, W_v = [\text{Trunc.}(\Sigma)]^{\frac{1}{2}} \times V^T \times S^{-1}$ in the compressed LLM.

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

In the following, 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). To further illustrate the feasibility of our proposed technique in compressing LLM, we also provide the spectrum analysis of the singular values obtained by our technique in Appendix A.4.

**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 $WS$ to reduce its rank for compression:

$$L_i = \|WX - W'X\|_F = \left\|WSS^{-1}X - SVD(WS)S^{-1}X\right\|_F = \left\|(WS - SVD(WS))S^{-1}X\right\|_F$$
$$= \left\|\sigma_i u_i v_i^T S^{-1}X\right\|_F = \sigma_i \; trace\left(u_i v_i^T S^{-1} X X^T \left(S^{-1}\right)^T v_i u_i^T\right)^{\frac{1}{2}} \tag{3}$$

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

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

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

*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:

$$L_i = \sigma_i \; trace(u_i v_i^T v_i u_i^T)^{\frac{1}{2}} = \sigma_i \, \text{trace}\left(u_i \left(v_i^T v_i\right) u_i^T\right)^{\frac{1}{2}} = \sigma_i \, \text{trace}\left(u_i u_i^T\right)^{\frac{1}{2}} = \sigma_i \tag{5}$$

Therefore, $L_i$ of truncating $\sigma_i$ equals to the singular value $\sigma_i$ 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 $\Sigma$ for compression, the square of the loss $L$ is:

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

The squared loss $L^2$ equals to the sum of the squared singular values (More detailed derivation is in Appendix A.2). Truncating the smallest singular values achieves the lowest compression loss. □

Apart from aligning the SVD truncation with the compression loss, our data whitening method can even obtain the optimal minimization of compression loss, which has been achieved by Drone on small model but unable to be applied for LLM. More detailed analysis about the optimality of SVD-LLM and comparison with Drone are provided in Appendix A.10.

### 3.2 Parameter Update with Sequential Low-rank Approximation

**Motivation:** Although aligning SVD truncation with the compression loss, as done in Section 3.1 helps to preserve the accuracy degradation during compression, with the increase of compression ratio, the performance of the compressed LLM may still become worse since more and more larger singular values will get truncated by SVD compression. Therefore, it is necessary to update the remaining parameters in the compressed LLM.

**Key Design:** Driven by previous post-training LLM compression works (Ma et al., 2023), SVD-LLM uses LoRA fine-tuning to update the remaining weight parameters of the compressed LLM for accuracy recovery. Specifically, suppose that we decompose the original weight matrix $W$ into two low-ranking matrices $W_u, Wv$ with SVD-LLM, as discussed in the pseudocode in Appendix A.8. Instead of directly fine-tuning the compressed weight matrix $W' = W_u \times W_v$, which would break its low-rank structure, we treat $W_u$ and $W_v$ as two linear layers and update them sequentially as follows:

$$Y = W_u' \times W_v' \times X \tag{7}$$

where $W_u' = W_u + B_u A_u$, $W_v' = W_v + B_v A_v$, and $A_u$, $B_u$, $A_v$, and $B_v$ are matrices used for LoRA fine-tuning. Simultaneously fine-tuning $W_u$ and $W_v$ will not guarantee a decrease in fine-tuning loss. This is because the derivatives of $W_u$ and $W_v$ are interdependent during the fine-tuning process, meaning that changes in one matrix may counteract or interfere with the optimization of the other. As

Table 1: Performance of LLaMA-7B compressed by SVD-LLM (SVD-LLM * denote the version without parameter update with sequential low-rank approximation) and baselines under different compression ratio (corresponding weight memory is listed inside bracket) on two language modeling datasets (measured by perplexity (↓)), eight common sense reasoning datasets (six measured by both individual and average accuracy (↑), TruthfulQA measured by BLEU score (↑), and GSM8K measured by Exact Match Accuracy (↑)). The best performance is marked in bold. The relative performance gain compared to the best-performing baseline is marked in green inside bracket.

| Ratio (Mem.) | Method | WikiText-2↓ | C4↓ | Openb. | ARC_e | WinoG. | HellaS. | PIQA | MathQA | **Average**↑ | TruthfulQA↑ | GSM8K↑ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0% (13.5 GB) | Original | 5.68 | 7.34 | 0.34 | 0.75 | 0.70 | 0.57 | 0.79 | 0.27 | 0.57 | 0.30 | 0.09 |
| 20% (10.2 GB) | SVD | 20061 | 18800 | 0.05 | 0.04 | 0.01 | 0.03 | 0.02 | 0.03 | 0.03 | 0.00 | 0.00 |
| | FWSVD | 1727 | 1511 | 0.09 | 0.11 | 0.05 | 0.08 | 0.10 | 0.05 | 0.08 | 0.00 | 0.00 |
| | ASVD | 11.14 | 15.93 | 0.29 | 0.53 | 0.64 | 0.41 | 0.68 | 0.17 | 0.45 | 0.21 | 0.04 |
| | SVD-LLM * | 7.94 (↓29%) | 15.84 (↓1%) | 0.31 | 0.62 | 0.61 | 0.45 | 0.71 | 0.21 | 0.49 (↑9%) | 0.26 (+0.05) | 0.05 (+0.01) |
| | SVD-LLM | **6.73** (↓40%) | **9.81** (↓38%) | **0.33** | **0.67** | **0.69** | **0.55** | **0.79** | **0.26** | **0.55** (↑22%) | **0.28** (+0.07) | **0.08** (+0.04) |
| 40% (7.76 GB) | SVD | 52489 | 47774 | 0.04 | 0.04 | 0.05 | 0.01 | 0.03 | 0.02 | 0.03 | 0.00 | 0.00 |
| | FWSVD | 18156 | 12847 | 0.06 | 0.05 | 0.02 | 0.00 | 0.05 | 0.03 | 0.04 | 0.00 | 0.00 |
| | ASVD | 1407 | 1109 | 0.08 | 0.11 | 0.09 | 0.08 | 0.13 | 0.08 | 0.10 | 0.01 | 0.00 |
| | SVD-LLM * | 13.73 (↓99%) | 75.42 (↓93%) | 0.25 | 0.33 | 0.61 | 0.40 | 0.63 | 0.12 | 0.39 (↑290%) | 0.17 (+0.17) | 0.02 (+0.02) |
| | SVD-LLM | **8.18** (↓99%) | **12.61** (↓99%) | **0.29** | **0.59** | **0.68** | **0.52** | **0.69** | **0.20** | **0.50** (↑400%) | **0.24** (+0.23) | **0.07** (+0.07) |
| 60% (5.35 GB) | SVD | 105474 | 106976 | 0.01 | 0.03 | 0.01 | 0.00 | 0.01 | 0.02 | 0.01 | 0.00 | 0.00 |
| | FWSVD | 32194 | 29292 | 0.06 | 0.02 | 0.01 | 0.01 | 0.02 | 0.03 | 0.03 | 0.00 | 0.00 |
| | ASVD | 57057 | 43036 | 0.05 | 0.04 | 0.06 | 0.09 | 0.08 | 0.05 | 0.06 | 0.00 | 0.00 |
| | SVD-LLM * | 66.62 (↓99%) | 471.83 (↓99%) | 0.10 | 0.05 | 0.17 | 0.10 | 0.21 | 0.04 | 0.11 (↑83%) | 0.01 (+0.01) | 0.00(+0.00) |
| | SVD-LLM | **13.31** (↓99%) | **19.72** (↓99%) | **0.27** | **0.52** | **0.60** | **0.41** | **0.66** | **0.21** | **0.45** (↑650%) | **0.04** (+0.04) | **0.04**(+0.04) |
| 80% (2.58 GB) | SVD | 687291 | 708243 | 0.00 | 0.04 | 0.02 | 0.01 | 0.01 | 0.00 | 0.01 | 0.00 | 0.00 |
| | FWSVD | 96872 | 89243 | 0.01 | 0.02 | 0.00 | 0.06 | 0.09 | 0.00 | 0.03 | 0.00 | 0.00 |
| | ASVD | 80425 | 67927 | 0.04 | 0.03 | 0.03 | 0.07 | 0.10 | 0.01 | 0.05 | 0.00 | 0.00 |
| | SVD-LLM * | 1349 (↓98%) | 6224 (↓91%) | 0.07 | 0.01 | 0.12 | 0.10 | 0.07 | 0.06 | 0.07 (↑40%) | 0.00 (+0.00) | 0.00 (+0.00) |
| | SVD-LLM | **31.79** (↓99%) | **43.71** (↓99%) | **0.21** | **0.33** | **0.51** | **0.29** | **0.53** | **0.21** | **0.35** (↑600%) | **0.14** (+0.14) | **0.02** (+0.02) |

a result, the overall effect on the fine-tuning loss function can be unpredictable and may not always lead to a reduction in loss. Therefore, as depicted in Figure 1, we propose a sequential fine-tuning strategy in SVD-LLM. To better illustrate the effectiveness of our sequential fine-tuning strategy compared to the normal simultaneous fine-tuning, we provide a comparison in Appendix A.12. Specifically, we first freeze matrix $W_v$ and fine-tune $W_u$ with LoRA for all the decomposed weight matrices in the LLM. We then perform the second-round LoRA fine-tuning on the matrix $W_v$ while freezing the updated weight matrix $W_u$. Finally, we fuse the $A_u$, $B_u$, $A_v$, and $B_v$ matrices into $W_u$ and $W_v$ as the final compressed matrices.

## 4 Experiments and Analysis

**Experiment Setup.** We compare SVD-LLM against three groups of methods. (1) We compare SVD-LLM with vanilla SVD and state-of-the-art SVD-based LLM compression methods: FWSVD (Hsu et al., 2022), ASVD (Yuan et al., 2023) (Section 4.1) and FLAP (Appendix A.11). (2) We compare SVD-LLM with other types of LLM compression methods. These include three state-of-the-art pruning-based LLM compression methods: LLM-Pruner (Ma et al., 2023), SliceGPT (Ashkboos et al., 2024), and BlockPruner (Zhong et al., 2024), and three state-of-the-art quantization-based LLM compression methods: PB-LLM (Yuan et al., 2024), BiLLM (Huang et al., 2024), and OneBit (Xu et al., 2024) (Section 4.4). (3) Lastly, we compare SVD-LLM against smaller LLM StableLM-3B (Tow et al.) pre-trained from scratch (Appendix A.7). More experimental setups are provided in Appendix A.3 due to page limit.

### 4.1 Comparison with State-of-the-Art SVD-based LLM compression Methods

First, we compare the performance of SVD-LLM with state-of-the-art SVD-based LLM compression methods from three aspects: (1) performance under different compression ratios, (2) performance on different LLMs, and (3) performance on LLMs with larger scales. The compression speed analysis is provided in Appendix A.5. Driven from FLAP An et al. (2023), to ensure a fair comparison, we not only evaluate the integrated SVD-LLM to show its best accuracy, but also compare SVD-LLM without without parameter update with sequential low-rank approximation (denoted as SVD-LLM *) with other baselines under the no LoRA fine-tuning setting. Example contents generated by the compressed LLMs are included in Appendix A.6.

**Performance under Different Compression Ratios.** We first evaluate the performance of LLaMA-7B compressed by SVD-LLM and the SVD-based baselines under compression ratios ranging from 20% to 80% on all 10 datasets. Table 1 summarizes the results. Both SVD-LLM and SVD-LLM * without LoRA fine-tuning consistently outperforms vanilla SVD, FWSVD and ASVD across all the

Table 2: Perplexity (↓) of SVD-LLM (SVD-LLM * denote the version without parameter update with sequential low-rank approximation) and baselines on WikiText-2 and the average accuracy (↑) of the six common sense reasoning datasets of four different LLMs – OPT-6.7B, LLaMA 2-7B, Mistral-7B, and Vicuna-7B – under 20% compression ratio. The relative performance gain compared to the best-performing baseline is marked in green color inside bracket.

| | OPT-6.7B | | LLaMA 2-7B | | Mistral-7B | | Vicuna-7B | |
|---|---|---|---|---|---|---|---|---|
| Method | Perplexity↓ | Accuracy↑ | Perplexity↓ | Accuracy↑ | Perplexity↓ | Accuracy↑ | Perplexity↓ | Accuracy↑ |
| Original | 10.86 | 0.52 | 5.47 | 0.57 | 5.25 | 0.61 | 6.78 | 0.56 |
| SVD | 66275 | 0.03 | 18192 | 0.09 | 159627 | 0.03 | 18644 | 0.05 |
| FWSVD | 14559 | 0.06 | 2360 | 0.12 | 6357 | 0.08 | 2758 | 0.09 |
| ASVD | 82.00 | 0.32 | 10.10 | 0.36 | 13.72 | 0.32 | 16.23 | 0.33 |
| SVD-LLM * | 16.04 (↓80%) | 0.41 (↑28%) | 8.50 (↓16%) | 0.53 (↑47%) | 10.21 (↓26%) | 0.42 (↑24%) | 8.41 (↓48%) | 0.51 (↑55%) |
| SVD-LLM | **11.61** (↓86%) | **0.48** (↑50%) | **6.07** (↓40%) | **0.56** (↑56%) | **6.01** (↓56%) | **0.59** (↑84%) | **7.43** (↓54%) | **0.54** (↑64%) |

compression ratios. In particular, when the compression ratio reaches 40% and above, SVD-LLM reduces the perplexity by more than 99% on two language modeling datasets and achieves over 400% higher average accuracy on six classification datasets. More importantly, the results on two generation datasets ((TruthfulQA, GSM8K) of all three baselines when compression ratios are 60% and above are zero, meaning that the compressed LLMs totally lose their generation ability. In contrast, SVD-LLM still outputs good generation even under the 80% compression ratio. These results indicate that SVD-LLM is more effective in compressing LLMs for more resource-constrained devices such as smartphones and IoT devices.

**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 from three different LLM families – OPT-6.7B (OPT family), LLaMA 2-7B (LLaMA family), Mistral-7B (Mistral family), and Vicuna-7B (LLaMA family) – under 20% compression ratio on WikiText-2 and six classification datasets. As shown in Table 2, SVD-LLM consistently outperforms baselines on all four LLMs, and exhibits more stable performance across different LLMs, especially compared to vanilla SVD and FWSVD.

Table 3: Perplexity (↓) of SVD-LLM (SVD-LLM * denote the version without parameter update with sequential low-rank approximation) and baselines on WikiText-2 and the average accuracy (↑) of the six classification datasets of LLaMA-13B and LLaMA-30B under 20% compression ratio. The relative performance gain compared to the best-performing baseline is marked in green color inside bracket.

| | LLaMA-13B | | LLaMA-30B | |
|---|---|---|---|---|
| Method | Perplexity↓ | Accuracy↑ | Perplexity↓ | Accuracy↑ |
| Original | 5.09 | 0.59 | 4.10 | 0.61 |
| SVD | 946.31 | 0.21 | 54.11 | 0.33 |
| FWSVD | 15.98 | 0.43 | 20.54 | 0.42 |
| ASVD | 6.74 | 0.54 | 22.71 | 0.44 |
| SVD-LLM * | 6.61 (↓2%) | 0.54 (↑0%) | 5.63 (↓75%) | 0.57 (↑30%) |
| SVD-LLM | **5.18** (↓23%) | **0.58** (↑7%) | **4.54** (↓80%) | **0.61** (↑39%) |

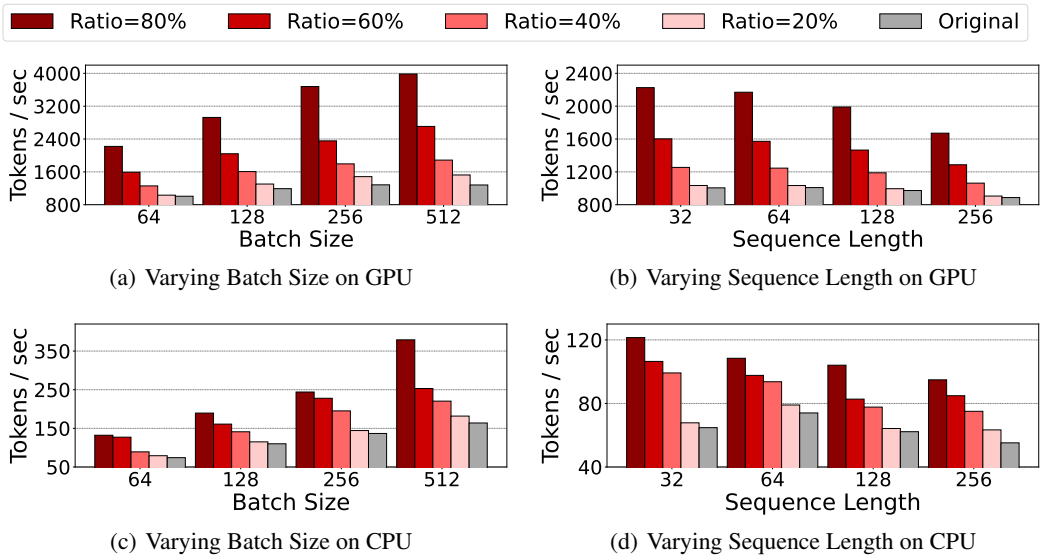
**Performance on LLMs with Larger Scales.** To examine the generability of SVD-LLM on LLMs with larger scales, we compare the performance between SVD-LLM and the baselines on LLaMA-13B, and LLaMA-30B under 20% compression ratio. As shown in Table 3, SVD-LLM consistently outperforms vanilla SVD, FWSVD, and ASVD on both of the two model sizes.

### 4.2 Inference Efficiency of SVD-LLM

**Theoretical Analysis of Inference Efficiency.** Suppose SVD-LLM compresses the weight matrix $W \in \mathbb{R}^{d\times n}$ into two low-ranking matrices $W_u \in \mathbb{R}^{d\times r}, W_v \in \mathbb{R}^{r\times n}$, as discussed in the pseudocode in Appendix A.8. The compression ratio $R_w$ will be $R_w = 1 - \frac{(d+n)r}{dn}$.

(1) Compute Complexity Analysis: Given the input $X \in \mathbb{R}^{n\times d}$, instead of recalculating the full weight matrix $W' = W_u \times W_v$ and then compute the output $W' \times X$, we calculate the intermediate state $M = W_v \times X$ and then compute the output $Y = W_u \times M$. In this way, the compute complexity will be reduced from original $O\left(d^2n\right)$ to $O\left(d^2r + rnd\right)$. If we set the compression ratio $R_w = 50\%$, since $R_w = 1 - \frac{(d+n)r}{dn}$, we have, $r = \frac{dn}{2(d+n)}$. The compute complexity will be $O\left(d^2r + rnd\right) = O(rd(d+n)) = O\left(\frac{d^2n}{2}\right) = \frac{1}{2}O\left(d^2n\right)$, which reduces 50%. Similarly, given a compression ratio $R_w$, the compute complexity will also be reduced to $1 - R_w$ times of the original.

(2) Inference Memory Analysis: Since SVD-LLM does not recalculate the full weight $W' = W_u \times W_v$, the weight memory will still be reduced to $1 - R_w$ times of the original one during inference. Additionally, SVD-LLM is able to reduce the runtime KV cache memory without further losing




(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 3: Throughput (Tokens/sec) of original LLaMA-7B and its compressed version by SVD-LLM under different 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.

accuracy. Specifically, instead of keeping $W_u \times W_v \times X$ into the KV cache, SVD-LLM provides the option to store the intermediate result $M = W_v \times X$ into the KV cache and recompute the original key and value states with the decomposed weight matrix $W_u$ if required. Therefore, the memory of the runtime KV cache will be reduced to $\frac{r}{d} = (1 - R_w) \times \frac{d}{n+d}$ times of the original. This a trade-off since the Floating Point Operations (FLOPs) will get increased and the inference could be slow. We leave the speedup a future work.oreover, since $W_u$ is already stored as the weight matrix in the decomposed LLM, the original intermediate state matrix can still be recovered by $Y = W_u \times M$ without accuracy drop. Therefore, SVD-LLM provides a unified solution that combines model compression and KV cache compression into a single process.

**Inference Speedup on Hardware.** To quantify the inference speedup achieved by SVD-LLM, we measure the numbers of tokens that the original LLaMA-7B and its compressed version by SVD-LLM generate per second with different batch sizes and sequence lengths on a single NVIDIA A100 GPU and a single AMD EPYC 7643 CPU. As shown in Figure 3, SVD-LLM consistently ensures an enhancement in the generation speed across all the compression ratios. More importantly, the enhancement becomes more significant as the batch size increases and the sequence length decreases, resulting in a maximum speedup of 1.2x on GPU and 1.1x on CPU under 20% compression ratio, 1.7x on GPU and 1.5x on CPU under 40% compression ratio, 2.1x on GPU and 1.64x on CPU under 60% compression ratio, and 3.1x on GPU and 2.3x on CPU under 80% compression ratio.

**Inference Memory Reduction on Hardware.** In this experiment, we evaluate the inference memory saving, including the compressed weight memory and the runtime KV cache memory saving on a single A100 GPU. 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 reduction. The results are illustrated in Figure 4 where the memory reduction from the dotted line to the blue bars comes mainly from model compression and the memory reduction from the blue to the yellow bars comes mainly from the reduced footprint of the KV cache. As shown, the weight memory saving brought by SVD-LLM is near linear to the compression ratio, which meets other previous theoretical analyses. Moreover, SVD-LLM is able to save additional 51% memory from its KV cache under 80% compression ratio.

### 4.3 Ablation Study

In this section, we provide three ablation studies of SVD-LLM while more are provided in Appendix A.12 due to page limit.

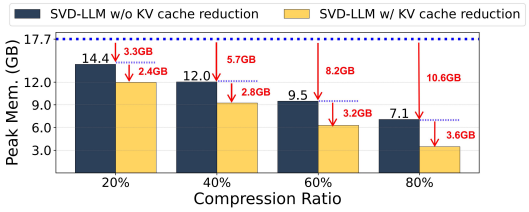



Figure 4: Peak memory to generate 128 tokens with batch size of 32 using LLaMA-7B compressed by SVD-LLM w/ and w/o KV-cache reduction. The dotted line indicates the peak memory of the original LLaMA-7B. The memory reduction from the dotted line to the blue bars mainly comes from the model compression. The memory reduction from the blue to the yellow bars mainly comes from the reduced footprint of the KV cache.

Table 4: Perplexity (↓) of compressed LLaMA-7B on WikiText-2 under different compression ratios. SVD-LLM (W) denotes the version of SVD-LLM with truncation-aware data whitening only; SVD-LLM (U) denote the version of SVD-LLM with parameter update with sequential low-rank approximation only; SVD-LLM (W+U) denotes the version of SVD-LLM with both truncation-aware data whitening and parameter update with sequential low-rank approximation. The relative performance gain compared to ASVD is marked in green color inside bracket.

| METHOD | 20% | 40% | 60% |
|---|---|---|---|
| ASVD | 11.14 | 1407 | 57057 |
| SVD-LLM (W) | 7.94 (↓29%) | 13.11 (↓99%) | 42.30 (↓99%) |
| SVD-LLM (U) | 10.12 (↓9%) | 19.28 (↓99%) | 49.88 (↓99%) |
| SVD-LLM (W+U) | **6.73** (↓40%) | **8.18** (↓99%) | **13.31** (↓99%) ) |

**Modular Sensitivity Study.** We conduct ablation studies to evaluate the separate contributions of the two key components (i.e., truncation-aware data whitening and parameter update with sequential low-rank approximation) 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 normal SVD truncation and parameter update with sequential low-rank approximation; and SVD-LLM (W+U) denote the version of SVD-LLM with both truncation-aware data whitening and parameter update with sequential low-rank approximation. As shown in Table 4. We have three observations. (1) SVD-LLM (W), SVD-LLM (U) and SVD-LLM (W+U) consistently outperform ASVD across all the compression ratios. Notably, when the compression ratio is at and above 40%, all variants reduce the perplexity by more than 99% compared to ASVD. (2) SVD-LLM (W+U) consistently outperforms SVD-LLM (U) across all compression ratios and SVD-LLM (W) achieves a lower perplexity compared to SVD-LLM (U) across all compression ratios, highlighting the effectiveness of truncation-aware data whitening component in SVD-LLM. (3) With the increase of compression ratio, SVD-LLM (W+U) achieves a much lower perplexity compared to SVD-LLM (W), highlighting the importance of combining both components in SVD-LLM when the compression ratio increases.

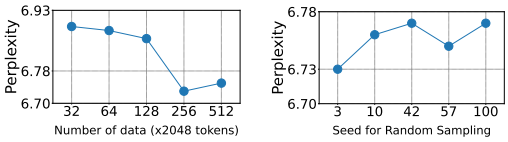
**Impact of Calibration Data.** Next, we examine the impact of calibration data on SVD-LLM. Figure 5 and Table 6 summarize the performance of compressed LLaMA-7B when changing three key characteristics of the calibration data: (1) the number of the calibration data, (2) the seed used to randomly sample the calibration data, and (3) the data set from which the calibration data is sampled. As shown, the changes on calibration data incur no more than 2% to the final performance, demonstrating that the sensitivity of SVD-LLM on calibration data is limited.

**Impact of Updating Order.** We finally examine the impact of updating order in parameter update with sequential low-rank approximation component to the final performance of the compressed LLM. Table 5 summarizes the performance of compressed LLaMA-7B under 20% to 80% compression ratios on WikiText-2 with different updating order. As shown, there is only a small difference of the final performance between updating matrix $U$ first and updating matrix $V$ first, indicating SVD-LLM is not sensitive to the updating order.

### 4.4 COMPARISON WITH OTHER TYPES OF LLM COMPRESSION METHODS

SVD-LLM is orthogonal to other post-trainingLLM compression methods including quantization and pruning. In this experiment, we compare the performance of SVD-LLM with the state-of-the-art structured pruning-based and quantization-based LLM compression methods. As discussed in Section 2, since unstructured pruning methods are difficult to realize its efficiency, we do not make a comparison with them in this experiment.

**Comparison with Structured Pruning.** First, we compare SVD-LLM with three state-of-the-art structured pruning-based LLM compression methods: LLM-Pruner (Ma et al., 2023), SliceGPT (Ashkboos et al., 2024), and BlockPruner (Zhong et al., 2024) under the same compressed weight memory, ranging from 10GB to 7GB. The results on LLaMA-7B are shown in Table 7.




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

Figure 5: Perplexity of LLaMA-7B under 20% compression ratio using calibration data sampled with different number or seeds from WikiText-2.

Table 5: Perplexity of LLaMA-7B compressed by SVD-LLM under 20% to 80% compression ratio on WikiText-2 with different updating order.

| UPDATING ORDER | 20% | 40% | 60% | 80% |
|---|---|---|---|---|
| $V$ first, then $U$ | 6.85 | 8.32 | **13.20** (↓1%) | **31.67**(↓1%) |
| $U$ first, then $V$ | **6.73** (↓2%) | **8.18** (↓2%) | 13.31 | 31.79 |

Table 6: Performance of LLaMA-7B compressed by SVD-LLM under 20% compression ratio using calibration data sampled from WikiText-2 (by default in our paper) and C4 datasets. The performance on WikiText-2 and C4 are reported by perplexity (↓), while the performance on six downstream datasets are reported by average accuracy (↑). The performance on TruthfulQA and GSM8K are reported by BLEU score(↑) and Exact Match Accuracy (↑) respectively. The relative performance gain for data sampled from one dataset compared to another is marked in green color inside bracket.

| WikiText-2↓ | C4↓ | Average↑ | TruthfulQA↑ | GSM8K↑ |
|---|---|---|---|---|
| Calibration data sampled from WikiText-2 | | | | |
| **6.73** (↓1%) | 9.81 | **0.55** (↑2%) | 0.28 | 0.08 |
| Calibration data sampled from C4 | | | | |
| 6.79 | **9.62** (↓2%) | 0.54 | 0.28 | 0.08 |

Table 7: Perplexity (↓) of LLaMA-7B compressed by structured pruning methods and SVD-LLM under various weight memory budget on WikiText-2. The relative performance gain compared to the best-performing baseline is marked in green.

| | PERPLEXITY UNDER WEIGHT MEMORY BUDGET | | | |
|---|---|---|---|---|
| METHOD | 10 GB | 9 GB | 8 GB | 7 GB |
| LLM-Pruner | 9.88 | 12.21 | 18.94 | 21.68 |
| SliceGPT | 8.78 | 12.73 | 16.39 | 27.41 |
| BlockPruner | 9.4 | 12.76 | 19.78 | 43.05 |
| SVD-LLM | **6.92** (↓26%) | **7.38** (↓40%) | **8.02** (↓51%) | **9.23** (↓57%) |

Table 8: Perplexity (↓) of LLaMA-7B compressed by 1-bit quantization methods and SVD-LLM on WikiText-2. The relative performance gain compared to the best-performing baseline is marked in green.

| METHOD | TYPE | MEMORY | PERPLEXITY |
|---|---|---|---|
| PB-LLM | Post-training | 1.9 GB | 104.83 |
| BiLLM | Post-training | 1.5 GB | 47.67 |
| SVD-LLM | Post-training | 1.5 GB | **47.21** (↓1%) |
| OneBit | Training-required | 1.3 GB | 10.20 |
| SVD-LLM + QuIP# | Post-training | 1.3 GB | **9.83** (↓4%) |

SVD-LLM outperforms all state-of-the-art structured pruning-based LLM compression methods. In particular, SVD-LLM achieves up to 57% reduction in perplexity under the 7G memory budget.

**Comparison with Quantization.** Finally, we compare SVD-LLM with three state-of-the-art quantization-based LLM compression methods that push the frontier to 1-bit quantization: BiLLM (Huang et al., 2024), PB-LLM (Yuan et al., 2024), and OneBit (Xu et al., 2024). Both BiLLM and PB-LLM are post-training methods, and OneBit is training-required. The results on LLaMA-7B are shown in Table 8: We have three observations: (1) Among all the post-training methods, SVD-LLM achieves the best performance compared to PB-LLM and BiLLM. (2) Training-required method OneBit outperforms SVD-LLM. This result is expected. This is because OneBit belongs to training-required method, which involves retraining using the large-scale dataset with intensive computing resources to boost performance after compression. However, compared to post-training methods such as SVD-LLM which does not require retraining, training-required method is way too expensive. (3) Lastly, by combining SVD-LLM with a 2-bit post-training quantization-based LLM compression method QuIP# (Tseng et al., 2024)[3], we can outperform training-required method OneBit without expensive retraining. This result is important, because it introduces a highly efficient post-training approach that outperforms state-of-the-art 1-bit training-required quantization-based LLM compression method without incurring expensive retraining.

## 5 CONCLUSION

In this paper, we presented SVD-LLM, a SVD-based post-training 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 parameter update with sequential low-rank approximation to compensate for accuracy degradation. We evaluated SVD-LLM on 10 datasets and seven models from three LLM families at three scales. Our results demonstrate the superiority of SVD-LLM over state-of-the-arts, especially at high model compression ratios.

[3]We first decompose all the weight matrices $W$ in the LLM to the two low-ranking matrices $W_u$ and $W_v$ with SVD-LLM under 40% compression ratio and quantize $W_v$ and then $W_u$ with QuIP#.

## REFERENCES


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.

Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. Fluctuation-based adaptive structured pruning for large language models, 2023.

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari Do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns. In *ICLR*. OpenReview.net, 2024.

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.

Patrick H. Chen, Hsiang-Fu Yu, Inderjit S. Dhillon, and Cho-Jui Hsieh. DRONE: data-aware low-rank compression for large NLP models. In *NeurIPS*, pages 29321–29334, 2021.

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/`.

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.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024.

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.

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`.

Shangqian Gao, Ting Hua, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. Adaptive rank selections for low-rank approximation of language models. In *NAACL-HLT*, pages 227–241. Association for Computational Linguistics, 2024.

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.

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

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.

Wei Huang, Yangdong Liu, Haotong Qin, Ying Li, Shiming Zhang, Xianglong Liu, Michele Magno, and Xiaojuan Qi. Billm: Pushing the limit of post-training quantization for llms. In *ICML*. OpenReview.net, 2024.

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.

Yixiao Li, Yifan Yu, Qingru Zhang, Chen Liang, Pengcheng He, Weizhu Chen, and Tuo Zhao. Losparse: Structured compression of large language models based on low-rank and sparse approximation. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 20336–20350. PMLR, 2023.

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. In *ACL (1)*, pages 3214–3252. Association for Computational Linguistics, 2022.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4A8KV4 quantization and system co-design for efficient LLM serving. *CoRR*, abs/2405.04532, 2024.

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

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

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

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.

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.

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.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

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.

Jonathan Tow, Marco Bellagente, Dakota Mahan, and Carlos Riquelme. Stablelm 3b 4e1t. URL [https://huggingface.co/stabilityai/stablelm-3b-4e1t](https://huggingface.co/stabilityai/stablelm-3b-4e1t).

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better LLM quantization with hadamard incoherence and lattice codebooks. In *ICML*. OpenReview.net, 2024.

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

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.

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.

Yuzhuang Xu, Xu Han, Zonghan Yang, Shuo Wang, Qingfu Zhu, Zhiyuan Liu, Weidong Liu, and Wanxiang Che. Onebit: Towards extremely low-bit large language models. *CoRR*, abs/2402.11295, 2024.

Hao Yu and Jianxin Wu. Compressing transformers: Features are low-rank, but weights are not! In *AAAI*, pages 11007–11015. AAAI Press, 2023.

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.

Zhihang Yuan, Yuzhang Shang, and Zhen Dong. PB-LLM: partially binarized large language models. In *ICLR*. OpenReview.net, 2024.

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.

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.

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.

Longguang Zhong, Fanqi Wan, Ruijun Chen, Xiaojun Quan, and Liangzhi Li. Blockpruner: Fine-grained pruning for large language models. *CoRR*, abs/2406.10594, 2024.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, Shengen Yan, Guohao Dai, Xiao-Ping Zhang, Yuhan Dong, and Yu Wang. A survey on efficient inference for large language models, 2024.

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 LOSS 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 $\sigma'_i$. 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 THE COMPRESSION LOSS OF SVD-LLM

In SVD-LLM, we also formulate the linear layer as $Y = (WS)S^{-1}X$, where $S^{-1}XX^T\left(S^{-1}\right)^T = I$. The compression is made by keeping the largest $m$ out of total $r$ singular values of $WS$. The compression loss $L$ is demonstrated as:

$$
\begin{aligned}
L^2 &= \|WX - W'X\|_F^2 = \left\|WSS^{-1}X - SVD(WS)S^{-1}X\right\|_F^2 \\
&= \left\|(WS - SVD(WS))S^{-1}X\right\|_F^2 \\
&= \left\|\left(WS - \sum_{i=1}^{m}\sigma_i u_i v_i^T\right)S^{-1}X\right\|_F^2 \\
&= \left\|\sum_{i=m+1}^{r}\sigma_i u_i v_i^T S^{-1}X\right\|_F^2 \\
&= \sum_{j=m+1}^{r}\sum_{i=m+1}^{r}\sigma_i\sigma_j \operatorname{trace}\left(u_i v_i^T S^{-1}XX^T\left(S^{-1}\right)^T v_j u_j^T\right) \\
&= \sum_{j=m+1}^{r}\sum_{i=m+1}^{r}\sigma_i\sigma_j \operatorname{trace}\left(u_i v_i^T\left(S^{-1}XX^T\left(S^{-1}\right)^T\right)v_j u_j^T\right) \\
&= \sum_{j=m+1}^{r}\sum_{i=m+1}^{r}\sigma_i\sigma_j \operatorname{trace}\left(u_i v_i^T v_j u_j^T\right)
\end{aligned}
$$

$$
\because v_i^T v_i = u_i^T u_i = 1; v_i^T v_j = u_i^T u_j = 0, \operatorname{trace}\left(v_i v_i^T\right) = \operatorname{trace}\left(u_i u_i^T\right) = 1, \forall i \neq j
$$

$$
\therefore L^2 = \sum_{j=m+1}^{r}\sum_{i=m+1}^{r}\sigma_i\sigma_j \operatorname{trace}\left(u_i v_i^T v_j u_j^T\right) = \sum_{i=m+1}^{r}\sigma_i^2 \operatorname{trace}\left(u_i v_i^T v_i u_i^T\right) = \sum_{i=m+1}^{r}\sigma_i^2
$$

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

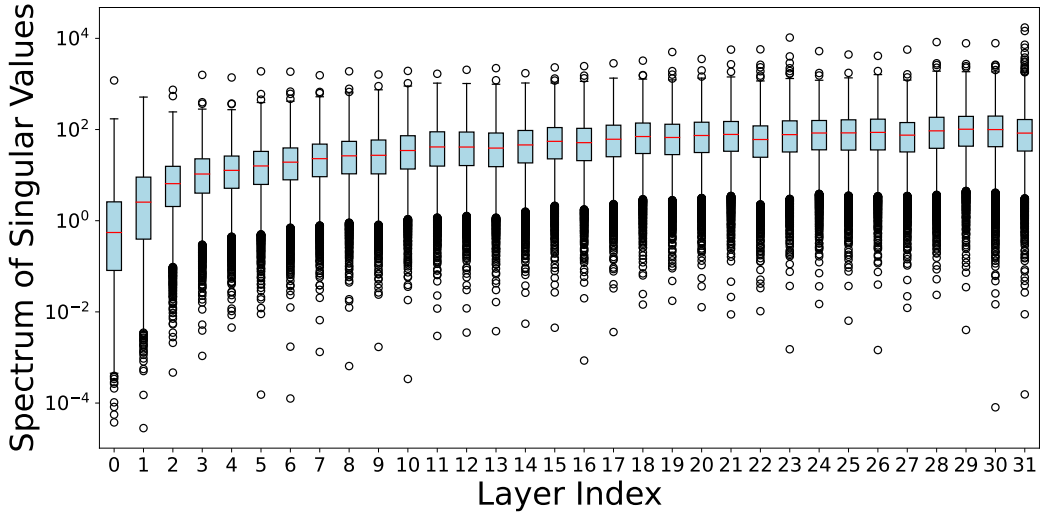
### A.3 More Experimental Setups

**Implementation Details of the experiments.** To ensure a fair comparison, we followed ASVD (Yuan et al., 2023) to randomly select 256 samples from WikiText-2 as the calibration data. We followed the same configuration used in LLM-Pruner (Ma et al., 2023) to use Alpaca (Taori et al., 2023) dataset with 50K samples for parameter update in `SVD-LLM`. The inference efficiency experiment is conducted on both NVIDIA A100 GPU and AMD EPYC 7643 CPU while the other experiments are conducted on NVIDIA A100 GPUs.

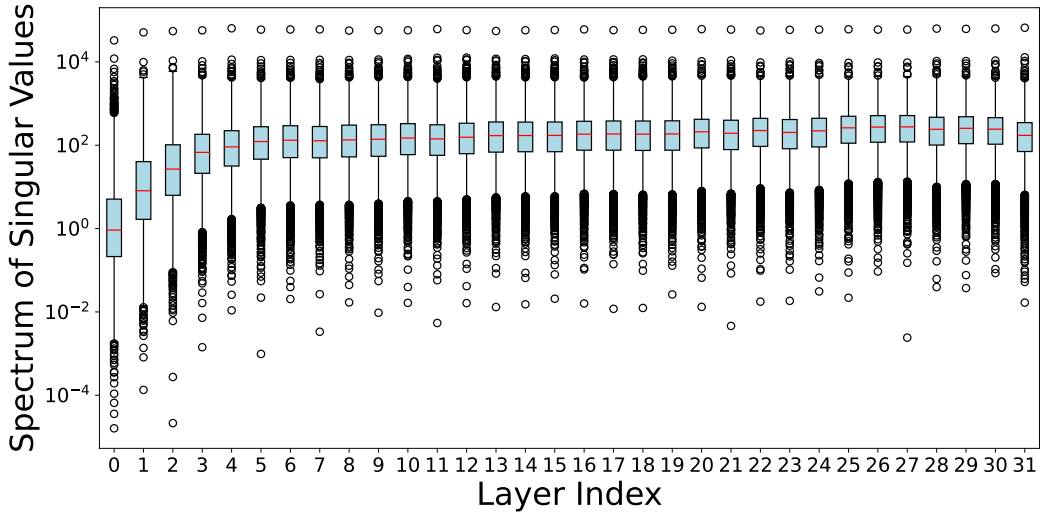
**Models and Datasets.** To demonstrate the generability of our method, we evaluate the performance of `SVD-LLM` on seven models from three different LLM families at three different scales (LLaMA-7B, 13B, 30B, LLaMA2-7B (Touvron et al., 2023), OPT-6.7B (Zhang et al., 2022), Vicuna-7B (Chiang et al., 2023) and Mistral-7B (Jiang et al., 2023)) and 10 datasets including two language modeling datasets (WikiText-2 (Merity et al., 2017), and C4 (Raffel et al., 2020)), six classification datasets (OpenbookQA (Mihaylov et al., 2018), WinoGrande (Sakaguchi et al., 2020), HellaSwag (Zellers et al., 2019), Arc_e (Clark et al., 2018), PIQA (Bisk et al., 2020), MathQA (Amini et al., 2019)), and two generation datasets (TruthfulQA (Lin et al., 2022), and GSM8K (Cobbe et al., 2021) with the LM-Evaluation-Harness framework (Gao et al., 2023).

### A.4 Spectrum Analysis of Singular Values decomposed by `SVD-LLM`

In general, SVD for compression is useful when the matrix to be compressed shows a sharp decay of the singular values. Since SVD-LLM decomposes the multiplication of the weight matrix $W$ and its corresponding whitening matrix $S$ instead of the original weight matrix $W$, which is different from the weight decomposition in the previous work (Yuan et al., 2023; Hsu et al., 2022), to study whether SVD compression is also applicable in `SVD-LLM`, we select the Query ($W_Q$) and Key ($W_K$) weight




(a) $W_Q \times S_Q$




(b) $W_K \times S_K$

Figure 6: The singular value spectrum of the decomposed matrices across layers in LLaMA-7B.

matrices and show the spectrum of the singular values of their multiplication with corresponding whitening matrices $S_Q$ and $S_K$. As shown in Figure 6, most of the single values are less than 100 with only a few extremely large values, indicating that SVD compression is applicable in SVD-LLM.

### A.5 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 20% compression ratio on an A100 GPU. The results are shown in Table 9. As shown,

Table 9: Compression time of SVD-LLM and ASVD on LLaMA-7B under 20% compression ratio. The relative speedup is marked in green color inside bracket.

| SVD-LLM | | | ASVD | | |
|---|---|---|---|---|---|
| Truncation-Aware Data Whitening | Parameter Update with Sequential Low-rank Approximation | **Total** | Normalize | Search | **Total** |
| 10min | 3.5h | **3.5h** (↓36%) | 5min | 5.5h | **5.5h** |

ASVD takes about 5.5 hours whereas SVD-LLM completes the compression process in 3.5 hours, which is 36% times faster. When breaking down the time, most of the time consumed by ASVD is dedicated to searching for the specific compression ratio for each weight matrix based on its calculated importance score. In contrast, SVD-LLM maintains a consistent compression ratio across all weight matrices and thus gets rid of the time-consuming search process.

### A.6 Contents Generated from the LLM Compressed by SVD-LLM and ASVD

Some examples of sentences generated by LLaMA-7B compressed with SVD-LLM and ASVD are shown in Table 10. The sentences generated by the model compressed by SVD-LLM exhibit better fluency, relevance, and informativeness compared to that compressed by ASVD. More importantly, when the compression ratio is increased to 40%, the previous state-of-the-art method ASVD completely loses its generation ability. In contrast, even when the compression ratio is up to 80%, SVD-LLM is still capable of generating complete sentences.

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

| Ratio | SVD-LLM | ASVD |
|---|---|---|
| Original | **What is the responsibility of the AI assistant?** The responsibility of an AI assistant is to complete the job without any hesitation. Also, the AI assistant has no problem with taking a decision and executing it. An AI assistant is responsible to complete the task and it needs to learn from his past experience to take the best possible decision without any hesitation... | |
| 20% | **What is the responsibility of the AI assistant?** The AI assistant's role and responsibilities are often ambiguous and can vary depending on the context and the type of assistant being used. Some AI assistants may have preset responses to common queries, but their behavior can also depend on the user's interaction and expectations... | **What is the responsibility of the AI assistant?** 1. To act in the same way as the programmer intended. 2. To make a value assessment of the user's intentions and offer an appropriate response. 3. To help the user carry out specific tasks or follow a strict script. 4. To make the system as independent of the user as possible... |
| 40% | **What is the responsibility of the AI assistant?** 1. The AI assistant is responsible for performing tasks that can only be done by AI technology such as scheduling appointments, setting reminders and making suggestions, while also providing quick and accurate responses. 2. The AI assistant is also responsible for learning and updating its knowledge base through continuous improvement, thus making it more efficient and effective. 3. The AI assistant is responsible for handling and managing user-generated and customer-oriented input... | **What is the responsibility of the AI assistant?** 1. Define the definition of the job specification. 2. Define the role of the AI assistant for the job. 3. Define the skills necessary in order to carryout the tasks assigned to the job to do the work of the same capacity in the market. 4. Determine the evaluation of the job competence of the AI assistant to ensure their professional integrity. 5. What type of AI assistant jobs are available? 6. What industries require technology assistance work to complete the tasks that it own personnel cannot perform... |
| 60% | **What is the responsibility of the AI assistant?** In a nutshell, the responsibility of the AI assistant could vary depending on the task, but generally, the focus would be on automatic tasks, without the need for human intervention. Some common tasks could include setting reminders, scheduling appointments, and making routine household or productivity tasks. The AI assistant also serves as a backup or a relief system, taking on responsibilities when the user is not available ... | **What is the responsibility of the AI assistant?** 2.3. ?? Brush for a discussion I wonder is it worth doing is important.2- It isn't useful just reducing labor costs; it helps employees feel a sense of connected to your attention which leads to better workplace values among staffers and leads to long relationships among org... |
| 80% | **What is the responsibility of the AI assistant?** Our Design is based on our understanding of the world, and we are actively learning, adapting and adapting, so we're always evolving new ideas, which we see to be most unique and relevant in our community... | **What is the responsibility of the AI assistant?** ygua AIeltemperaturen/2, (64mbz/.3/.1/, 7.kbld.org.0/2/ In these puthebout les bnvols n merginels ... |

### A.7 Comparison with smaller LLMs pre-trained from scratch

To compare the performance between SVD-LLM and scratch training, following the previous experimental design (Ma et al., 2023), we compress LLaMA-7B to the size of the 3B parameter with SVD-LLM and select StableLM-3B (Tow et al.) as the baseline for comparison. The result is shown in Table 11. LLaMA-3B compressed from LLaMA-7B by SVD-LLM achieves better accuracy in all datasets, indicating that SVD-LLM could even achieve better accuracy than some scratch training methods. Furthermore, SVD-LLM ensures higher throughput and lower memory consumption than StableLM-3B as shown in the table, which also meets other efficiency analysis and discussion in Section 4.2.

Table 11: Comparison of LLaMA-3B (compressed from LLaMA-7B by SVD-LLM) and original StableLM-3B (Tow et al.) trained from scratch. Both the throughput and the peak memory footprint during the inference are measured with batch size=32, sequence length = 128 on single A100 GPU.

| MODEL | Throughput | Peak Mem. | Openb. | Arc_e | WinoG. | HellaS. | PIQA | MathQA | **Average**↑ | TruthfulQA↑ | GSM8K↑ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| StableLM-3B | 8463 Tokens/sec | 9.41 GB | 0.19 | 0.51 | 0.55 | 0.37 | 0.69 | 0.21 | 0.44 | 0.22 | 0.02 |
| LLaMA-3B | 9254 Tokens/sec | 7.43 GB | **0.27** | **0.54** | **0.58** | **0.46** | 0.68 | **0.23** | **0.46** (↑5%) | **0.23** (+ 0.01) | **0.04** (+ 0.02) |

**Algorithm 1** Pseudocode of 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$(M, C)$
  $\text{Set}_W \leftarrow M$ ▷ Obtain the set of weights in $M$ to compress
  **for** $W$ **in** $\text{Set}_W$ **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 $\Sigma$
    $W_u \leftarrow U(\Sigma_1)^{1/2}, W_v \leftarrow (\Sigma_1)^{1/2}V^T S^{-1}$ ▷ Obtain two low-rank matrices
    $M'(W) \leftarrow W_u, W_v$ ▷ Replace $W$ with $W_u$ and $W_v$ in $L$
  **end for**
  $M'' \leftarrow$ PARAMETER UPDATE WITH SEQUENTIAL LOW-RANK APPROXIMATION$(M')$
  **return** $M''$
**end procedure**

### A.8 PSEUDOCODE OF 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 $\text{Set}_S$ for the weight to compress. After that, it runs the SVD and truncation with $\text{Set}_S$ 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 parameter update with sequential low-rank approximation as shown in Algorithm 3.

### A.9 COMPARISON ON EXTREMELY LARGE-SCALE AND MORE RECENT LLMS

To further show the generalization of SVD-LLM, we compare its performance with other SVD-based baselines on extremely large-scale and more recent LLMs. Below shows the perplexity of SVD-LLM and other baselines on WikiText-2 when compressing LLaMA-2 70B (Touvron et al., 2023) and LLaMA-3 8B (Dubey et al., 2024) under 20% compression ratio. For LLaMA-2 70B, we only apply truncation-aware data whitening of SVD-LLM without parameter update with sequential low-rank approximation due to the limited computational budget. OOM means that running the algorithm causes out-of-memory on 4 A100 GPUs. As shown, SVD-LLM still consistently outperforms other baselines when applied on these two LLMs.

Table 12: Perplexity (↓) on WikiText-2 of LLaMA 2-70B and LLaMA 3-8B under 20% compression ratio.

| MODEL | LLAMA 2-70B | LLAMA 3-8B |
|---|---|---|
| Original | 3.32 | 6.14 |
| SVD | 19.82 | 29871 |
| FWSVD | OOM | 4782 |
| ASVD | OOM | 17.55 |
| SVD-LLM | **4.21** (↓66%) | **8.16** (↓54%) |

### A.10 COMPARISON WITH DRONE

Previous work Drone (Chen et al., 2021) focusing on compressing the Bert model also proposes their data-aware method for SVD compression. They even provide a theoretical analysis to prove the

**Algorithm 2** Pseudocode of Truncation-Aware Data Whitening

**Input:** $M$: Original LLM
**Input:** $C$: Calibration Data
**Output:** $\text{Set}_S$: 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** $\text{Set}_W$ **do**
        $X \leftarrow M(W, C)$ ▷ 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** $\text{Set}_S$
**end procedure**

**Algorithm 3** Pseudocode of Parameter Update with Sequential Low-rank Approximation

**Input:** $M'$: Compressed LLM by Truncation-aware Data Whitening
**Output:** $M''$: Compressed LLM with Parameter Update with Sequential Low-rank Approximation
**procedure** PARAMETER UPDATE WITH SEQUENTIAL LOW-RANK APPROXIMATION($M'$)
    $M'_u \leftarrow \text{LoRA}_\text{u}(M')$ ▷ Fix all $W_v$, fine-tune all $W_u$
    $M'' \leftarrow \text{LoRA}_\text{v}(M'_u)$ ▷ Fix all $W_u$, fine-tune all $W_v$
    **return** $M''$
**end procedure**

optimal solution that their method achieve.s Specifically, Drone represents the low-rank compressed weight matrix $W'$ by $WM$. It performs SVD on both weight matrix $W = U_w S_w V_w^T$ and the transpose of input activation $X^T = U_x S_x V_x^T$ and then split these decomposed matrices as follows:

$$U_W = \begin{bmatrix} U_{W,r} & \bar{U}_{W,r} \end{bmatrix}, S_W = \begin{bmatrix} S_{W,r} & 0 \\ 0 & 0 \end{bmatrix}, V_W = \begin{bmatrix} V_{W,r} & \bar{V}_{W,r} \end{bmatrix}$$
$$U_X = \begin{bmatrix} U_{X,t} & \bar{U}_{X,t} \end{bmatrix}, S_X = \begin{bmatrix} S_{X,t} & 0 \\ 0 & 0 \end{bmatrix}, V_X = \begin{bmatrix} V_{X,t} & \bar{V}_{X,t} \end{bmatrix}.$$

where $r$ and $k$ are the rank of the $W$ and $X$. $U_{W,r}, V_{W,r}, U_{X,t}, V_{X,t}$ denote corresponding row spaces and column spaces and $\bar{U}_{W,r}, \bar{V}_{W,r}, \bar{U}_{X,t}, \bar{V}_{X,t}$ are null spaces. Through theoretical deduction, Drone converts the minimization of compression loss $||WX - W'X||_F = ||WX - WMX||_F$ to the minimization of $\left\|S_{W,r}V_{W,r}^T V_{X,t}S_{X,t} - S_{W,r}V_{W,r}^T M V_{X,t}S_{X,t}\right\|_F$, whose optimal value $L_{min}$ is the rank-k truncated SVD of $Z = S_{W,r}V_{W,r}^T V_{X,t}S_{X,t}$ by the fundamental property of SVD decomposition. To achieve the optimal value, Drone formulates a solution $M = V_{W,r}S_{W,r}^{-1} Z_k S_{X,t}^{-1} V_{X,t}^T$, where $Z_k$ is the rank-k SVD truncation of $Z$.

In short, compared with Drone, SVD-LLM is also optimal with the same theoretical compression loss as Drone. Moreover, SVD-LLM has **three** key advantages.

**SVD-LLM is also optimal with the same theoretical compression loss as Drone.** Although the motivation of SVD-LLM originates from the LLM-based SVD compression method, especially ASVD, and its motivation is to align the SVD truncation with the truncated singular values for a correct truncation, as discussed in Section 3.1, our theoretical analysis shows that SVD-LLM is also optimal with the same compression loss as Drone. Specifically, the theoretical minimum compression loss $L_{min}$ is the F-norm loss of rank-k SVD truncation of $WX$, which has also been achieved by Drone in their paper. Unlike Drone, SVD-LLM constructs the whitening matrix $S$ so that $S^{-1}X$ is orthonormal. Therefore, we have $||AS^{-1}X||_F = ||A||_F$. Suppose that we decompose $S$ with SVD to $U_s, S_s, V_s$, we can have $S_s = S_x, U_s = U_x, Us = Ux, V_s = QV_x$, where $Q$ is an orthogonal matrix. The matrix $WS$ to which SVD-LLM applies SVD could be represented by

$U_w S_w V_w^T U_s S_s V_s^T$. Suppose that we use $Trunc.(C)$ to represent the rank-k truncation of the matrix $C$ during SVD compression, the compression loss $L$ is derived as follows:

$$
\begin{aligned}
L &= ||WX - W'X||_F = ||(WSS^{-1}X - SVD(WS)S^{-1}X)||_F = ||(WS - SVD(WS))S^{-1}X||_F \\
&= ||Trunc.(WS)S^{-1}X||_F = ||Trunc.(WS)||_F \\
&= ||Trunc.(U_w S_w V_w^T U_s S_s V_s^T)||_F \\
&= ||Trunc.(WXQ^T)||_F = L_{min}
\end{aligned}
$$

Therefore, SVD-LLM shares the same theoretical compression loss as Drone.

**Advantage #1: SVD-LLM has better numerical stability, which leads to superior empirical compression loss.** While SVD-LLM shares the same theoretical compression loss as Drone, Drone's higher complexity—stemming from additional SVD operations and inverse calculations on large-scale matrices—makes it less numerically stable compared to SVD-LLM. This often results in higher empirical compression losses in practice. To illustrate this, we compare SVD-LLM and Drone in terms of the empirical compression losses for randomly generated matrices of various shapes. We also include the theoretical minimum value, represented by the rank-k Frobenius norm loss of $WX$. The results are summarized in the following table. As shown, we observe that SVD-LLM achieves lower empirical compression losses than Drone, underscoring its superior numerical stability.

Table 13: Compression loss of the randomly generated weight and activation matrices with different shapes under 50% compression ratio using SVD-LLM, Drone, and the theoretical minimum.

| Loss | $[128 \times 128] \times [128 \times 128]$ | $[2048 \times 2048] \times [2048 \times 2048]$ | $[4096 \times 4096] \times [4096 \times 4096]$ |
|---|---|---|---|
| Minimum | 276.1130 | 17784.2637 | 50321.9141 |
| Drone | 276.1130 | 17785.6992 | 50337.2148 |
| SVD-LLM | 276.1130 | 17784.2676 | 50321.9727 |

**Advantage #2: In practice, Drone incurs out-of-memory when compressing LLMs due to its requirement of storing the full large-scale activations, whereas SVD-LLM is feasible.** To achieve data-awareness during compression, Drone caches all input activations $X$ and spans them to calculate the corresponding singular vectors and singular values. In the Drone paper, the authors apply Drone to BERT. However, the activations generated by LLMs are often extremely large and are much larger than BERT. For example, using Drone, caching 16 input activations produced by LLaMA-7B in FP32 format, as required for SVD computation, already exceeds the memory capacity of an A100 GPU with 80GB memory. In contrast, SVD-LLM incrementally updates its $XX^T$ matrix by adding the $xx^T$ of each new input x. As such, SVD-LLM eliminates the need to store the full activations, requiring only the storage of the $XX^T$ matrix, which is considerably smaller than even a single input activation. Due to this advantage, SVD-LLM is far more practical to compress LLMs of size 7B or larger compared to Drone.

**Advantage #3: SVD-LLM incurs much shorter compression time compared to Drone.** Drone involves more complex matrix operations, leading to longer compression time compared to SVD-LLM. To illustrate this, we measured the time required by Drone and SVD-LLM to compress randomly generated weight and activation matrices of varying shapes under a 50% compression ratio. The results show that SVD-LLM is approximately three times faster than Drone.

Table 14: Compression Time of the randomly generated weight and activation matrices with different shapes using SVD-LLM and Drone. The compression time is measured for 5 times' compression.

| Time | $[128 \times 128] \times [128 \times 128]$ | $[2048 \times 2048] \times [2048 \times 2048]$ | $[4096 \times 4096] \times [4096 \times 4096]$ |
|---|---|---|---|
| Drone | 0.07 seconds | 5.81 seconds | 30.35 seconds |
| SVD-LLM | 0.02 seconds | 1.98 seconds | 10.37 seconds |

### A.11 Comparison with FLAP

Recent work FLAP (An et al., 2023) is also a post-training structured-pruning method. Below we compare the perplexity of SVD-LLM and FLAP on WikiText-2 under different compression ratios when compressing LLaMA-7B. As shown in Table 15, SVD-LLM consistently outperforms FLAP, especially under high compression ratios.

Table 15: Perplexity (↓) of SVD-LLM and FLAP on WikiText-2 to compress LLaMA-7B under different compression ratios. The better performance is marked in bold. The relative performance gain of SVD-LLM compared to FLAP is marked in green inside bracket.

| RATIO (MEM.) | 20% (10.2GB) | 40% (7.76GB) | 60% (5.35GB) | 80% (2.58GB) |
|---|---|---|---|---|
| FLAP | 7.99 | 14.43 | 106.87 | 15023 |
| SVD-LLM | **6.73** (↓16%) | **8.18** (↓43%) | **13.31** (↓88%) | **31.79** (↓99%) |

Table 16: Perplexity (↓) of SVD-LLM with original LoRA fine-tuning (denoted as SVD-LLM (SFT)), ASVD with sequential LoRA fine-tuning (denoted as ASVD (SFT)), and SVD-LLM with sequential LoRA fine-tuning (denoted as SVD-LLM (SFT)) on WikiText-2 to compress LLaMA-7B under different compression ratios.

| RATIO (MEM.) | 20% (10.2GB) | 40% (7.76GB) | 60% (5.35GB) | 80% (2.58GB) |
|---|---|---|---|---|
| SVD-LLM (NFT) | 7.25 | 11.98 | 16.30 | 80.23 |
| ASVD (SFT) | 8.37 | 14.86 | 44.81 | 271 |
| SVD-LLM (SFT) | **6.73** | **8.18** | **13.31** | **31.79** |

### A.12 MORE ABLATION STUDIES

**SVD-LLM + Normal LoRA Fine-tuning v.s. SVD-LLM + Sequential LoRA Fine-tuning.** To illustrate the superiority of the designed parameter update with sequential low-rank approximation in SVD-LLM, which is a kind of sequential LoRA fine-tuning strategy over the normal LoRA fine-tuning strategy, we compare the compression performance of SVD-LLM by applying either of these two fine-tuning strategies. Let's denote SVD-LLM (SFT) as SVD-LLM by applying sequential LoRA fine-tuning and SVD-LLM (NFT) as SVD-LLM by applying normal LoRA fine-tuning. As shown in Table 16, SVD-LLM (SFT) consistently outperforms SVD-LLM (NFT), which also reaffirms our analysis in Section 3.2 that optimizing both of the low-rank matrices $W_u, W_v$ at the same time is not stable and may lead to poor fine-tuning performance.

**ASVD + Sequential LoRA Fine-tuning v.s. SVD-LLM + Sequential LoRA Fine-tuning.** Although the designed sequential LoRA fine-tuning strategy could also be applied in other SVD-based LLM compression methods, other methods' performance is still poorer than SVD-LLM even being integrated with this strategy for enhancement. To illustrate this, we compare the performance of previous state-of-the-art method ASVD when be applied with the sequential LoRA finetuning with SVD-LLM. Let's denote SVD-LLM (SFT) as SVD-LLM by applying sequential LoRA fine-tuning and ASVD (SFT) as ASVD by applying sequential LoRA fine-tuning. As shown in Table 16, SVD-LLM (SFT) consistently outperforms ASVD (SFT) under various compression ratios.

**Orthogonality of whitening matrix on inference activations.** Below we randomly select three weight matrices $W_1, W_2, W_3$ in LLaMA-7B and compute their on-the-fly whitening matrix $S_{fly}$ computed during inference. To test whether $S_{pre}$ precomputed on the calibration data can also make the inference activation orthogonal like $S_{fly}$, we measure the difference between $S_{fly}S_{fly}^T$ and $S_{pre}S_{pre}^T$ in the MSE format, e.i., $\frac{1}{N}||S_{fly}S_{fly}^T - S_{pre}S_{pre}^T||_F^2$, where $N$ is the number of elements in $S_{pre}S_{pre}^T$. As shown in Table 17, the difference is small, indicating that the pre-computed whitening matrix $S_{pre}$ is still effective in making the inference activation orthogonal to ensure the alignment between SVD truncation and compression loss.

Table 17: MSE between $S_{pre}S_{pre}^T$ pre-computed on WikiText-2 calibration data and $S_{fly}S_{fly}^T$ computed on-the-fly on three randomly selected weight matrices in LLaMA-7B.

| DATA | $W_1$ | $W_2$ | $W_3$ |
|---|---|---|---|
| MSE | 0.003 | 0.0011 | 0.0009 |