Adaptive Nonlinear Compression for Large Foundation Models

Liang Xu[1,2], **Shufan Shen**[1,2], **Qingming Huang**[1,2,3], **Yao Zhu**[4], **Xiangyang Ji**[5], **Shuhui Wang**[1,3]*
[1] State Key Lab. of AI Safety, Institute of Computing Technology, Chinese Academy of Sciences
[2] University of Chinese Academy of Sciences [3] Peng Cheng Laboratory
[4] Zhejiang University [5] Tsinghua University
{xuliang23z, shenshufan22z, wangshuhui}@ict.ac.cn
qmhuang@ucas.ac.cn ee_zhuy@zju.edu.cn xyji@tsinghua.edu.cn

Abstract

Despite achieving superior performance, large foundation models (LFMs) have substantial memory requirements, leading to a growing demand for model compression methods. While low-rank approximation presents a promising hardware-friendly solution, existing linear methods suffer significant information losses due to rank truncation. Nonlinear kernels can enhance expressiveness by operating in higher-dimensional spaces, yet most kernels introduce prohibitive overhead and struggle to support adaptive rank allocation across heterogeneous matrices. In this paper, we propose a compression method called Nonlinear Low-Rank Approximation with Adaptive Budget Allocation (NLA). Instead of relying on linear products, we employ piecewise-linear kernels with a forward-pass optimization operator to approximate weight matrices, enhancing the recovery of high-rank weight matrices from low-rank matrices. Moreover, considering the heterogeneous representation abilities and dynamic sensitivities of different weight matrices, we adaptively allocate the compression ratio of each weight matrix during the re-training process by cubic sparsity scheduling. Through evaluations on large language models and vision models across various datasets, NLA demonstrates superior performance while achieving a higher compression ratio compared to existing methods. Our codes will be released in `https://github.com/Liang08/NLA`.

1 Introduction

Large Foundation Models (LFMs) (Touvron et al., 2023a;b; OpenAI, 2023) have demonstrated remarkable performance across various tasks. Despite their impressive capabilities, LFMs possess an excessive number of pre-trained parameters that require high memory usage for inference. This substantial memory usage severely limits the deployment of such models in resource-constrained scenarios. To this end, researchers have proposed various methods to reduce the parameter volume of LFMs, which can be categorized into distillation-based and compression-based strategies. The distillation-based strategy (Magister et al., 2023) trains a smaller model to emulate the behavior of the original model, thereby reducing resource requirements by only deploying this small model. However, retraining such a small model

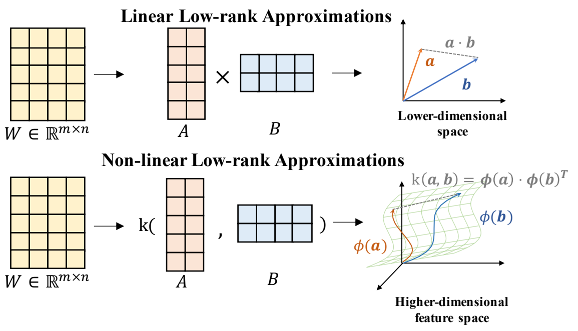
Linear Low-rank Approximations
$W \in \mathbb{R}^{m\times n}$
A
B
$a \cdot b$
a
b
Lower-dimensional space
Non-linear Low-rank Approximations
k(
$\mathrm{k}(a,b)=\phi(a)\cdot\phi(b)^T$
$\phi(b)$
$\phi(a)$
Higher-dimensional feature space

Figure 1: Existing low-rank compression methods (*top*) rely on linear approximations, resulting in significant information loss. In contrast, NLA approximates the weight matrix non-linearly (*bottom*), achieving less information loss with the same number of parameters.

*Corresponding author.

from scratch entails substantial data requirements and high computational costs. In contrast, the compression-based strategy directly reduces the scale of the original model in terms of parameter size or memory by pruning weights (Xia et al., 2023) or quantizing (Park et al., 2022) them to a lower precision. Despite reducing the resource requirements with minimal retraining costs, these methods impose higher demands on specialized hardware for model deployment, such as GPUs that support unstructured sparse inference (Han et al., 2016) and low-bit precision computations (Jouppi et al., 2017).

Recently, researchers have proposed low-rank approximation methods (Wang et al., 2024; Yuan et al., 2023; Saha et al., 2023) that compress the model by representing the large pre-trained weight matrix using small low-rank matrices. As these low-rank matrices remain compact while preserving the precision of the original matrix, this line of methods avoids the dependence on specialized hardware. Existing methods typically rely on singular value decomposition (SVD) to perform low-rank approximation. First, they decompose the pre-trained weight matrix into a linear combination of singular vectors weighted by their corresponding singular values. Second, model size is reduced by truncating these singular vectors associated with smaller singular values. Finally, the model performance is restored through lightweight re-training of these resulting truncated matrices.

However, the truncation of singular vectors inevitably discards many singular directions. As the rank decreases, more informative components are removed from the approximated matrix, leading to severe information loss and performance degradation under high compression ratios. An effective way to mitigate this issue is to replace the linear inner product with nonlinear kernel functions. By implicitly mapping the low-rank matrices into a higher-dimensional Hilbert space, kernel functions allow the approximation to operate in a richer feature space and thus enhance the expressiveness of low-rank matrices. However, directly applying nonlinear functions poses critical challenges: most common kernels require reconstructing the full weight matrix during forward propagation, which significantly increases the computational cost and memory consumption. Moreover, dynamic rank allocation significantly improves performance after compression, as different matrices exhibit heterogeneous sensitivity. Yet, kernel-based formulations make it difficult to directly adopt such adaptive rank adjustment strategies during the re-training process.

In this paper, we propose **N**onlinear **L**ow-rank approximation with **A**daptive budget allocation (NLA), a low-rank approximation method that compresses LFMs non-linearly with an adaptive compression ratio allocation mechanism. To make nonlinear low-rank approximation practical, we design a piecewise-linear forward propagation operator that directly operates on the low-rank factors through self-term and interaction-term decomposition, thus avoiding explicit reconstruction of the full weight matrix. This design preserves the expressive power of nonlinear kernels while significantly reducing their computational and memory overhead. Considering the heterogeneous representation abilities and the dynamic sensitivities of different parameters during retraining, we propose an end-to-end method to adaptively adjust the compression ratio of each weight matrix. By calculating the importance scores of the parameters, we adjust the number of active pieces in the low-rank matrices, and the overall compression ratio is determined by the number of pieces preserved. Across different layers, the allocation is guided by their importance scores, with more sensitive matrices being assigned a larger number of pieces. The budget allocation is controlled by a cubic sparsity scheduling (Sanh et al., 2020) strategy, which starts conservatively to preserve model capacity and becomes more aggressive to achieve higher compression ratios.

In experiments, we validate the effectiveness of NLA on both pre-trained Large Language Models (LLMs) (Touvron et al., 2023a) and vision models (Liu et al., 2021). On LLMs, NLA consistently outperforms state-of-the-art methods (Hsu et al., 2022; Yuan et al., 2023; Wang et al., 2024). For example, on LLaMA-7B (Touvron et al., 2023a), NLA achieves a perplexity of 15.21 on WikiText-2 under a 50% compression ratio, compared with 23.97 for SVD-LLM (Wang et al., 2024). After fine-tuning, NLA further reduces perplexity to 9.68 (13.26 for SVD-LLM). On common sense reasoning benchmarks, NLA achieves a relative improvement of about 12% over the SOTA method (0.37 vs. 0.33). On Swin-Base (Liu et al., 2021), NLA achieves higher accuracy (82.7% vs. 82.4%) while compressing the model to a significantly greater extent (54.2% vs. 29.2%) compared to the current SOTA method. These results demonstrate that NLA achieves higher compression ratios with nearly lossless accuracy, offering a flexible and effective solution for large-scale model compression.

2 Related Work

Model Compression. With the rapid growth in model parameter size, a variety of compression methods have emerged to address the challenges of high memory demands for LFMs (Zhu et al., 2024). These methods can be broadly categorized into four categories: quantization (Park et al., 2022; Yao et al., 2022; Dettmers et al., 2022), pruning (Frantar & Alistarh, 2023; Ma et al., 2023), knowledge distillation (Huang et al., 2022; Jiang et al., 2023), and low-rank approximation (Yuan et al., 2023; Saha et al., 2023). Despite demonstrating effective performance in model compression, existing methods have several limitations. Distillation methods typically require a large amount of data for training, while pruning and quantization methods often impose high hardware requirements. Additionally, these methods face significant performance degradation when applied in extreme compression settings. Among existing studies, low-rank approximation methods stand out for their low dependence on specialized hardware and their ability to combine with other compression methods. Our proposed method also belongs to this category.

Low-rank Approximation. Low-rank approximation, a method that employs small low-rank matrices as trainable parameters in place of large weight matrices, is widely used to reduce memory usage during fine-tuning (Hu et al., 2021; Valipour et al., 2022; Shen et al., 2024). Considering its advantage in memory efficiency, the low-rank approximation has been employed for model compression (Wen et al., 2017; Khodak et al., 2021; Li et al., 2023) to reduce memory usage during inference. PELA (Guo et al., 2024) combines low-rank compression and fine-tuning by applying feature distillation and weight perturbation in compressed models. ASVD (Yuan et al., 2023) scales the weight matrix using a matrix that represents the influence of the input channels. In FWSVD (Hsu et al., 2022), Fisher information is introduced during weighted SVD to measure the importance of parameters to model predictions, ensuring that the compressed model achieves accuracy closer to the original model. These methods directly truncate the matrix based on the magnitude of the singular values, which can lead to performance degradation. To address this, SVD-LLM (Wang et al., 2024), which establishes a direct mapping between singular values and compression loss, exhibits less performance degradation and is better at recovering performance through fine-tuning. However, these methods remain limited to linear approaches for low-rank compression. These linear approaches are accompanied by large and irreducible information loss between high-rank weight matrices and the low-rank matrices during compression. Beyond standalone quantization or low-rank approaches, several recent works explore hybrid compression strategies that jointly leverage both techniques. One such method is CALDERA (Saha et al., 2024), a compression algorithm for LLMs that exploits the inherent low-rank structure by approximating the weight matrix W via a low-rank, low-precision decomposition. Different from these linear approaches, our proposed NLA leverages non-linear methods to achieve less information loss with larger compression ratios.

Nonlinear Matrix Representation Methods. Nonlinear methods have been widely applied in the field of machine learning. KPCA(Kernel Principal Component Analysis, Schölkopf et al. (1998)) uses kernel functions to map the data into a high-dimensional feature space to capture the nonlinear structures in the data. Dynamics-inspired Neuromorphic Learning (DyN) (Pei & Wang, 2023) and its subsequent work (Pei et al., 2024) is a new paradigm that uses a neural-weight-free architecture consisting of finite sub-models instead of the weight-based model structure inspired by the theory of Hamilton dynamics. DyN interprets FC layers as integrals of the neuronal path, demonstrating comparable or even better performance than weight-based neural models with fewer parameters. Similar ideas have been applied in various domains, such as fine-tuning (Shen et al., 2024) and graph neural networks (Sun et al., 2024). Inspired by these methods, we treat the piecewise-linear model as a low-rank approximator for pre-trained weight matrices, achieving a high compression ratio and low-performance degradation simultaneously. By designing an optimized forward propagation algorithm and introducing adaptive strategies, we facilitate a more comprehensive application of the nonlinear method on model compression.

3 Method

3.1 Preliminaries

Low-Rank Approximation. The low-rank approximation methods (Yuan et al., 2023; Wang et al., 2024) have emerged as a promising technique for model compression, which approximates the

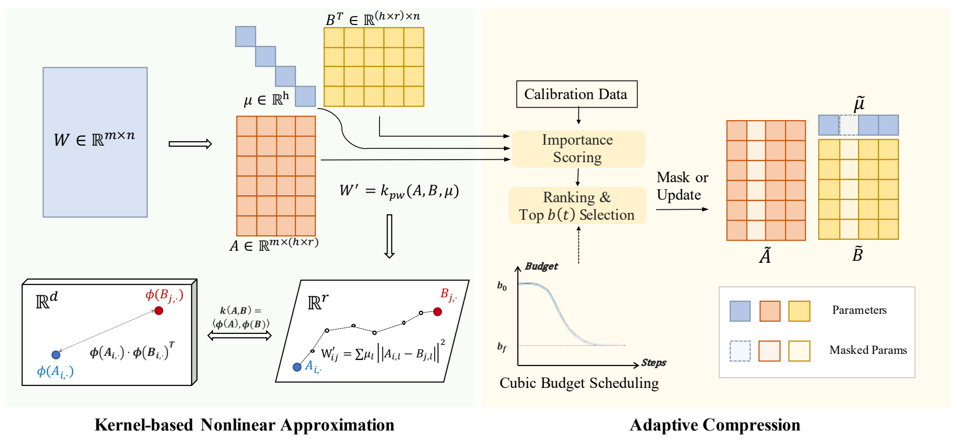

Figure 2: Overview of our proposed NLA framework. (Left) Kernel-based Nonlinear Approximation: approximate a pre-trained weight matrix W with two low-rank factors A, B and shared coefficients μ through a piecewise-linear kernel function k_{pw}. (Right) Adaptive Compression: compute the important scores of the triplets, rank the scores under a cubic budget scheduling strategy, and mask the least important triplets while updating the others.

pre-trained weight matrix $W \in \mathbb{R}^{m\times n}$ with the product of smaller low-rank matrices. Specifically, they first apply singular value decomposition (SVD) to decompose the weight matrix, then compress W to W_r by retaining only the top r singular values and their corresponding singular vectors.

$$W = U\Sigma V^T, W_r = U_r\Sigma_r V_r^T \tag{1}$$

where $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_{\min(m,n)})$ with singular values $\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant \sigma_{\min(m,n)}$. Despite being easy to implement, SVD-based linear truncation inevitably leads to significant information loss, as more informative components are discarded with increasing compression ratios.

Kernel Function. A kernel function $k(\cdot,\cdot)$ implicitly maps data from an input space $\mathcal{X}$ to a high-dimensional Hilbert feature space $\mathcal{H}$ via a mapping function $\phi : \mathcal{X} \to \mathcal{H}$. The inner product in $\mathcal{H}$ can be computed efficiently in the original space $\mathcal{X}$ by:

$$k(\mathbf{x}, \mathbf{x}') = \langle\phi(\mathbf{x}), \phi(\mathbf{x}')\rangle_{\mathcal{H}} \tag{2}$$

This capability of kernel methods, operating in high- or even infinite-dimensional Reproducing Kernel Hilbert Spaces (RKHS), allows them to represent highly nonlinear functions that go beyond simple linear hypothesis classes (Hofmann et al., 2008), thereby providing a powerful tool of modeling complex relationships in high-dimensional spaces by leveraging low-dimensional vectors.

3.2 Kernel-based Nonlinear Approximation

To capture richer interactions beyond linear low-rank structure, we choose a piecewise-linear kernel inspired by Pei & Wang (2023), which introduces nonlinearity while remaining computational lightweight. Given a weight matrix $W \in \mathbb{R}^{m\times n}$, we approximate it with a matrix W' represented by two smaller matrices $A \in \mathbb{R}^{m\times h\times r}$ and $B \in \mathbb{R}^{n\times h\times r}$ with piecewise linear kernel k_{pw},

$$W'_{i,j} = k_{pw}(A_{i,\cdot,\cdot}, B_{j,\cdot,\cdot}) = \sum_{l=1}^{h} \mu_l \,||A_{i,l,\cdot} - B_{j,l,\cdot}||^2 \tag{3}$$

where W'_{ij} denotes the (i,j)-th element of W'. To minimize the difference between the approximation matrix W' and the target weight matrix W, we use the gradient descent optimization method to adjust the distribution of low-dimensional vectors. The optimization objective is

$$W^* = \arg\min_{A,B} \mathcal{L}_{MSE}(W', W). \tag{4}$$

Through the optimization, we can compress a pre-trained weight matrix containing $m \times n$ parameters into low-rank matrices containing $(m+n)\cdot r\cdot h + h$ parameters in a non-linear manner. The introduction of nonlinearity largely reduces the information loss during compression.

However, nonlinearity is not a free lunch. During the forward propagation, introducing nonlinearity necessitates merging the low-rank matrices into a large-weight matrix before processing input features. Unlike the linear approach, one cannot directly multiply the features by the low-rank matrices, which leads to significant memory consumption during training and inference. To this end, we propose a specifically designed forward algorithm 1. This algorithm enables forward propagation for nonlinearly compressed low-rank models without the need to merge weight matrices, significantly reducing memory usage during training and inference. Algorithm 1 computes the nonlinear kernel approximation in a factorized manner by decomposing the kernel into three components: the input self-term, the output self-term and the interaction term. Specifically, the piecewise linear kernel is expanded into its additive parts, and each component is computed using only the nonlinear factors and the shared coefficients μ, without ever constructing the full weight matrix. The detailed mathematical proofs and algorithmic resource consumption are provided in Appendix A.

Algorithm 1 Forward with Nonlinear Kernels

```
Require: Input features $X \in \mathbb{R}^{b\times m}$, input neurons $Q_{\text{in}} \in \mathbb{R}^{m\times h\times r}$, output neurons $Q_{\text{out}} \in \mathbb{R}^{n\times h\times r}$,
    shared coefficients $\mu \in \mathbb{R}^{h}$.
Compute input & output self-term:
    $\text{InS} \leftarrow \sum_{k=1}^{r} Q_{\text{in}}[:,:,k]^2 \odot \mu$
    $\text{OutS} \leftarrow \sum_{k=1}^{r} Q_{\text{out}}[:,:,k]^2 \odot \mu$
Aggregate input & output contribution:
    $X_{\text{in}} \leftarrow X \cdot \text{InS}^\top$
    $X_{\text{out}} \leftarrow \mathbf{1}_b \cdot \left(\sum_{i=1}^{m} X_{:,i}\right) \cdot \text{OutS}^\top$
Compute interaction term:
    $X_{\text{in\&out}} \leftarrow (Q_{\text{out}} \cdot (Q_{\text{in}}^\top X^\top)) \odot \mu$
    $X_{\text{in\&out}} \leftarrow \sum_{l=1}^{h} X_{\text{in\&out}}[l]$
Final output:
    $Y \leftarrow X_{\text{in}} + X_{\text{out}} - 2 \cdot X_{\text{in\&out}}$
```

3.3 Adaptive Compression

Through the above method, we can compress the weight matrix into low-rank matrices. However, compressing all weight matrices to the same rank is not the optimal strategy. This is because compressing different matrices has varying impacts on model performance, and the difficulty of compression also differs between matrices. As shown in Fig. 3, the final mean loss and the rate of loss change during approximation vary significantly between different layers and matrices, such as the Q and FC matrices. This highlights that compression difficulty is not uniform across all matrices.

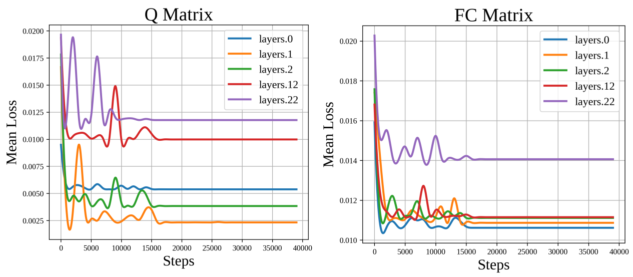

Figure 3: The loss value during nonlinear approximation of different weight matrices. Experiments are conducted with the Q matrix and FC matrix of different layers in OPT-1.3B.

To achieve a higher compression rate while preventing a significant performance degradation, we design an adaptive compression method for the nonlinear low-rank approximation. Inspired by (Zhang et al., 2023), we adaptively adjust the compression ratio for each weight matrix based on its importance score.

For a weight matrix W in MLP or Attention layers, our nonlinear formulation decomposes it into three low-rank components due to Section 3.2: $A \in \mathbb{R}^{m\times h\times r}$, $B \in \mathbb{R}^{n\times h\times r}$ and a shared coefficient vector $\mu \in \mathbb{R}^h$. This naturally defines h segment-level triplets

$$\tau_l = \{A_l = \{a_{i,l,\cdot}, i \in [1,m]\}, \mu_l, B_l = \{b_{j,l,\cdot}, j \in [1,n]\}\}, l \in [1,h] \tag{5}$$

This structure provides a convenient granularity for computing importance scores and adaptively allocate budgets of each matrices.

To adaptively compress the matrix, we assign an importance score $s_{W,l}$ to each triplet based on its contribution to the model. This score is a weighted sum of the importance $s(\cdot)$ of the shared coefficient μ_l and the average importance of its associated low-rank matrices A_l and B_l.

$$s_{W,l} = s(\mu_l) + \frac{1}{m\times r}\sum_{i=1}^{m}\sum_{j=1}^{r} s(A_{i,l,j}) + \frac{1}{n\times r}\sum_{i=1}^{n}\sum_{j=1}^{r} s(B_{i,l,j}) \tag{6}$$

To make the importance scores more robust, we use a sensitivity-based approach. The importance of an individual parameter w is defined as the absolute product of its value and the gradient: $I(w) = |w \cdot \nabla_w \mathcal{L}|$. To account for the dynamic nature of the training process, we compute a smoothed importance score $\bar{I}$ and a corresponding uncertainty measure $\bar{U}$ over time step:

$$\begin{aligned}\bar{I}^{(t)}(w) &= \beta_1 \bar{I}^{(t-1)}(w) + (1-\beta_1) I^{(t)}(w) \\ \bar{U}^{(t)}(w) &= \beta_2 \bar{U}^{(t-1)}(w) + (1-\beta_2) \left| I^{(t)}(w) - \bar{I}^{(t)}(w) \right|\end{aligned} \tag{7}$$

Here, β_1 and β_2 control the exponential smoothing strength. The final importance score $s(w)$ for any parameter is then defined as the product of the smoothed importance and its uncertainty:

$$s(w) = \bar{I}(w) \cdot \bar{U}(w) \tag{8}$$

During training, we dynamically adjust the overall compression budget to preserve performance. This is crucial because aggressively compressing the model too early can destabilize the training process. We use a cubic sparsity scheduling method (Sanh et al., 2020) to gradually reduce the number of active parameters. The budget $b(t)$ at a given step t is defined as:

$$b(t) = \begin{cases} b_0 & t < t_i \\ b_f + (b_0 - b_f)\left(1 - \frac{t - t_i - t_f}{T - t_i - t_f}\right)^3 & t_i \leqslant t < T - t_f \\ b_f & T - t_f \leqslant t < T \end{cases} \tag{9}$$

Here, b_0 is the initial budget and b_f is the final budget. The warm-up and cool-down steps, t_i and t_f, can ensure a smooth transition.

By combining the importance scores with the dynamic budget, we can mask the less important shared coefficients μ_l and thus effectively adjust the compression ratio for each matrix. The mask is applied during the update step:

$$\tilde{\mu_l} = \begin{cases} \mu_l - \eta \nabla_\mu \mathcal{L} & s_{W,l} \text{ is the top } b \text{ of all scores in } S \\ 0 & \text{else} \end{cases} \tag{10}$$

where S is the set of all importance scores for all shared coefficients, and b is the current model budget. This process allows us to allocate the parameter budget while minimizing the impact on performance degradation for model compression.

4 Experiments

Baselines. For LLM, we compare our method against four baselines, including SVD and state-of-the-art linear low-rank approximation methods FWSVD (Hsu et al., 2022), ASVD (Yuan et al., 2023), and SVD-LLM (Wang et al., 2024). For vision models, we compared our method with TinyBert (Jiao et al., 2020), MaskAlign (Xue et al., 2023), and PELA (Guo et al., 2024).

Datasets and base models. To demonstrate the effectiveness of our method, we evaluate the performance of NLA and the baselines in several models and datasets. For LLM, we select the models LLaMA (Touvron et al., 2023a) and OPT (Zhang et al., 2022). The models are estimated in ten different datasets, including language modeling datasets (Wikitext2 (Merity et al., 2017), PTB (Marcus et al., 1993), C4 (Raffel et al., 2020)) and common sense reasoning datasets (OpenbookQA (Mihaylov et al., 2018), ARC-e and ARC-c (Clark et al., 2018), WinoGrande (Sakaguchi et al., 2021), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), and MathQA (Amini et al., 2019)). For vision models, we test on Swin-Transformer (Liu et al., 2021) with the ImageNet1k (Deng et al., 2009) dataset. All of our experiments are conducted on 2 NVIDIA A100 GPUs. The hyperparameter details of the experiments are provided in Appendix C.

4.1 Main Experimental Results

Experiments on LLMs. To thoroughly compare the performance of NLA with the baselines, we perform three sets of experiments with LLaMA-7b (Touvron et al., 2023a): (i) language modeling tasks, (ii) common sense reasoning tasks, and (iii) the performance of compressed models after

Table 1: The perplexity of LLaMA-7b compressed by NLA and baselines under different compression ratios on Wikitext2 (Merity et al., 2017), PTB (Marcus et al., 1993) and C4 (Raffel et al., 2020).

METHOD	RATIO	WikiText-2	PTB	C4
Original	0%	5.68	8.35	7.34
SVD	40%	52489	59977	47774
	50%	131715	87227	79815
	60%	105474	79905	106974
FWSVD	40%	18156	20990	12847
	50%	24391	28321	23104
	60%	32194	43931	29292
ASVD	40%	1407	3290	1109
	50%	15358	47690	27925
	60%	57057	45218	43036
SVD-LLM	50%	23.97	150.58	118.57
	60%	42.30	321.27	246.89
NLA (Ours)	50%	**15.21**	**84.65**	**70.32**
	70%	39.91	320.35	240.56

Table 2: Perplexity of LLaMA-7b compressed by NLA and SVD-LLM (Wang et al., 2024) under different compression ratios on Wikitext2 (Merity et al., 2017) before and after fine-tuning.

METHOD	RATIO	WikiText-2
Original	0%	5.68
Performance Before Fine-tuning		
SVD-LLM	50%	23.97
NLA (ours)	50%	**15.21**
SVD-LLM	70%	185
NLA (ours)	70%	**39.91**
Performance After Fine-tuning		
SVD-LLM	50%	13.26
NLA (ours)	50%	**9.68**
SVD-LLM	70%	28.45
NLA (ours)	70%	**17.77**

Table 3: Performances of LLaMA-7b compressed by NLA and baselines under different compression ratio on common sense reasoning datasets measured by accuracy. Compression ratio is calculated as the ratio of the number of reduced parameters to the number of parameters in the original model.

METHOD	RATIO	Openb.	ARC_e	WinoG.	HellaS.	ARC_c	PIQA	MathQA	Average
Original	0%	0.28	0.67	0.67	0.56	0.38	0.78	0.27	0.52
SVD	40%	0.15	0.26	0.51	0.26	0.21	0.54	0.22	0.30
	50%	0.16	0.26	0.50	0.26	0.23	0.52	0.19	0.30
	60%	0.16	0.26	0.50	0.26	0.22	0.52	0.21	0.30
FWSVD	40%	0.16	0.26	0.51	0.26	0.22	0.53	0.21	0.30
	50%	0.12	0.26	0.50	0.26	0.23	0.53	0.20	0.30
	60%	0.15	0.26	0.49	0.26	0.22	0.53	0.18	0.30
ASVD	40%	0.13	0.28	0.48	0.26	0.22	0.55	0.19	0.30
	50%	0.12	0.26	0.51	0.26	0.22	0.52	0.19	0.30
	60%	0.12	0.26	0.49	0.26	0.21	0.51	0.18	0.29
SVD-LLM	50%	0.16	0.33	0.54	0.29	0.23	0.56	0.21	**0.33**
	60%	0.14	0.28	0.50	0.27	0.22	0.55	0.21	0.31
NLA (ours)	**50%**	**0.19**	**0.43**	**0.58**	**0.33**	**0.26**	**0.60**	0.21	**0.37**
	70%	0.16	0.33	0.50	0.28	0.23	0.57	**0.22**	0.33

fine-tuning. The compression ratio is defined as the fraction of parameters removed relative to the total number of original parameters.

We first evaluate the perplexity of LLaMA-7B compressed by NLA and baseline models on three different datasets, Wikitext2, PTB, and C4. The results in Table 1 show that NLA consistently achieves lower perplexity than the baselines at the same compression ratio. For instance, at a 50% compression ratio, NLA reduces perplexity to 15.21 on Wikitext2, which is significantly better than SVD-LLM (23.97) and far below all other baselines. More importantly, even under more aggressive compression (e.g., 70%), NLA still maintains competitive performance: its perplexity (39.91) is substantially lower than SVD-LLM at the same ratio (185) and even outperforms SVD-LLM at 60% compression (42.30). Similar trends are also observed on PTB and C4. These results indicate that NLA not only achieves superior performance at comparable compression ratios but also retains robustness when pushed to higher compression.

To further evaluate the effectiveness of NLA, we test the performance of LLaMA-7b compressed by NLA and baseline methods on seven common sense reasoning datasets. The results are summarized in Table 3. As shown, NLA achieves the best overall performance at a 50% compression ratio, with an average accuracy of 0.37, surpassing the best-performing baseline (SVD-LLM at 50%, 0.33) by 0.04. Even at a higher compression ratio of 70%, NLA maintains competitive accuracy (0.33 average), comparable to SVD-LLM at 60% while reducing more parameters. These results demonstrate that NLA not only achieves superior performance under moderate compression but also sustains competitive accuracy under aggressive compression, highlighting its robustness and effectiveness for commonsense reasoning tasks.

To estimate the performance of NLA with fine-tuning, we fine-tune LLaMA-7B compressed by NLA, ASVD, and SVD-LLM on Wikitext-2. As shown in Table 2, NLA consistently surpasses SVD-LLM before and after fine-tuning. Specifically, on the WikiText-2 dataset, the perplexity of NLA is more than 70% lower than that of SVD-LLM with the same compression ratio before pre-training, and 37% lower after pre-training. Furthermore, fine-tuning the NLA-compressed model results in a 55% reduction in perplexity, indicating that the nonlinear low-rank approximation not only compresses the model effectively but also preserves its capacity to improve performance through fine-tuning.

In summary, both pre- and post-fine-tuning results consistently verify that NLA delivers a more efficient and effective compression strategy than existing SVD-based methods. It achieves better performance at the same compression ratio, and even under higher compression ratios, NLA remains competitive with, or superior to, the best baselines. This shows the effectiveness and robustness of NLA for compressing large-scale language models.

Table 4: Image classification performance of the compressed model on ImageNet-1K.

METHOD	Params (M)	Ratio	GFLOPs	Throughput	Acc (%)
Swin-Base	87.8	0%	33.7	293.73	83.5
TinyBert	58.6	33.3%	20.6	-	78.8
MaskAlign	58.6	33.3%	20.6	-	79.1
PELA	62.2	29.2%	21.3	295.96	82.4
NLA (ours)	**40.2**	**54.2**%	21.9	231.68	**82.7**

Experiments on vision models. We apply our method to the Swin-Transformer. After compressing the Swin-Base model with a compression ratio of approximately 54%, we fine-tune it on the ImageNet-1k dataset and test it on the corresponding test set. The results are reported in Table 4. As shown, NLA achieves higher accuracy than other baselines after fine-tuning, while reducing the parameter count more substantially. In particular, compared to PELA, NLA reduces the number of parameters by 35.3% while delivering better top-1 accuracy. Moreover, NLA retains performance close to the original model (82.7% vs. 83.5%) despite a reduction of more than half of the parameters.

The experiments above validate the advantages of NLA over existing low-rank approximation techniques in compressing both vision and language models. The results emphasize the flexibility of NLA, which adapts effectively to various types of models, making it a robust choice for a wide range of real-world applications. Furthermore, NLA's ability to retain and even enhance performance after compression and fine-tuning positions it as a promising solution for model compression tasks, particularly for large-scale models.

Table 5: Perplexity of NLA and SVD-based methods on Wikitext-2, we only convert the layers of LLaMA-7B without further training

METHOD	RATIO	WIKITEXT-2
SVD	60%	105474
FWSVD	60%	32194
ASVD	60%	57057
NLA (LOW-RANK ONLY)	70%	**16774**

Table 6: Mean loss during the fitting of weight matrices of OPT-1.3B using SVD and nonlinear low-rank approximation.

LAYER	NAME	SVD LOSS	NLA LOSS
0	Q	0.0118	0.0053
0	K	0.0119	0.0052
0	FC1	0.0177	0.0106
5	Q	0.0250	0.0074

4.2 Ablation Study

Ability of linear vs. nonlinear low-rank approximation. We conduct an experiment to verify the differences in the ability of linear low-rank approximation and nonlinear low-rank approximation to fit the weight matrices in models. We first compare SVD-based methods and NLA on LLaMA-7B without further training. As

Table 7: Memory usage during training and inference of different models with or without the adoption of Algorithm 1 in NLA. The batch size used for testing is 1.

MODEL	Params (M)	With Alg. 1	Inference (MB)	Training (MB)
Swin-Tiny	9.3	✗	190.01	1053.47
		✓	105.76	155.47
Swin-Base	40.2	✗	430.58	3683.84
		✓	273.64	456.21
OPT-125m	73.5	✗	3440.70	8857.18
		✓	374.10	1562.14
OPT-1.3B	461.7	✗	18053.07	>81920
		✓	2204.22	12623.75

Table 8: Ablation of adaptive budget allocation with Swin-Transformer on ImageNet-1K and OPT on WikiText2

MODEL	Adaptive	Acc (%)	Perplexity
Swin-Tiny	✗	77.40	-
	✓	77.63	-
Swin-Base	✗	82.55	-
	✓	82.73	-
OPT-125M	✗	-	31.52
	✓	-	31.34
OPT-1.3B	✗	-	20.13
	✓	-	19.77

Table 9: Ablation on kernel functions. We report average inference runtime (s) and peak GPU memory (GB) measured on OPT-1.3B and LLaMA-7B

Kernel	OPT-1.3B		LLaMA-7B	
	Time (s)	Mem (GB)	Time (s)	Mem (GB)
Gaussian	0.45	2.7	1.45	6.5
Sigmoid	0.37	2.4	1.32	6.3
Cosine	0.31	2.3	1.26	6.2
Piece-wise Linear (ours)	**0.25**	2.8	**0.82**	6.7

Table 10: Perplexity of LLaMA-7B compressed by GPTQ combined with SVD-LLM and NLA.

Method	Memory (GB)	Perplexity
4-bit	3.9	6.21
3-bit	2.8	16.28
SVD-LLM + 4-bit	2.1	13.29
NLA + 4-bit	**1.9**	**12.18**

shown in Table 5, NLA achieves a perplexity of 16,774 at a 70% compression ratio, which is over 70% lower than the best SVD-based result (32,194 with FWSVD at 60%). To further evaluate the performance of SVD and NLA in fitting weight matrices, we conduct experiments on the OPT-1.3B model that compares the mean loss for each layer using both SVD and NLA. As shown in Table 6, for different weight matrices in various layers, approximating with NLA gains much less mean loss than with SVD. These results demonstrate that nonlinear approximation (NLA) consistently achieves lower mean losses compared to SVD, highlighting its superior ability to better fit the model's weight matrices.

Memory-saving algorithm. As discussed in Section 3.2, we adopt the forward algorithm in Algorithm 1 to reduce memory usage during model execution. To verify its effectiveness, we measure the memory footprint of the compressed models on both vision and language tasks, considering inference (without gradients) and training (with backpropagation) under a batch size of 1. The test results are shown in Table 7. For OPT-1.3B, the proposed algorithm reduces memory consumption to 12.2% of the baseline during training and 15.4% during inference. Similar reductions are observed for other models. Overall, these results demonstrate that the proposed memory-saving algorithm substantially decreases memory usage in both training and inference across different architectures.

Ablation on Kernel Functions. To assess the effect of different kernel choices in our nonlinear low-rank approximation, we compare the proposed piecewise-linear kernel with alternative options, including Gaussian, Sigmoid, and Cosine kernels. Table 9 summarizes the inference costs in terms of runtime (s) and peak GPU memory (GB). We observe that all kernels consume a similar amount of GPU memory during inference (2.3–2.8 GB & 6.2-6.7 GB), while the Piece-wise Linear kernel achieves the fastest inference speed on both models. These results suggest that although different kernels provide comparable memory footprints, their computational efficiency varies significantly. The piecewise-linear kernel consistently yields the most favorable balance between accuracy, efficiency, and memory usage, which justifies our choice to adopt it as the default kernel in NLA.

Ablation of adaptive budget allocation. We compare NLA with and without the adaptive budget allocation module on both vision and language models in Table 8. For vision tasks, we evaluate Swin-Tiny and Swin-Base; incorporating adaptive budget allocation yields higher top-1 accuracy. For language tasks, we test OPT-125M and OPT-1.3B on WikiText2, where adaptive allocation consistently lowers perplexity. Across both model families, the addition of adaptive budget allocation brings measurable performance improvements, confirming its effectiveness.

Computational Complexity and Throughput. We compare the computational complexity and throughput of nonlinear approximation methods and linear methods on the vision model and language model. As shown in Table 4, with comparable computations and throughput, our NLA achieves superior performance with fewer parameters on Swin-Base, demonstrating that the integration of non-linear techniques can achieve a more favorable trade-off between performance and resource requirements. For language models, NLA introduces a slightly longer inference time but reduces GPU memory usage by 35%, while achieving superior performance (17.77 vs 17.93) compared to linear SVD baselines, as shown in Table 15. This significant reduction in memory makes NLA particularly suitable for deploying large models on memory-constrained devices.

Quantization and Low-Rank Combination. To demonstrate the compatibility of NLA with other compression methods, we tested the perplexity of LLaMA-7B on Wikitext-2 when compressed by GPTQ (Frantar et al., 2022), combining with two low-rank approximation methods. As shown in Table 10, NLA combined with 4-bit quantization attains the lowest memory footprint (1.9 GB) while delivering the best perplexity (12.18), outperforming both GPTQ-3bit and SVD-LLM combined with GPTQ-4bit. This result shows the strong compatibility between NLA and quantization methods.

5 Conclusion

In this paper, we present NLA, a nonlinear low-rank approximation method with adaptive budget allocation. By replacing linear products with a piecewise-linear kernel and dynamically allocating compression budgets across layers, NLA achieves higher compression ratios while preserving most of the original ability within the pre-trained model. For memory-efficient inference with nonlinear methods, we design a special forward algorithm that enables inference and training without re-computing full weight matrices. Across multiple baselines on vision and language models, NLA demonstrates more favorable compression–performance trade-offs than linear low-rank methods, especially under high compression ratios, and ablations confirm the benefits of adaptive allocation and the choice of kernel. In future work, we aim to extend NLA to multi-modal tasks and investigate strategies to further reduce its computational and temporal costs.

Ethics statement

We assert that the methodology and applications of this study do not involve human subjects, generate harmful insights, or engage in applications that could lead to discrimination, bias, or social harm. All datasets used in this research are publicly available benchmark datasets widely used in academic studies. We commit to adhering to all applicable data privacy and intellectual property laws and regulations. We confirm that we have read and understood the ICLR Code of Ethics and pledge to comply with its guidelines in all aspects of our research, paper submission, and subsequent interactions.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China under grant number 62236008, in part by the Natural Science Foundation of Beijing under grant number L251082, and in part by Shandong Provincial Natural Science Foundation under project ZR2025ZD01.

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 *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2357–2367, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1245. URL `https://aclanthology.org/N19-1245`.

Yonatan Bisk, Rowan Zellers, Ronan LeBras, Jianfeng Gao, and Yejin Choi. PIQA: reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 7432–7439. AAAI Press, 2020. URL `https://aaai.org/ojs/index.php/AAAI/article/view/6239`.

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. *ArXiv preprint*, abs/1803.05457, 2018. URL `https://arxiv.org/abs/1803.05457`.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pp. 248–255. IEEE Computer Society, 2009. doi: 10.1109/CVPR.2009.5206848. URL `https://doi.org/10.1109/CVPR.2009.5206848`.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35: 30318–30332, 2022.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *ArXiv preprint*, abs/2210.17323, 2022. URL `https://arxiv.org/abs/2210.17323`.

Yangyang Guo, Guangzhi Wang, and Mohan Kankanhalli. Pela: Learning parameter-efficient models with low-rank approximation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15699–15709, 2024.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016.

Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. 2008.

Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. Language model compression with weighted low-rank factorization. In *International Conference on Learning Representations*, 2022.

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. *ArXiv preprint*, abs/2106.09685, 2021. URL `https://arxiv.org/abs/2106.09685`.

Yukun Huang, Yanda Chen, Zhou Yu, and Kathleen McKeown. In-context learning distillation: Transferring few-shot learning ability of pre-trained language models. *ArXiv preprint*, abs/2212.10670, 2022. URL `https://arxiv.org/abs/2212.10670`.

Yuxin Jiang, Chunkit Chan, Mingyang Chen, and Wei Wang. Lion: Adversarial distillation of proprietary large language models. *ArXiv preprint*, abs/2305.12870, 2023. URL `https://arxiv.org/abs/2305.12870`.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4163–4174, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.372. URL `https://aclanthology.org/2020.findings-emnlp.372`.

Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt

Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. *ArXiv preprint*, abs/1704.04760, 2017. URL `https://arxiv.org/abs/1704.04760`.

Mikhail Khodak, Neil Tenenholtz, Lester Mackey, and Nicolo Fusi. Initialization and regularization of factorized neural layers. *ArXiv preprint*, abs/2105.01029, 2021. URL `https://arxiv.org/abs/2105.01029`.

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 *International Conference on Machine Learning*, pp. 20336–20350. PMLR, 2023.

Chi-Heng Lin, Shangqian Gao, James Seale Smith, Abhishek Patel, Shikhar Tuli, Yilin Shen, Hongxia Jin, and Yen-Chang Hsu. Modegpt: Modular decomposition for large language model compression. *ArXiv preprint*, abs/2408.09632, 2024. URL `https://arxiv.org/abs/2408.09632`.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 10012–10022, 2021.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023.

Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. Teaching small language models to reason. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1773–1781, 2023.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993. URL `https://aclanthology.org/J93-2004`.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=Byj72udxe`.

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 *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260. URL `https://aclanthology.org/D18-1260`.

OpenAI. GPT-4 Technical Report. *ArXiv preprint*, abs/2303.08774, 2023. URL `https://arxiv.org/abs/2303.08774`.

Gunho Park, Baeseong Park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models. *ArXiv preprint*, abs/2206.09557, 2022. URL `https://arxiv.org/abs/2206.09557`.

Zhengqi Pei and Shuhui Wang. Dynamics-inspired neuromorphic visual representation learning. In *International Conference on Machine Learning*, pp. 27521–27541. PMLR, 2023.

Zhengqi Pei, Anran Zhang, Shuhui Wang, Xiangyang Ji, and Qingming Huang. Data-free neural representation compression with riemannian neural dynamics. In *Forty-first International Conference on Machine Learning*, 2024.

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. *Journal of machine learning research*, 21(140):1–67, 2020.

Rajarshi Saha, Varun Srivastava, and Mert Pilanci. Matrix compression via randomized low rank and low precision factorization. *Advances in Neural Information Processing Systems*, 36, 2023.

Rajarshi Saha, Naomi Sagan, Varun Srivastava, Andrea Goldsmith, and Mert Pilanci. Compressing large language models using low rank and low precision decomposition. *Advances in Neural Information Processing Systems*, 37:88981–89018, 2024.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in neural information processing systems*, 33:20378–20389, 2020.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

Shufan Shen, Junshu Sun, Xiangyang Ji, Qingming Huang, and Shuhui Wang. Expanding sparse tuning for low memory usage. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Junshu Sun, Chenxue Yang, Xiangyang Ji, Qingming Huang, and Shuhui Wang. Towards dynamic message passing on graphs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: open and efficient foundation language models. arxiv. *ArXiv preprint*, abs/2302.13971, 2023a. URL `https://arxiv.org/abs/2302.13971`.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *ArXiv preprint*, abs/2307.09288, 2023b. URL `https://arxiv.org/abs/2307.09288`.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. *ArXiv preprint*, abs/2210.07558, 2022. URL `https://arxiv.org/abs/2210.07558`.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. Svd-llm: Truncation-aware singular value decomposition for large language model compression. *ArXiv preprint*, abs/2403.07378, 2024. URL `https://arxiv.org/abs/2403.07378`.

Wei Wen, Cong Xu, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Coordinating filters for faster deep neural networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pp. 658–666. IEEE Computer Society, 2017. doi: 10.1109/ICCV.2017.78. URL `https://doi.org/10.1109/ICCV.2017.78`.

Haojun Xia, Zhen Zheng, Yuchao Li, Donglin Zhuang, Zhongzhu Zhou, Xiafei Qiu, Yong Li, Wei Lin, and Shuaiwen Leon Song. Flash-llm: Enabling cost-effective and highly-efficient large generative model inference with unstructured sparsity. *Proceedings of the VLDB Endowment*, 17(2):211–224, 2023.

Hongwei Xue, Peng Gao, Hongyang Li, Yu Qiao, Hao Sun, Houqiang Li, and Jiebo Luo. Stare at what you see: Masked image modeling without reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22732–22741, 2023.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183, 2022.

Zhihang Yuan, Yuzhang Shang, Yue Song, Qiang Wu, Yan Yan, and Guangyu Sun. Asvd: Activation-aware singular value decomposition for compressing large language models. *ArXiv preprint*, abs/2312.05821, 2023. URL `https://arxiv.org/abs/2312.05821`.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL `https://aclanthology.org/P19-1472`.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*. Openreview, 2023.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *ArXiv preprint*, abs/2205.01068, 2022. URL `https://arxiv.org/abs/2205.01068`.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics*, 12:1556–1577, 2024.

A Forward Propagation with Nonlinear Low-Rank Approximation

A.1 Forward Code

```
def forward(self, x):
    in_shape = x.shape
    assert in_shape[-1] == self.in_features
    x = x.reshape(-1, self.in_features)
    In_Qs_Square, Out_Qs_Square = self.In_Qs**2, self.Out_Qs**2

    In_Qs_Square = In_Qs_Square.sum(-1)*self.shared_coeff.unsqueeze(1)
    S_In_Qs_Square = x @ In_Qs_Square.permute(1, 0)
    S_In_Qs_Square = S_In_Qs_Square.sum(-1).unsqueeze(-1)

    Out_Qs_Square = Out_Qs_Square.sum(-1)*self.shared_coeff.unsqueeze(1)
    x_sum = x.sum(-1).unsqueeze(-1).repeat(1, self.num_Hs)
    S_Out_Qs_Square = x_sum @ Out_Qs_Square

    S_Inner_Product = self.In_Qs.permute(0, 2, 1) @ x.T
    S_Inner_Product = self.Out_Qs @ S_Inner_Product
    S_Inner_Product = S_Inner_Product*self.shared_coeff
                      .unsqueeze(-1).unsqueeze(-1)
    S_Inner_Product = S_Inner_Product.sum(0).permute(1, 0)

    out_shape = list(in_shape)
    out_shape[-1] = self.out_features
    x_out = S_In_Qs_Square + S_Out_Qs_Square - 2 * S_Inner_Product
    x_out = x_out.reshape(*tuple(out_shape))
    return x_out
```

Figure 4: The Python code for the forward propagation corresponding to Algorithm 1

A.2 Formula derivation

For $A \in \mathbb{R}^{m\times H\times r}, B \in \mathbb{R}^{n\times H\times r}$ representing the in and out neurons in Section 3.2, and $\mu \in \mathbb{R}^H$ representing the shared coefficients, we have

$$w'_{ij} = \sum_{h=1}^{H} \mu_h \left|\left|\mathbf{a}_{ih} - \mathbf{b}_{jh}\right|\right|^2 \tag{11}$$

where $\mathbf{a}_{ih}, \mathbf{b}_{jh} \in \mathbb{R}^r$. For an input $x = [x_1, \dots, x_m] \in \mathbb{R}^m$, we can calculate the output $y = [y_1, \dots, y_n]$ by

$$\begin{aligned}
y_j &= \sum_{i=1}^{m} w'_{ij} x_i = \sum_{i=1}^{m} x_i \sum_{h=1}^{H} \mu_h \left\|\mathbf{a}_{ih} - \mathbf{b}_{jh}\right\|^2 \\
&= \sum_{i=1}^{m} x_i \sum_{h=1}^{H} \mu_h (\mathbf{a}_{ih} - \mathbf{b}_{jh}) \cdot (\mathbf{a}_{ih} - \mathbf{b}_{jh}) \\
&= \sum_{i=1}^{m} x_i \sum_{h=1}^{H} \mu_h \left(\left\|\mathbf{a}_{ih}\right\|^2 + \left\|\mathbf{b}_{jh}\right\|^2 - 2\mathbf{a}_{ih} \cdot \mathbf{b}_{jh}\right) \\
&= \sum_{i=1}^{m} x_i \sum_{h=1}^{H} \mu_h \left\|\mathbf{a}_{ih}\right\|^2 + \sum_{i=1}^{m} x_i \sum_{h=1}^{H} \mu_h \left\|\mathbf{b}_{jh}\right\|^2 - 2\sum_{i=1}^{m} x_i \sum_{h=1}^{H} \mu_h \mathbf{a}_{ih} \cdot \mathbf{b}_{jh}
\end{aligned} \tag{12}$$

Thus, the multiplication of activations and matrices can be swapped with the piecewise linear operation, allowing for the separate calculation of A^2, B^2 and $A \cdot B$, thereby saving memory.

A.3 Complexity Analysis

The FLOP count of Algorithm 1 can be calculated as follows:

- The FLOP count for $SumX_{in} = O(H \cdot r \cdot in) + O(H \cdot r \cdot in) + O(2 \cdot b \cdot H \cdot in)$

- The FLOP count for $SumX_{out} = 2O(H \cdot r \cdot out) + O(2 \cdot b \cdot H \cdot out)$.
- For $SumX_{in\&out}$, the FLOP count is $O(2 \cdot H \cdot r \cdot b \cdot (in + out))$.

So the total FLOP count of Algorithm 1 is

$$\begin{aligned} FLOPs =& 2O(H \cdot r \cdot in) + 2O(b \cdot H \cdot in) + 2O(H \cdot r \cdot out) \\ &+ 2O(b \cdot H \cdot out) + 2O(H \cdot r \cdot b \cdot (in + out)) \end{aligned} \tag{13}$$

"in" and "out" represent the input and output dimensions, respectively, and "b" denotes the batch size. And the total memory of Algorithm 1 is:

$$2 \cdot H \cdot r \cdot (in + out) \cdot 4 + b \cdot H \cdot 4 + b \cdot out \cdot 4 + b \cdot H \cdot out \cdot 4 \tag{14}$$

The computational results show that the computation and memory usage increase linearly with the number of parameters, so the increase in model parameters does not lead to an explosive growth in computation and memory usage.

B Kernel Interpretation of the Piecewise-Linear Function

The piecewise-linear function can be expressed as:

$$k_{pw}(A, B) = \sum_{l=1}^{h} \mu_l \left\| A_{l,\cdot} - B_{l,\cdot} \right\|^2 \tag{15}$$

For any coefficients α_i with $\sum_i \alpha_i = 0$, we have

$$\begin{aligned} S &= \sum_{i,j} \alpha_i \alpha_j \|x_i - x_j\|^2 = \sum_i \alpha_i \sum_j \alpha_j \|x_i - x_j\|^2 \\ &= \sum_i \alpha_i \sum_j \alpha_j \left(\|x_i\|^2 + \|x_j\|^2 - 2x_i \cdot x_j \right) \\ &= \left(\sum_j \alpha_j \right) \left(\sum_i \alpha_i \|x_i\|^2 \right) + \left(\sum_i \alpha_i \right) \left(\sum_j \alpha_j \|x_j\|^2 \right) - 2 \sum_i \alpha_i x_i \cdot \sum_j \alpha_j x_j \\ &= -2 \left\| \sum_i \alpha_i x_i \right\|^2 \leqslant 0 \end{aligned} \tag{16}$$

So that each individual term $\|\mathbf{a} - \mathbf{b}\|^2$ is a conditionally negative definite function, and thus its negation $-\|\mathbf{a} - \mathbf{b}\|^2$ is a valid conditionally positive definite(CPD) kernel. The weighted sum of such kernels

$$K(A, B) = -k_{pw}(A, B) = -\sum_{l=1}^{h} \mu_l \left\| A_{l,\cdot} - B_{l,\cdot} \right\|^2 \tag{17}$$

is also a CPD kernel. So the piecewise-linear function can be used as a kernel function.

Convergence guarantee. NLA is effectively performing a low-rank approximation in a Hilbert space where standard convergence guarantees of kernel methods apply, and where higher-order interactions can be represented linearly (Schölkopf et al., 1998). Moreover, the expressiveness of piecewise-linear structures is supported by the Dynamical Universal Approximation Theorem (Pei & Wang, 2023), which states that piecewise-linear systems can approximate arbitrary sequential functions.

C Experimental Details

We applied compression operations to the weight matrices of all linear layers, including those in the attention modules (Q, K, V, and Out). For LLaMA-7B, the hyperparameter r is set to 20, b_f is the final budget determined by the target compression ratio, and the initial budget $b_0 = 1.2b_f$. The hyperparameter H is obtained by dividing the initial budget b_0 by the number of weight matrices.

It can be observed from Appendix A.3 that under the same compression ratio (i.e., fixed $H \times r$), a larger H leads to higher computational complexity and memory usage. However, if H is too small, the number of segments in the piecewise linear function becomes insufficient, which may degrade model performance. Therefore, we chose a moderate setting for H and r to balance efficiency and performance.

In the retraining step, we use the Adam optimizer. We select 512 texts from the Wikitext-2 dataset as training data, with a batch size of 8, a learning rate of 5e-6, and train for 4 epochs.

For Swin-Transformer in Table 4, we used the standard fine-tuning recipe: AdamW optimizer, learning rate 1e-4, weight decay 0.05, batch size 64, 50 epochs. All low-rank baselines and NLA were retrained using exactly these hyperparameters with only the compression methods changed.

D Experiments

D.1 Ablation Study of the Budget Allocation Strategy

Table 11: Comparison of Cubic and Linear Budget Allocation Strategies

MODEL	STRATEGY	PERPLEXITY
OPT-1.3B	CUBIC	19.77
	LINEAR	43.24
OPT-125M	CUBIC	31.34
	LINEAR	63.04

We compare the cubic budget allocation strategy in Equation (9) with a linear strategy on OPT-1.3b and OPT-125m. The experimental results are shown in Table 11. The experimental results indicate that the cubic setting leads to less performance degradation compared to the linear setting.

D.2 Applicability of NLA across Different Model Scales

Table 12: Comparison of NLA and Baseline Methods on Models of Different Scales

METHOD	BASE MODEL	PARAMS.	PERPLEXITY (↓)/ACC. (↑)
PELA	SWIN-90M	62.2M	82.4
NLA	SWIN-90M	40.2M	**82.7**
MODEGPT	OPT-125M	75M	38.37
NLA	OPT-125M	63M	**31.34**
MODEGPT	OPT-1.3B	0.78B	21.92
NLA	OPT-1.3B	0.65B	**19.77**
SVD-LLM	LLAMA-7B	2.9B	17.93
NLA	LLAMA-7B	2.1B	**17.76**

To verify the scalability of NLA, we conduct experiments on models of different scales, including vision models (Swin-Transformer on ImageNet-1K) and language models (OPT and LLaMA on Wikitext2). As shown in Table 12, NLA consistently achieves a lower parameter count while maintaining or even improving performance. On WikiText2, NLA demonstrates clear advantages over MoDeGPT (Lin et al., 2024) and SVD-LLM (Wang et al., 2024) across different model sizes: it reduces parameters more aggressively and achieves lower perplexity on both OPT and LLaMA-7B. These results confirm that NLA provides a scalable and robust compression framework applicable to both vision and language models of various sizes.

D.3 Linear vs. Nonlinear Approximation Loss

Table 13 serves as a supplementary result to Table 6, providing a more fine-grained comparison between linear (SVD) and nonlinear (NLA) low-rank approximation. For different weight matrices in various layers, approximating with NLA gains much less mean loss than with SVD.

Table 13: Mean loss during the fitting of different weight matrices using SVD and nonlinear low-rank approximation.

LAYER	NAME	SVD LOSS	NLA LOSS
0	Q	0.0118	0.0053
0	K	0.0119	0.0052
0	V	0.0056	0.0026
0	OUT	0.0047	0.0021
0	FC1	0.0177	0.0106
5	Q	0.0250	0.0074
8	Q	0.0290	0.0114

D.4 INFERENCE MEMORY UNDER DIFFERENT COMPRESSION RATIOS.

Table 14: Memory usage during inference under different compression ratios. The batch size is set to 1.

MODEL	RATIO	INFERENCE MEMORY(MB)
SWIN-SMALL	0%	210.19
	60%	100.89
	70%	73.57
	80%	48.34
SWIN-BASE	0%	366.69
	60%	161.26
	70%	117.34
	80%	77.03
OPT-125M	0%	555.83
	50%	350.33
	55%	301.03
	60%	269.34
OPT-1.3B	0%	5175.34
	60%	2456.31
	70%	1870.06
	80%	1296.57

We evaluate the GPU memory consumption during inference at various compression ratios, as summarized in Table 14. Across all tested models, memory usage decreases substantially as the compression ratio increases. The effect is especially pronounced for larger models: for OPT-1.3B, inference memory drops from 5175.34 MB at 0% compression to only 1296.57 MB at 80% compression. Similar trends are observed for vision models such as Swin-Small and Swin-Base. These results demonstrate that our compression technique can significantly reduce inference memory footprint while preserving model functionality, which is particularly critical for deploying large-scale models in resource-constrained environments.

D.5 TIME AND MEMORY COST FOR LANGUAGE MODEL

Table 15: Inference Time and Memory Usage of LLaMA-7B compressed by SVD and NLA

METHOD	PARAMS	PERPLEXITY	RATIO	AVE. INFERENCE TIME (S)	GPU MEMORY (GB)
ORIGIN	7B	5.68	0%	-	26.27
SVD	3.0B	42.30	60%	0.409	13.89
NLA	2.1B	39.91	70%	0.593	8.97

E Reproducibility Statement

We are committed to ensuring the reproducibility of the results presented in this paper. All novel algorithms, including our Kernel-based Nonlinear Approximation method and the Adaptive Compression strategy, are described in detail in Section 3, Appendix A, and Appendix B. We will release our implementation as part of the supplementary material. Details regarding the experimental setup, hyperparameter configurations, and all utilized benchmark datasets are comprehensively documented in Section 4 and Appendix C.

F LLM Usage

Large Language Models (LLMs) were used solely as general-purpose assist tools to refine the language and improve the overall clarity of the writing. Specifically, an LLM was utilized for minor tasks such as grammar correction and enhancing sentence structure.