IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 4 December 2025, accepted 6 January 2026, date of publication 12 January 2026, date of current version 15 January 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3653132

RESEARCH ARTICLE

Compressing What Matters: Neuron Importance Meets Data-Aware Low Rank Approximation for Language Model Compression

ATHANASIOS NTOVAS, ALEXANDROS DOUMANOGLOU, PETROS DRAKOULIS, AND DIMITRIS ZARPALAS
Information Technologies Institute (ITI), Centre for Research and Technology Hellas (CERTH), 57001 Thessaloniki, Greece

Corresponding author: Athanasios Ntovas (atdovas@iti.gr)

This work was supported by the European Union (EU) Project VOXReality (Voice-Driven Interaction in Extended Reality (XR) Spaces) under Grant 101070521.

ABSTRACT To excel at their domain, large language models are comprised of billions of parameters. Yet, this comes at the cost of huge memory requirements, restricting their applicability in resource-constrained environments. To address the problem of neural network (NN) compression, Singular Value Decomposition (SVD) has played a key role as a fundamental component for matrix compression through decomposition. To minimize compression error and to maximize the efficacy of the compressed model on the downstream tasks, previous works focused on low-rank approximation of the NN's weight matrices either from the perspective of parameter importance or per-layer functional equivalence. While previous works studied the aforementioned perspectives in isolation, in this work, we are investigating the effectiveness of an approach that combines ideas from these two perspectives in a single objective. In parallel to this, an important aspect that affects the compression quality is the distribution of the compression rate across layers and NN parameters. Earlier works mostly considered distributing the compression rate uniformly across layers and network weights or relied on computationally expensive heuristic search. Contrary to them, in this work, we propose an enhanced and computationally efficient algorithm for dynamic compression rate allocation. Experimental results support the efficacy of the proposed approach, which performs on par or substantially better than the previous state-of-the-art, especially under high compression ratios.

INDEX TERMS Language models, neural network compression, neuron importance, SVD.

I. INTRODUCTION

The advances in Artificial Intelligence (AI) over the last decade have enabled a multitude of applications that were previously considered unreachable using traditional programming techniques. Nowadays, AI powers computer vision applications that can understand our world and chatbots that can communicate with humans in natural language, among others. Currently, AI is mainly realized by deep neural networks (DNNs), which are computational architectures that are trained to solve tasks from a particular domain at human-level performance. Large Language Models (LLMs) [1], [2], [3], [4], [5], [6] constitute DNNs that excel in Natural Language Understanding (NLU) and Processing (NLP) and enable fluent human-machine interfaces based on natural language. However, these DNNs come with a ton of parameters and a large memory and computational footprint. With AI becoming ubiquitous, their efficient deployment in resource-constrained environments such as edge devices, smartphones, and mini-PCs becomes a hard requirement. Thus, compressing LLMs for efficient deployment has naturally emerged as a field of its own [7], [8].

Since 2017, the transformer architecture [9] has been the main workhorse behind LLMs. In their simplest form, transformers process natural language by splitting sentences and words into tokens, which are subsequently processed sequentially by a series of transformer blocks, organized in layers. Each transformer block consists of a self-attention module and a feedforward layer, both of which are parameterized by matrices of weights, with each weight corresponding to a

The associate editor coordinating the review of this manuscript and approving it for publication was Jolanta Mizera-Pietraszko.

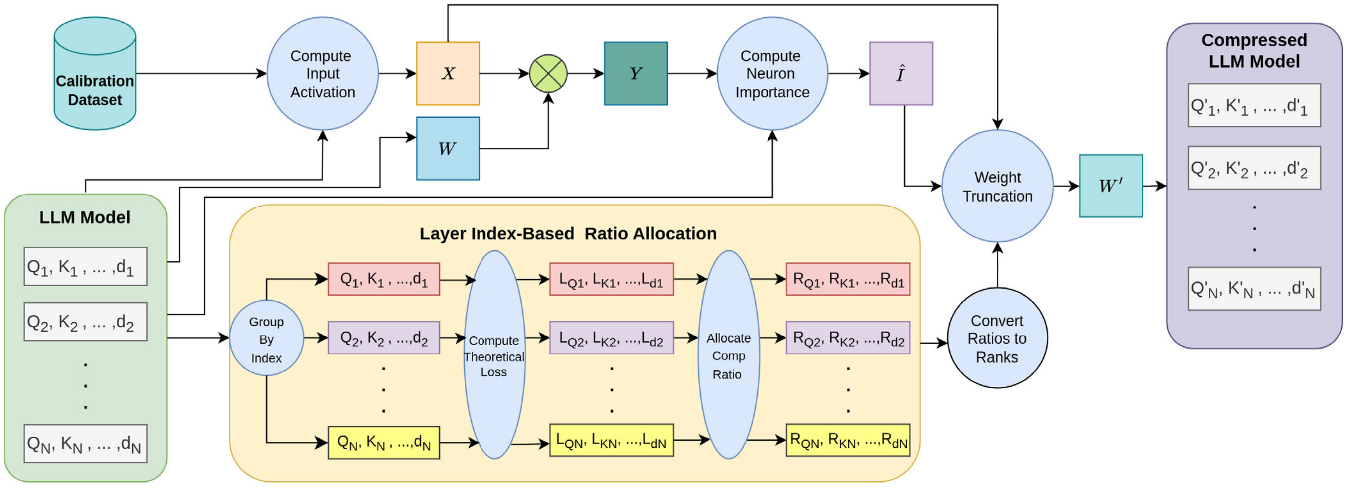


FIGURE 1. Architectural overview of the hybrid compression pipeline (NIDA-SVD). Our approach synthesizes the weight importance with data-aware low-rank approximation, guided by our dynamic rank allocation scheme. In the architectural representation, layer components such as Q_i (Query) and K_i (Key) are used to denote the corresponding compressible weight matrices W_{Q_i} and W_{K_i} respectively.

parameter that is tuned during network training. Compressing a transformer network can be accomplished in a variety of ways, but the end goal is common: to reduce the number of network parameters that need to be stored in memory with a minimal compromise in network performance.

One line of research for LLM compression attempts to reduce the LLM's parameters by approximating the weight matrices of the transformer blocks with the product of two low-rank matrices, a technique also known as *low-rank approximation*. Since low-rank, these matrices are crafted to have a sum of parameters that is less than the parameters found in the original matrix. Singular Value Decomposition (SVD) [10] is a matrix decomposition method [11] that plays a key role in the low-rank approximation problem, as, given a target rank, it is proven to produce the best possible approximation with a minimum matrix reconstruction error.

Beyond its naive use, SVD has been employed in more sophisticated approaches for compressing DNN matrices. First, using weighted low-rank factorization, FWSVD [12] takes into account the importance of each parameter to the DNN's output, improving model performance under equivalent compression ratios. And second, the data-aware approach of SVD-LLMv2 [13] aims for a functionally equivalent approximation of the DNN's weight matrix by taking into account layer inputs to be multiplied with the transformation matrix through a small calibration dataset. Each one of the approaches has demonstrated significant improvements compared to baselines, yet an approach combining ideas from both is currently missing from the literature.

In this work, our first contribution is to address this gap. Since data-aware low rank approximation has been the previous state-of-the-art in LLM compression, we opt for keeping it as a fundamental component of our method while seeking to improve it. Our investigation and analysis show that naively combining data-aware low rank approximation with the **parameter** importance estimation method that was suggested by FWSVD degrades the LLM's performance on downstream tasks compared to using data-aware low rank approximation alone. To overcome this, we propose to use **neuron** importance estimation, an alternative approach which considers parameters in functional groups, that often leads to substantial improvements in downstream model performance compared to the previous state-of-the-art under similar compression ratios.

Besides the contributions that we make in the compression method itself, we also consider the problem of parameter count allocation across weight matrices. Although in previous works uniform allocation has been the most common approach to distribute parameters across matrices, in the recent approach of SVD-LLMv2, a novel algorithm has been proposed as an improvement of the previous naive uniform strategy. This previous approach relies on grouping weight matrices based on their functionality in the transformer blocks. In this work, we make a second contribution by providing experimental evidence that a different grouping strategy, based on layer index, is more effective. Our analysis, ablation studies, and experimental findings support that when compressing BERT [2], DistilBERT [14], MobileBERT [15], and TinyBERT [16], foundational models for natural language understanding, in the vast majority of cases, the proposed approach achieves state-of-the-art performance compared to other SVD-based algorithms. Furthermore, we include a computational analysis across these compressed architectures (DistilBERT, MobileBERT, and TinyBERT) detailing the significant reduction in both MFLOPS/token and total parameter count.

II. RELATED WORK

Several approaches have been proposed to compress transformer architectures [9] in pre-trained LLMs [1], [2], [3], [4],

[5], [17], [18], [19]. Broadly, these methods fall into four categories.

Pruning methods [20], [21], [22], [23] aim to eliminate as many parameters as possible, effectively zeroing out the corresponding weights in the model's transformation matrices. On the one hand, **unstructured** pruning removes individual parameters without constraints, but typically requires specialized hardware for efficient deployment. On the other hand, to improve hardware compatibility, **structured** pruning removes entire rows or columns from weight matrices, which aligns well with standard hardware architectures optimized for fast matrix multiplication.

Distillation-based methods [14], [16], [24] leverage knowledge distillation [25] to reduce the number of neural network (NN) parameters by training a smaller model to mimic the behavior of the original. While often effective, this approach can be computationally expensive, as it requires retraining a model from scratch.

Third, and most relevant to our work, **low-rank approximation** methods [26] approximate transformer weight matrices using singular value decomposition (SVD), factorizing them into the product of two low-rank matrices. Standard SVD treats all entries as equally important for minimizing reconstruction error. FWSVD [12] improves on this by weighting parameters according to their importance to the model's output. Parameter importance is determined via differentiating the task loss with respect to the parameter and taking the magnitude of the gradient, a process that requires additional access to a calibration dataset. A more recent study [27] revealed that most transformer matrices are not low rank, and thus, when trying to compress them via low-rank approximation, the resulting compressed networks exhibit significant performance drops. However, it was found that the opposite holds for intermediate token representations, which appear to have a low rank structure. Since the rank of a matrix product is less than or equal to the minimum among the ranks of individual matrices participating in the multiplication, DRONE [28], ASVD [29], and SVD-LLM [30] approached compression from the perspective of functional equivalence, minimizing errors in the result of matrix multiplications. However, DRONE required caching intermediate feature activations, resulting in large memory requirements. SVD-LLM alleviated this restriction by relying only on their covariance matrix. At the same time, SVD-LLM ensured faithful estimation of the matrix multiplication error based on the truncated singular values, a limitation of ASVD. Yet, in its initial version, SVD-LLM relied on Cholesky decomposition, which (a) requires positive-definite matrices, a requirement that is not always fulfilled, and (b) can suffer from numerical instabilities during iterative optimization. Both issues were addressed by SVD-LLMv2 [13], using a two-step SVD algorithm. Low-rank approximation has also been combined with knowledge distillation in [31]. Our approach combines SVD-LLMv2, the best performing method among the previously discussed state-of-the-art, with ideas inspired from FWSVD.

In low-rank approximation approaches, a fundamental hyperparameter that controls the compression rate - performance tradeoff is the rank of the compressed matrix. Since the network itself has varying sensitivity with respect to the different matrices across layers, distributing the compression rate across matrices for optimal performance typically requires an exhaustive sensitivity analysis, which is practically infeasible. Some of the approaches [12], [30] simply distribute the compression rate evenly across layers, while other approaches [13], [28], [29] heuristically simplify the exhaustive search in more manageable terms. Our approach is computationally efficient and builds on top of the rank selection algorithm of SVD-LLMv2.

Fourth, and orthogonal to the previous approaches for LLM compression, is weight **quantization** [32], [33], [34], [35], which typically reduces the memory requirements to perform model inference by quantizing the parameters of the LLM under a fixed budget of binary digits. Recent advances in post-training quantization (PTQ) [36], demonstrate that high-accuracy LLM quantization can be achieved without retraining. In parallel, quantization-aware training (QAT) techniques [37] tailored for LLMs further improve robustness by modeling quantization noise during fine-tuning. Similar to other low-rank approximation techniques, our approach can additionally benefit from such schemes.

III. BACKGROUND

In this section, we provide foundational components to understand our approach. We begin by revisiting core elements of the Transformer architecture [9], followed by a description of BERT [2], the Language Model that will become the basic subject of our experiments and an overview of the architectures of DistilBERT, MobileBERT, and TinyBERT, given their role as additional models for our compression method. We then review SVD [10] and its application as a post-training compression technique. Finally, we provide an overview of two prominent SVD-based methods, FWSVD [12] and SVD-LLMv2 [13], which serve both as comparison baselines and inspiration for the development of the proposed method.

A. THE TRANSFORMER ARCHITECTURE

In transformer architectures, a token is the smallest unit of input (such as a word, subword, or character) that the model processes after being mapped to an embedding vector. Let $\boldsymbol{X} \in \mathbb{R}^{D\times N}$ denote a matrix of N tokens, each one represented by a D-dimensional vector in the embedding space.

The core of a Transformer is a stack of layers. Each **transformer layer** or **block** contains two main components: a self-attention mechanism and a feed-forward network (FFN). The self-attention mechanism uses two matrices, namely the query $\boldsymbol{Q} = \boldsymbol{W}_Q^T\boldsymbol{X}$ and key $\boldsymbol{K} = \boldsymbol{W}_K^T\boldsymbol{X}$ to weigh the importance of each token to the others and create a contextualized representation by linearly combining a matrix of values $\boldsymbol{V} = \boldsymbol{W}_V^T\boldsymbol{X}$. The output of the self-attention mechanism is subsequently transformed by a linear layer with weight matrix $\boldsymbol{W}_a$. The feed-forward network, also referred to as the multi-layer perceptron (MLP) block, then processes the previous output to learn higher-level features. In the self-attention mechanism, four weight matrices are involved, which are amenable to compression, namely $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V$, and $\boldsymbol{W}_a$, while the feed-forward block introduces two additional matrices, with the first performing up-projection and the second down-projection. We refer to those matrices as $\boldsymbol{W}_u$ and $\boldsymbol{W}_d$, respectively.

These two components of the transformer block are wrapped in a residual connection, and each one is followed by layer normalization, which both are essential for stable training and scaling to the deep architectures that characterize modern LLMs.

B. NEURONS AND PRE-ACTIVATIONS

A neuron in a NN can be understood as a simple function that computes the inner product between an input vector and a weight vector. If the input is an embedding vector $\boldsymbol{x} \in \mathbb{R}^D$ corresponding to a token representation, and the neuron's weights are $\boldsymbol{w} \in \mathbb{R}^D$, the output is given by $y = \boldsymbol{w}^T\boldsymbol{x}$, which is a scalar representing the neuron's output, also sometimes referred to as the neuron's *pre-activation* due to often becoming the input to a non-linear activation function such as the rectified linear unit (ReLU) [38].

At a layer level, many neurons are organized together, and their weights form a weight matrix $\boldsymbol{W} \in \mathbb{R}^{D\times D'}$, where D' corresponds to the number of neurons in the layer. Additionally, a collection of N input token representations may be arranged in a matrix $\boldsymbol{X} \in \mathbb{R}^{D\times N}$. Then, the layer's output computation can be expressed compactly as $\boldsymbol{Y} = \boldsymbol{W}^T\boldsymbol{X}$, where the output $\boldsymbol{Y} \in \mathbb{R}^{D'\times N}$ is a matrix, and each element of $\boldsymbol{Y}$ corresponds to the output of a single neuron for a single token. Typically, a **transformer layer** is comprised of a set of weight matrices, each one being responsible for supporting a different mechanism, such as the self-attention mechanism or the transformation of the feed-forward network. Still, we will be referring to these individual weight matrices as **layers** due to stacking a collection of neurons. In most cases, we will be mentioning transformer layers explicitly, but occasionally whether the term layer refers to them can be inferred from context.

C. BERT-BASED ARCHITECTURES

The standard Bidirectional Encoder Representations from Transformers (BERT) model is an encoder-only architecture composed of 12 identical Transformer blocks, all following the description given in Section III-A. BERT pre-training involves bi-directional mask-language modeling, i.e. training the network to predict masked tokens by conditioning both on the left and on the right context of the masked token, and additionally performing next sentence prediction, i.e. to predict whether a second sentence is a natural continuation of the first one. This pre-training scheme allows BERT to excel in NLU tasks with subsequent fine-tuning and a minimal computation budget.

DistilBERT is essentially a compact version of BERT, successfully distilled down to 6 layers half the size of the original resulting in 40% fewer parameters. The trick is knowledge distillation: the small student model is trained not just on text, but also on the outputs of the larger BERT teacher, which allows it to retain a remarkable $\approx 97\%$ of the original model's language understanding capability.

MobileBERT was designed specifically for speed and efficiency on mobile devices. Its unique structure uses a bottleneck design by introducing an intermediate projection layer that significantly narrows the Transformer's hidden dimensions, coupled with factorized self-attention mechanisms. This innovative layer decomposition reduces the parameter count and latency dramatically, allowing it to maintain strong performance while being highly efficient.

TinyBERT adopts a more aggressive strategy for model compression, aiming to substantially reduce model size while preserving downstream performance. Its training relies on a comprehensive two-stage knowledge distillation framework. In the first stage, the student model learns from the teacher's internal representations by distilling information from intermediate Transformer layer outputs. In the second stage, it captures the teacher's behavior by distilling its attention distributions. By jointly aligning both hidden states and attention patterns with those of the teacher, TinyBERT achieves significant compression often up to 7.5× smaller than BERT base while maintaining strong task performance.

D. LOW-RANK APPROXIMATION FOR MODEL COMPRESSION

A common approach to reduce the size of large neural networks is via SVD and low-rank approximation. A weight matrix $\boldsymbol{W} \in \mathbb{R}^{m\times n}$ can be approximated as $\boldsymbol{W}' = \boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r^T$, where $\boldsymbol{U}_r \in \mathbb{R}^{m\times r}$ and $\boldsymbol{V}_r \in \mathbb{R}^{n\times r}$ are truncated orthogonal matrices, $\boldsymbol{\Sigma}_r \in \mathbb{R}^{r\times r}$ is a diagonal matrix of the top r singular values, and $r \leq \text{rank}(W)$. The singular values in $\boldsymbol{\Sigma}_r$, ordered from largest to smallest, represent the contribution of the corresponding components of $\boldsymbol{U}_r$ and $\boldsymbol{V}_r$ in the original matrix $\boldsymbol{W}$.

This technique is used to compress the large, weight matrices found in models like BERT. Instead of storing the full original matrix $\boldsymbol{W} \in \mathbb{R}^{m\times n}$, we can store two smaller matrices, $\boldsymbol{A} \in \mathbb{R}^{m\times r}$ and $\boldsymbol{B} \in \mathbb{R}^{r\times n}$, which, when multiplied, approximate the original matrix ($\boldsymbol{W} \approx \boldsymbol{AB}$). This can be done by simply setting $\boldsymbol{A} = \boldsymbol{U}_r\sqrt{\boldsymbol{\Sigma}_r}$ and $\boldsymbol{B} = \sqrt{\boldsymbol{\Sigma}_r}\boldsymbol{V}_r^T$, with $\sqrt{\boldsymbol{\Sigma}_r}$ denoting the matrix with all elements equal to the square root of $\boldsymbol{\Sigma}_r$.

According to our definition, the original model's matrix $\boldsymbol{W}$ has mn parameters. The total number of parameters in $\boldsymbol{A}$ and $\boldsymbol{B}$ is $mr + rn = r(m + n)$, where r is the chosen rank for the approximation. For effective compression, the new parameter count must be less than the original. This condition is met when: $mr + rn < mn$. This concludes that r must be chosen such that $r < (mn)/(m + n)$. A smaller r decreases parameter count and speeds up computations, but increases approximation error, which can affect performance.

The foundational method for low-rank approximation within the Transformer architecture is truncated SVD (tSVD), which is deterministic. tSVD works by isolating and keeping only the most important singular components (vectors and values) while discarding the rest. The complexity of tSVD is managed by utilizing specialized Krylov-based algorithms [39] that focus computation exclusively on these dominant components, avoiding the cost of calculating the full matrix spectrum. An alternative approach is randimized SVD (rSVD) [40], which uses random projections to create a smaller, sampled version of the matrix, significantly reducing the computation time. While rSVD offers a notable speed advantage, our methodology relies on the deterministic tSVD because it provides the essential guarantee of optimality (Eckart-Young theorem) and numerical stability. This stability is crucial for zero-shot compression tasks.

E. FWSVD

Fisher-Weighted Singular Value Decomposition (FWSVD) extends low-rank approximation by integrating the concept of parameter importance. While the SVD method assumes that all parameters of a model's weight matrix $\boldsymbol{W}$ have the same importance to the matrix reconstruction error, FWSVD leverages the Fisher information matrix to quantify the importance of each parameter in $\boldsymbol{W}$ to the task loss, ensuring that the most critical components for model performance are better preserved during the compression process.

For a single parameter w_{ij} at the location (i, j) in a model's weight matrix $\boldsymbol{W}$, the Fisher Information, denoted as $\hat{I}_{w_{ij}}$, measures the amount of information that a dataset D provides about the parameter. This is computed as the sample average of the squared partial derivative of the task's loss function $L(d_i; w_{ij})$ with respect to w_{ij}, given by:

$$\hat{I}_{w_{ij}} = \frac{1}{|D|} \sum_{i=1}^{|D|} \left(\frac{\partial}{\partial w_{ij}} L(d_i; w_{ij}) \right)^2 \quad (1)$$

with d_i denoting the i-th sample of the dataset and $|D|$ its total number of samples.

The Fisher-Weighted approach changes the optimization objective from a generic mathematical one to a task-specific one. While the standard SVD objective is to find a low-rank approximation $\boldsymbol{W}'$ that minimizes the reconstruction error, expressed as $\min_{\boldsymbol{W}'} ||\boldsymbol{W} - \boldsymbol{W}'||_2$, the FWSVD objective is given by $\min_{\boldsymbol{W}'} ||\hat{\boldsymbol{I}}_w \circ (\boldsymbol{W} - \boldsymbol{W}')||_F$ with $\hat{\boldsymbol{I}}_w$ a matrix with elements $\hat{I}_{w_{ij}}$ and $\circ$ denoting the Hadamard product. However, this optimization problem does not have a closed-form solution. For this reason, the authors of [12] proposed an approximation: $\min_{\boldsymbol{W}'} ||\hat{\boldsymbol{I}}\boldsymbol{W} - \hat{\boldsymbol{I}}\boldsymbol{W}'||_F$, with $\hat{\boldsymbol{I}}$ a diagonal matrix whose diagonal element $\hat{I}_i$ in row i is equal to $\hat{I}_i = \sqrt{\sum_j \hat{I}_{w_{ij}}}$. In this way, each neuron i is assigned an importance weight $\hat{I}_i$ based on the Fisher information that the dataset provides regarding its parameters. The approximation holds due to the fact that whenever all elements in a row of $\boldsymbol{W}$ share the same importance, the Hadamard product can be written as a standard matrix product with a diagonal matrix.

F. SVD-LLMv2

The most prominent SVD-based work is SVD-LLMv2, which combines the concepts of data-aware low-rank approximation with a computationally efficient and effective rank allocation algorithm, an algorithm that assigns ranks to each one of the approximated matrices under a predefined parameter budget. Instead of minimizing the error on the weight matrix (as SVD and FWSVD do), the optimization objective of SVD-LLMv2 is defined on the layer's output pre-activations, given by $\min_{\boldsymbol{W}'} ||\boldsymbol{W}\boldsymbol{X} - \boldsymbol{W}'\boldsymbol{X}||_2$, with $\boldsymbol{X}$ denoting a matrix of token representations to be transformed by the layer's matrix $\boldsymbol{W}$. The SVD-LLMv2 approach, by focusing on the layer's functional behavior, has been shown to better preserve task performance.

Additionally, the algorithm for matrix approximation that was introduced by SVD-LLMv2 ensures that the error in the layer's output pre-activations can be directly predicted from the truncated singular values. This is a significant improvement over previous approaches [28], [29], which suffer from a sharp drop in performance when truncating the smallest singular values due to the fact that they are not directly related to the truncation error of the layer's pre-activations.

Rather than applying a uniform compression ratio to every layer, which can be inefficient given that the sensitivity of the network with respect to different layers may vary, SVD-LLMv2 considers a heterogeneous allocation. It first groups the weights by their type, regardless of transformer layer index (e.g $\boldsymbol{W}_K$, $\boldsymbol{W}_Q$, $\boldsymbol{W}_V$, $\boldsymbol{W}_a$, $\boldsymbol{W}_u$, $\boldsymbol{W}_d$, etc.). Then it assigns a target rank to each of the weight matrices in the group by using the error in the layer's pre-activations under uniform rank allocation as a proxy for the layer's sensitivity with respect to rank reduction. This approach considers the matrix sensitivity to rank reduction from a functional perspective, resulting in a better trade-off between model size and performance preservation. What makes this grouping strategy efficient is that matrices within the same group share the same target parameter sub-budget, i.e. a parameter sub-budget is defined for each group based on the overall target compression ratio, and subsequently this sub-budget is distributed across matrices within the group.

IV. MOTIVATION

A. COMBINING NEURON IMPORTANCE WITH DATA-AWARE LOW RANK APPROXIMATION

Summarizing the discussion of the previous Section, to compute the low-rank approximation of a weight matrix, on the one hand, FWSVD considers the effect of each parameter on the task loss, which we refer to as **parameter importance** (PI). On the other hand, SVD-LLMv2 considers the effect of the parameters on the layer's output pre-activations, aiming for a functional equivalence of the layer before and after the compression. In this work, we are inspired by both methods in order to explore an approach that combines ideas from both perspectives. In particular, instead of minimizing $\min_{\boldsymbol{W}'} ||\boldsymbol{W}\boldsymbol{X}-\boldsymbol{W}'\boldsymbol{X}||_F$ aiming for low reconstruction error on the layer's output, i.e. the functional equivalence objective of SVD-LLMv2, we also consider the effect of the layer's output pre-activation on the task loss. This leads to the following optimization objective $\min_{\boldsymbol{W}'} ||\hat{\boldsymbol{I}}_y \circ (\boldsymbol{W}\boldsymbol{X} - \boldsymbol{W}'\boldsymbol{X})||_F$, with $\hat{\boldsymbol{I}}_y$ denoting a matrix containing the importance of each neuron's output to the task loss.

Solving this optimization problem faces the same limitation as the original formulation of FWSVD, i.e. a lack of closed-form solution. For this reason, we also approximate the Hadamard product by considering standard multiplication with a diagonal matrix $\hat{\boldsymbol{I}}$ whose diagonal element in row i is equal to the overall importance of the i-th neuron to the task loss. The latter is computed by aggregating the neuron's importance over several samples in a calibration dataset. Thus, in our approach, we find $\boldsymbol{W}'$ that minimizes $||\hat{\boldsymbol{I}}\boldsymbol{W}\boldsymbol{X} - \hat{\boldsymbol{I}}\boldsymbol{W}'\boldsymbol{X}||_F$. A first intuitive choice for computing the diagonal elements of $\hat{\boldsymbol{I}}$ is to aggregate **parameter importances** for each neuron, as it is done in FWSVD. However, in the experiments, we provide statistically significant evidence that this is suboptimal, leading to compressed networks that perform worse than when using data-aware low-rank approximation alone. As we discuss in Section V-A, we were able to improve on that by considering a different strategy. This strategy is based on a **direct way** to measure **neuron importance** (NI) that does not rely on the **parameter importance** previously considered by FWSVD.

B. DYNAMIC RANK ALLOCATION ACROSS MATRICES

Each target compression ratio (CR) can be directly translated to a total parameter budget under which the compressed LLM should fit. In low-rank approximation, the number of parameters in a layer is directly controlled by the rank of the approximation. Thus, distributing the total available parameters to individual layers can be accomplished by deciding on the rank of the approximation for each weight matrix. Performing **rank allocation**, that is, distributing target matrix ranks under a fixed parameter budget to individual layers, is a combinatorial and practically intractable problem, as optimal allocation relies on exhaustive search.

Previous approaches either resorted to proportionally equivalent reduction of parameters across layers [12], [30] (which we call uniform allocation), or based their decision on algorithms that were inspired by studies in a reduced search space. For instance, [29] found that optimal rank allocation varies both with layer depth and the layer's functional type (i.e. if it corresponds to $\boldsymbol{W}_Q$, $\boldsymbol{W}_V$, $\boldsymbol{W}_a$, $\boldsymbol{W}_u$, etc.). Meanwhile, [28] found that compression distortions from lower layers may result in progressively accumulative errors towards the latter layers, implying that rank allocation should vary with depth and compressing lower layers should be more conservative compared to higher ones. Additionally, [13] focused on allocating ranks for layers in the same functional group, exploiting the fact that the matrices in the same group share the same size. Despite the reduction in the search space, the rank allocation algorithm of [28] is still computationally expensive and is driven by model performance instead of parameter budget. Even though the rank allocation algorithm of [13] is computationally efficient, adapting it to a different grouping strategy requires addressing the challenge of matrix size variability among matrices in the same group. Our approach addresses these challenges by building on the findings of [28], while adapting the computationally efficient algorithm of [13] to group layers by their depth index instead of their functional role.

V. PROPOSED METHOD

Our proposed hybrid compression methodology is a post-training, SVD-based pipeline that fuses key concepts from recent advancements to achieve more efficient and effective compression. As shown in Figure 1, our approach synthesizes data-aware low-rank approximation with the concept of Fisher-weighted neuron importance. We also introduce a heterogeneous rank allocation that improves upon previous methods by adapting compression on a per-layer basis. The following subsections detail the core components of our pipeline, outlining our complete process from the computation of neuron importance to the final data-aware low-rank approximation and rank allocation algorithm.

A. NEURON IMPORTANCE ESTIMATION

We propose to estimate neuron importance by computing the amount of Fisher Information that a dataset provides about the neuron's **output pre-activation**, **instead of the neuron's parameters**. Following the notation of Section IV-A, we calculate the element of $\hat{\boldsymbol{I}}_y$ at the position (i, j) for the k-th sample of the dataset d_k as:

$$\hat{I}_{y_{ij}} = \left(\frac{\partial}{\partial y_{ij}} L(y_{ij}(d_k))\right)^2 \tag{2}$$

with L the task loss function and y_{ij} a spatial element in the matrix of layer outputs: $\boldsymbol{Y} = \boldsymbol{W}^T\boldsymbol{X}$. We subsequently form the neuron importance matrix $\hat{\boldsymbol{I}}$, a diagonal matrix with its element in row i being equal to the importance of neuron i:

$$\hat{I}_i = \sqrt{\frac{1}{|D|}\sum_{k=1}^{|D|} \mathbb{E}_j\left[\hat{I}_{y_{ij}}|d_k\right]} \tag{3}$$

B. DATA-AWARE LOW RANK APPROXIMATION DRIVEN BY NEURON IMPORTANCE

We adapt the weight truncation algorithm of SVD-LLMv2 by integrating the neuron importance matrix $\hat{\boldsymbol{I}}$ of the previous section into the process of low-rank approximation. Algorithm 1 summarizes the process of decomposing a matrix $\boldsymbol{W}$ to its low-rank approximation $\boldsymbol{W}' = \boldsymbol{A}\boldsymbol{B}$ and is a generalization over the algorithm proposed by SVD-LLMv2, with the two algorithms being identical whenever the neuron importance matrix is equal to the identity matrix.

Algorithm 1 Weight Truncation Algorithm

```
Input: $W$: Original weight matrix
       $X$: Matrix of layer input activations
       $\hat{I}$: Diagonal matrix of neuron importances
       $R$: Target rank for the low-rank approximation
Output: $A$, $B$, $W'$: Low rank approximation factors $A$, $B$ and compressed weight matrix $W'$.
1:  procedure Weight_Truncation($W$, $X$, $I$, $R$)
2:      $S \leftarrow XX^T$    ▷ Construct matrix $S$ from $X$
3:      $U_s, S_s, V_s \leftarrow \text{SVD}(S)$    ▷ Perform SVD on matrix $S$
4:      $D \leftarrow \hat{I}WU_s\sqrt{S_s}$    ▷ Construct matrix $D$
5:      $U_{ws}, S_{ws}, V_{ws} \leftarrow \text{SVD}(D)$    ▷ Perform SVD on matrix $D$
6:      $U^t_{ws}, S^t_{ws}, V^t_{ws} \leftarrow \text{Truncate}(U_{ws}, S_{ws}, V_{ws}, R)$    ▷ Perform SVD truncation based on target rank approximation $R$
7:      $A \leftarrow \hat{I}^{-1}U^t_{ws}\sqrt{S^t_{ws}}$    ▷ The first matrix of low-rank approximation
8:      $B \leftarrow \sqrt{S^t_{ws}}V^t_{ws}\sqrt{S_s}^{-1}U_s^{-1}$    ▷ The second matrix of low-rank approximation
9:      $W' \leftarrow AB$    ▷ Reconstructed low-rank approximation $W'$
10:     return $A$, $B$, $W'$
11: end procedure
```

Let $L = ||\hat{I}WX - \hat{I}W'X||_F$ denote the weighted compression loss when approximating W with a low rank matrix W'. For any given rank of the approximation, the theoretical minimum loss is given by the error induced by the truncated SVD decomposition of $\hat{I}WX$, denoted as $||\mathbf{SVD}(\hat{I}WX)||_F$.

Theorem 1: If U_s, S_s, V_s are obtained by the SVD decomposition of XX^T and $U^t_{ws}, S^t_{ws}, V^t_{ws}$ are obtained by the truncated SVD decomposition of $\hat{I}WU_s\sqrt{S_s}$, the compressed weight matrix $W' = \hat{I}^{-1}U^t_{ws}S^t_{ws}V^t_{ws}\sqrt{S_s}^{-1}U_s^{-1}$ minimizes the weighted compression loss L.

Proof: Let U_x, S_x, V_x denote the matrices from the SVD decomposition of X. Since XX^T is symmetric, $U_s = V_s$. Moroever, $U_x = U_s$ and $S_x = \sqrt{S_s}$. Let also $C = U_s\sqrt{S_s}$ which implies that $C^{-1} = \sqrt{S_s}^{-1}U_s^{-1}$. It is easy to show that $C^{-1}X = V_x$ which is orthogonal and thus does not affect the Frobenius norm under matrix multiplication. For the proposed choice of W' the weighted compression loss L becomes:

$$
\begin{aligned}
L &= ||\hat{I}WX - \hat{I}W'X||_F \\
&= ||\hat{I}WX - \hat{I}\hat{I}^{-1}U^t_{ws}S^t_{ws}V^t_{ws}\sqrt{S_s}^{-1}U_s^{-1}X||_F \\
&= ||\hat{I}WCC^{-1}X - U^t_{ws}S^t_{ws}V^t_{ws}C^{-1}X||_F \\
&= ||(\hat{I}WC - U^t_{ws}S^t_{ws}V^t_{ws})C^{-1}X||_F \\
&= ||(\hat{I}WC - U^t_{ws}S^t_{ws}V^t_{ws})||_F \\
&= ||\mathbf{SVD}(\hat{I}WC)||_F \\
&= ||\mathbf{SVD}(\hat{I}WU_s\sqrt{S_s})||_F \\
&= ||\mathbf{SVD}(\hat{I}WU_xS_x)||_F \\
&= ||\mathbf{SVD}(\hat{I}WU_xS_xV_x)||_F \\
&= ||\mathbf{SVD}(\hat{I}WX)||_F
\end{aligned}
$$

which is equal to the theoretical minimum. □

Algorithm 1 computes W' based on Theorem 1 and suggests a low rank matrix approximation that minimizes the weighted compression loss.

C. DYNAMIC RANK ALLOCATION

Following Section III-F, and based on the findings of [28] we propose an adaptation of the rank allocation algorithm of [13] by changing the grouping strategy of the weight matrices to be based on transformer layer index (i.e. layer's depth in the transformer's sequence of layers). This change in the grouping implies that a target parameter sub-budget is defined for all matrices within a transformer layer, instead of matrices with the same functional role. The algorithm still distributes this sub-budget to each matrix in the group based on matrix sensitivity to rank reduction, as it was proposed in [13].

Our layer index-based rank allocation method groups all weight types (e.g. W_Q, W_K, W_V, W_a, W_u and W_d) with the same transformer layer index l, by treating each one of the transformer layers in the BERT-family models as a distinct group. This layer-wise approach modifies the algorithm to derive a compression ratio r for each weight matrix and ensures iterative refinement for optimal balance.

Algorithm 2 summarizes our layer index-based rank allocation strategy. The algorithm requires as an input the original LLM, a representative set of input activations resulting from the calibration dataset, and the effective compression ratio. The effective compression ratio is calculated to account for parameters non amenable to compression, possibly due to the high sensitivity of the network with respect to these parameters, while still aiming for a specific overall target compression ratio. The algorithm first groups the model's weights by their layer (line 2) and then computes a normalized theoretical loss (lines 6-10), which serves as a proxy for that layer's sensitivity to rank reduction. For a given matrix W, the theoretical loss function first translates the target compression ratio (R) into the corresponding retained SVD rank (k) to compute its low-rank approximation W'. It then uses W' to calculate the error on the layer's output pre-activations $L_{\min} = \|WX - W'X\|_F$, indicated as **theoretical loss**. Compression ratios are then distributed proportionally to these normalized errors (lines 12-15), ensuring that layers deemed more sensitive to rank reduction receive higher ranks. Line 18 performs the necessary transformation from the calculated proportional ratios (R_d) into the specific, non-zero integer SVD ranks (k) required for each weight matrix W in the model (M). To further meet the consistency with the global compression target, the iterative refinement loop (lines 23-34) adjusts matrix ranks to ensure that the desired ratio is achieved. The update rule (lines 28-29) ensures

Algorithm 2 Layer Index-Based Rank Allocation Algorithm

```
Input: $M$: Original LLM
       $X$: Input activations
       $R$: Effective target compression ratio
Output: $R_d$: Compression rank allocation per layer (list of lists)
1:  procedure Rank_Allocation($M, X, R$)
2:      $G \leftarrow$ Group($M$)                    ▷ Group weights by layer index
3:      $R_d \leftarrow \emptyset$                  ▷ Initialize the compression ratio list
4:      for $g$ in $G$ do
5:          $L_G \leftarrow \emptyset$              ▷ Initialize the loss list in the group
6:          for $W$ in $g$ do
7:              $L_{min} \leftarrow$ Theoretical_Loss($W, X, R$) ▷ Compute per weight theoretical loss
8:              $L_G \leftarrow L_G \cup L_{min}$   ▷ Append loss to list
9:          end for
10:         $L_G \leftarrow 1/\text{Log}(L_G)$      ▷ Normalize $L_G$
11:         $layer\_ratios \leftarrow \emptyset$    ▷ Initialize layer compression ratios list
12:         for $L_{min}$ in $L_G$ do
13:             $r \leftarrow \text{Len}(L_G) \times R \times L_{min}/\text{Sum}(L_G)$ ▷ Allocate ratios proportionally to normalized loss
14:             $layer\_ratios \leftarrow layer\_ratios \cup r$ ▷ Append ratio to list
15:         end for
16:         $R_d \leftarrow R_d \cup layer\_ratios$ ▷ Append $layer\_ratios$ list to $Rd$ list
17:     end for
18:     $R_d \leftarrow$ Convert_Ratios_to_Ranks($M, R_d$) ▷ Convert ratios to ranks for each weight
19:     $total\_params \leftarrow$ Count_Total_Parameters($M$) ▷ Count all the parameters of the model
20:     $comp\_params \leftarrow$ Count_Compressed_Parameters($M, R_d$) ▷ Count all the parameters of the compressed model
21:     $achieved\_ratio \leftarrow 1 - comp\_params/total\_params$) ▷ Compute achieved ratio after rank allocation
22:     $i \leftarrow 0$
23:     while $|achieved\_ratio - R| > \varepsilon$ do
24:         $sign \leftarrow achieved\_ratio - R$
25:         $scale \leftarrow (1 - R)/(1 - achieved\_ratio)$ ▷ Compute scaling factor
26:         for $layer\_ranks$ in $R_d$ do
27:             for $r$ in $layer\_ranks$ do
28:                 $r \leftarrow r + sign \cdot i$ ▷ Iteratively adjust rank based on error direction
29:                 $R_d \leftarrow$ Update($R_d, r$) ▷ Update rank allocation list
30:             end for
31:         end for
32:         $i \leftarrow i + 1$
33:     end while
34:     return $R_d$
35: end procedure
```

convergence by increasing/decreasing matrix ranks whenever the compressed model is under/over the target parameter budget. This dynamic adjustment makes the allocation procedure adaptive and computationally efficient, avoiding exhaustive search strategies.

Overall, our layer index-based rank allocation algorithm balances two objectives: (i) preserving the most critical layers through proportional loss-aware rank allocation, and (ii) meeting the global compression constraint through iterative refinement.

VI. EXPERIMENTAL RESULTS

A. EXPERIMENTAL SETUP

We evaluate the effectiveness of our method by compressing the standard 12-layer BERT transformer architecture. We consider BERT, fine-tuned on 8 datasets of the GLUE-Benchmark [41], namely *cola, mnli-m, mnli-mm, mrpc, qnli, qqp, sst-2* and *sts-b*. Each dataset corresponds to a different task, and the performance of the (compressed) models is evaluated under a task-specific metric. Single sentence tasks, *cola* and *sst-2* are measured by Matthew's correlation and classification accuracy, respectively. The sentence similarity tasks *mrpc* and *qqp* are assessed by F-1 score, while *sts-b*, using Pearson-Spearman correlation. Finally, natural language inference tasks *mnli-m*, *mnli-mm* and *qnli* are measured by accuracy.

For each one of the layers in the BERT architecture, the matrices that we consider for compression are the ones discussed in Section III-A, namely $\boldsymbol{W}_K$, $\boldsymbol{W}_Q$, $\boldsymbol{W}_V$, $\boldsymbol{W}_a$, $\boldsymbol{W}_u$, $\boldsymbol{W}_d$. We do not consider compressing the token embedding matrix, nor the matrices involved in the layer-normalization blocks. To compute the uncentered covariance matrix $\boldsymbol{S}$ in Algorithm 1, we randomly sample a (balanced) dataset of 256 text samples from the dataset's training split. We also compute neuron importances (NIs) using the full training split of the dataset, as it is done in FWSVD for parameter importances (PIs). In contrast to FWSVD or SVD-LLMv2 which additionally consider fine-tuning the compressed models to recover performance, we emphasize results and comparisons without a fine-tuning step. Even though fine-tuning may boost the results of any previous SVD-based method, including ours, we want to highlight the benefits of the proposed approach, namely Neuron Importance driven Data-Aware SVD (NIDA-SVD), under a low computation budget which does not involve finetuning. Nevertheless, we report results with fine-tuning when compressing MobileBERT and TinyBERT.

The experimental setup is extended to include the structurally similar DistilBERT, as well as the specialized MobileBERT and TinyBERT architectures. Since these models are direct descendants of BERT, they are also fine-tuned and evaluated on specific subsets of the GLUE Benchmark using the corresponding task-specific metrics described above. Specifically, the models are evaluated on the following subsets: DistilBERT on *mnli-mm*, *mrpc*, *qnli*, *qqp*, and *sst2*, MobileBERT on *mrpc*, *qnli*, *qqp*, *sst2*, and *stsb*, and TinyBERT on *mnli-mm*, *mrpc*, *qnli*, *qqp*, *sst2*, and *stsb*. For all models, compression is applied to their Transformer encoder blocks and dense layers. The methodology for computing the matrix $\boldsymbol{S}$ and the neuron importance $\boldsymbol{I}$ remains identical for all models.

In the following subsections, we present experimental results comparing our method with the recent state-of-the-art in low-rank approximation: SVD, FWSVD, and SVD-LLMv2, and conduct ablation studies under different strategies regarding parameter importance, neuron importance, and rank allocation. Since the algorithm of SVD-LLMv2

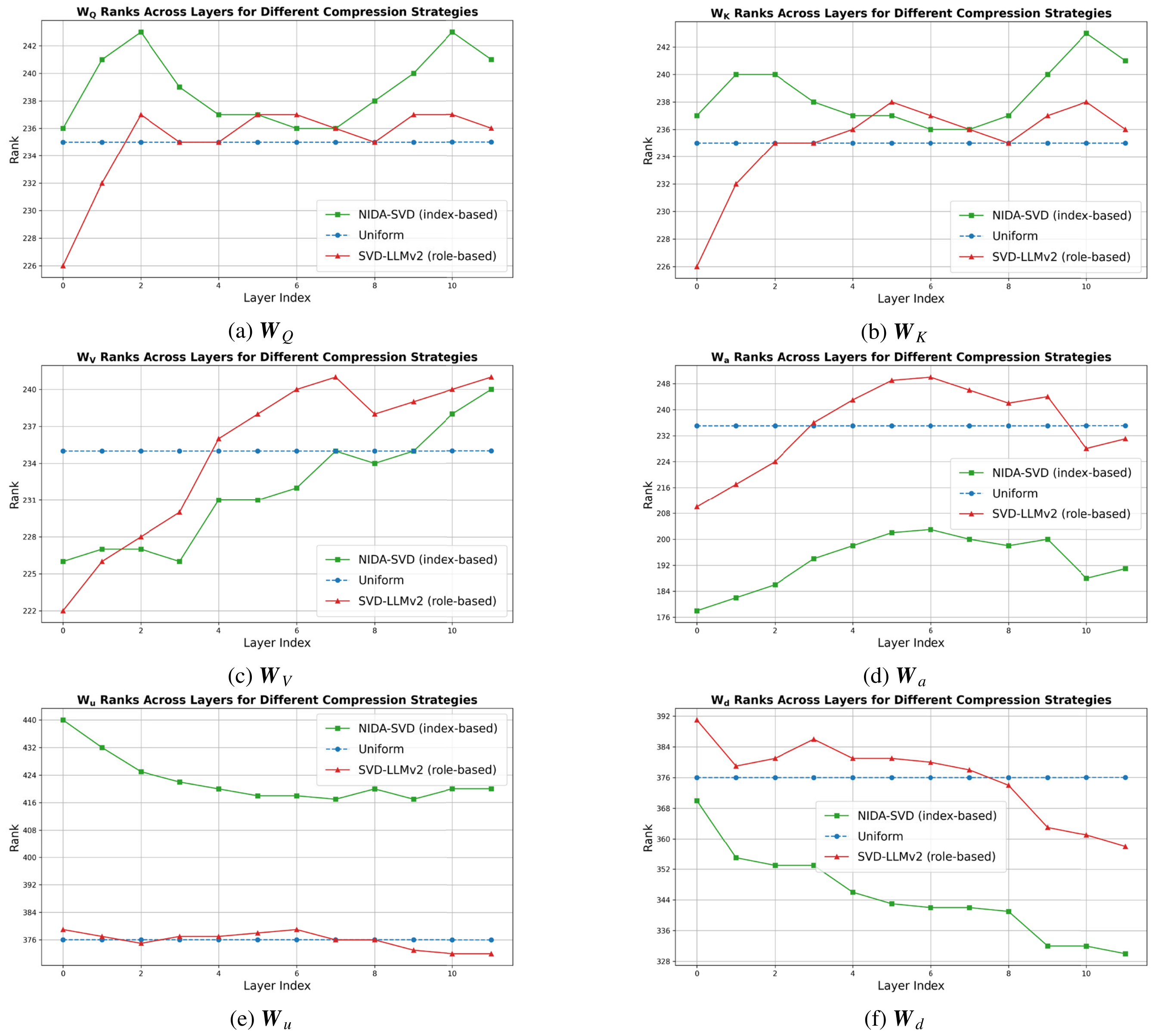


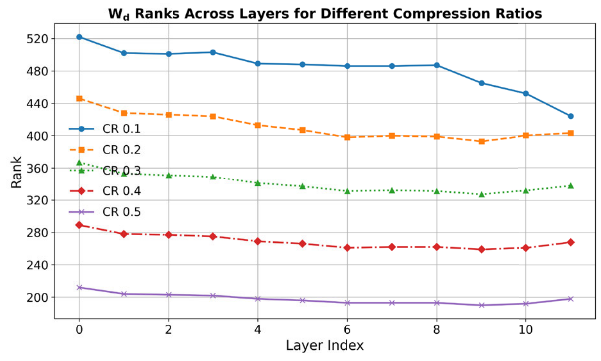
FIGURE 2. Visualization of BERT model ranks under 30% compression on QNLI dataset, comparing SVD-LLMv2 (role-based), Uniform, and NIDA-SVD strategies: (a) Query, (b) Key, (c) Value, (d) Attention, (e) Up, (f) Down.

performs rank allocation based on the layer's functional role in the transformer block, we call it *role-based allocation*. Contrariwise, our algorithm approaches rank allocation from the perspective of the layer's depth index, and thus we refer to it as *index-based allocation*.

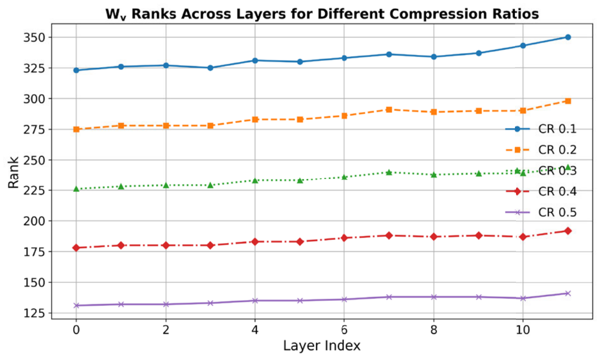
B. COMPARISON AGAINST THE STATE-OF-THE-ART

Table 1 summarizes performance metrics for the BERT model compressed under different Compression Ratios (CR) with a uniform rank allocation strategy. Overall, in 35 out of 40 cases (87.5%), our method is ranked first; the compressed model retains most of its task performance when compressed with our method. In the rest of the cases, our method is ranked second, remaining competent to the top-performing method, which is always SVD-LLMv2. We emphasize the improvement that our method offers compared to the previous state-of-the-art, especially in the highest compression rate (0.5): +3.15% **absolute improvement** in *cola*, +4.16% in *mnli-m*, +4.82% in *mnli-mm*, +1.54% in *mrpc*, +2.48% in *qnli*, +2.35% in *qqp*, and +2.33% in *stsb*.

Similarly, Table 2 summarizes performance metrics for the BERT model compressed either with our method or the previous state-of-the-art, SVD-LLMv2. In this comparison, we consider each method to use its own rank allocation algorithm: *role-based* allocation for SVD-LLMv2 and *index-based* allocation for the proposed method. For the extended experiments presented below, Table 3 (DistilBERT), Table 4 (MobileBERT), and Table 5 (TinyBERT), each utilized its own rank allocation strategy (role-based for SVD-LLMv2 and index-based for NIDA-SVD). For BERT,



(a) $\boldsymbol{W}_d$



(b) $\boldsymbol{W}_V$

FIGURE 3. **Visualization of BERT model weight ranks across different compression ratios (layer-based strategy) on the MRPC dataset: (a) Down, (b) Value.**

TABLE 1. **Compressed model performance metrics for different compression ratios (CR) on the BERT base model under uniform rank allocation. In 35 out of 40 cases (87.5%), our method (NIDA-SVD) achieves state-of-the-art performance. The best results are in bold, while the second-best are underlined.**

CR	Method	Rank Allocation	cola	mnli-m	mnli-mm	mrpc	qnli	qqp	sst2	stsb
N/A	BERT base	N/A	0.562	0.847	0.849	0.906	0.916	0.88	0.923	0.891
0.1	SVD	uniform	0.2372	0.7745	0.7363	0.3693	0.7151	0.8043	0.8864	0.8028
	FWSVD		0.3488	0.8289	0.8148	0.8877	0.8936	0.8667	0.8933	0.8677
	SVD-LLMv2		0.5548	0.8407	**0.8423**	0.9075	0.9115	0.8740	**0.9174**	0.8838
	NIDA-SVD		**0.5600**	**0.8434**	0.8418	**0.9157**	**0.9119**	**0.8747**	0.9162	**0.8846**
0.2	SVD	uniform	0.1010	0.6744	0.6183	0.0282	0.5663	0.7145	0.8761	0.7025
	FWSVD		0.0657	0.7987	0.7526	0.8675	0.8500	0.8359	0.8761	0.8375
	SVD-LLMv2		**0.5254**	0.8349	0.8335	0.9071	0.9009	0.8652	0.9151	0.8729
	NIDA-SVD		0.5094	**0.8383**	**0.8352**	**0.9128**	**0.9068**	**0.8684**	**0.9185**	**0.8740**
0.3	SVD	uniform	0.0358	0.4895	0.4284	0.0000	0.4980	0.6083	0.8245	0.5043
	FWSVD		0.0270	0.7065	0.6275	0.8320	0.6760	0.7488	0.8360	0.7871
	SVD-LLMv2		**0.4543**	0.8210	0.8152	0.8863	0.8755	0.8514	0.9025	0.8239
	NIDA-SVD		0.4451	**0.8245**	**0.8203**	**0.9003**	**0.8898**	**0.8561**	**0.9048**	**0.8420**
0.4	SVD	uniform	-0.0028	0.3850	0.3614	0.0000	0.4946	0.5764	0.6559	0.4276
	FWSVD		0.0207	0.5562	0.5152	0.6867	0.4978	0.6952	0.7213	0.7102
	SVD-LLMv2		0.3206	0.7766	0.7691	0.8376	0.8088	0.8197	0.8876	0.7295
	NIDA-SVD		**0.3271**	**0.7901**	**0.7851**	**0.8657**	**0.8359**	**0.8322**	**0.8979**	**0.7652**
0.5	SVD	uniform	-0.0252	0.3713	0.3703	0.0000	0.4946	0.5743	0.5126	0.2265
	FWSVD		-0.0172	0.3541	0.4778	0.0000	0.4949	0.4345	0.6032	0.5981
	SVD-LLMv2		0.2079	0.6727	0.6678	0.8176	0.6875	0.7602	**0.8772**	0.6426
	NIDA-SVD		**0.2394**	**0.7143**	**0.7160**	**0.8330**	**0.7113**	**0.7837**	0.8704	**0.6659**

In 35 out of 40 cases (87.5%), our method outperforms SVD-LLMv2 while remaining comparable to SVD-LLMv2 in the remaining 5. In this case, under the highest compression ratio (0.5), our method makes an **absolute improvement** of +4.41% in *cola*, +4.94% in *mnli-m*, +5.01% in *mnli-mm*, +1.82% in *mrpc*, +4.14% in *qnli*, +1.92% in *qqp* and +6.41% in *stsb*.

Table 3 presents the DistilBERT results, comparing our NIDA-SVD against the SVD-LLMv2. Our method shows a clear performance advantage, achieving the best result in 23 out of 25 comparisons (92%). This advantage is especially pronounced at the compression ratio of 0.3, where NIDA-SVD delivers significant absolute improvements, including +0.90% in *mnli-mm* and +2.43% in *qnli*.

The MobileBERT results, detailed in Table 4 , compare our NIDA-SVD approach against the SVD-LLMv2. Our methodology demonstrates a clear performance advantage in 21 of the 25 total comparisons (84%). This general advantage is mostly observed across 0.3 compression ratio, including *qqp* (+3.68%), *stsb* (+2.03%), and *sst2* (+1.03%).

The compression performance on the highly reduced TinyBERT architecture is documented in Table 5 . Comparing NIDA-SVD against the SVD-LLMv2, NIDA-SVD exhibits a robust performance advantage, securing the superior metric in 17 out of 18 total comparisons (94.4%). The most substantial performance gains are concentrated at the compression ratio of 0.2. At this level, NIDA-SVD registers notable absolute increases, specifically +1.43% in *qnli*, +0.98% in *mnli-mm*, and +0.54% in *qqp*, confirming the efficacy of the data-aware approach even on extremely compact models. Note that TinyBERT, being a highly distilled and compact model (14.3M parameters), was only compressed up to a compression ratio of 0.3 (70% parameter reduction), as pushing for a higher compression ratio would likely lead to catastrophic performance collapse due to its limited architectural redundancy.

While the primary focus of this work is post-training compression, it is informative to examine how well the performance of highly compressed models is recovered by subsequent fine-tuning. Table 6 presents the results after a single fine-tuning epoch on both MobileBERT and

TABLE 2. **Comparison of our method (NIDA-SVD) against the previous state-of-the-art (SVD-LLMv2) for BERT base model under different compression ratios (CR). Each method is complemented with its own rank allocation algorithm. In 34 out of 40 cases (85%), our method outperforms SVD-LLMv2, especially in high compression ratios. The best results are in bold.**

CR	Method	Rank Allocation	cola	mnli-m	mnli-mm	mrpc	qnli	qqp	sst2	stsb
N/A	BERT base	N/A	0.562	0.847	0.849	0.906	0.916	0.88	0.923	0.891
0.1	SVD-LLMv2	role-based	**0.5573**	0.8419	0.8429	**0.9087**	**0.9126**	0.8736	0.9139	0.8841
	NIDA-SVD	index-based	0.5522	**0.8456**	**0.8444**	0.9078	0.9121	**0.8739**	**0.9162**	**0.8866**
0.2	SVD-LLMv2	role-based	**0.5325**	0.8369	0.8340	**0.9134**	0.9002	0.8657	0.9110	0.8722
	NIDA-SVD	index-based	0.5147	**0.8396**	**0.8387**	0.9090	**0.9068**	**0.8690**	**0.9116**	**0.8793**
0.3	SVD-LLMv2	role-based	0.4556	0.8206	0.8171	0.8907	0.8766	0.8531	0.9002	0.8242
	NIDA-SVD	index-based	**0.4736**	**0.8283**	**0.8224**	**0.9026**	**0.8923**	**0.8576**	**0.9013**	**0.8608**
0.4	SVD-LLMv2	role-based	0.3276	0.7802	0.7715	0.8366	0.8103	0.8199	0.8944	0.7231
	NIDA-SVD	index-based	**0.3486**	**0.7954**	**0.7907**	**0.8708**	**0.8458**	**0.8345**	**0.8956**	**0.8031**
0.5	SVD-LLMv2	role-based	0.2039	0.6762	0.6750	0.8176	0.6917	0.7698	**0.8727**	0.6435
	NIDA-SVD	index-based	**0.2480**	**0.7255**	**0.7251**	**0.8358**	**0.7331**	**0.7890**	0.8704	**0.7076**

TABLE 3. **Comparison of our method (NIDA-SVD) against the previous state-of-the-art (SVD-LLMv2) for DistilBERT base model under different compression ratios (CR). Each method is complemented with its own rank allocation algorithm. In 23 out of 25 cases (92%), our method outperforms SVD-LLMv2, especially in high compression ratios. The best results are in bold.**

CR	Method	Rank Allocation	mnli-mm	mrpc	qnli	qqp	sst2
N/A	DistilBERT base	N/A	0.8232	0.8892	0.8874	0.8643	0.9128
0.1	SVD-LLMv2	role-based	0.8147	0.8851	0.8720	0.8569	0.9101
	NIDA-SVD	index-based	**0.8172**	**0.8877**	**0.8724**	**0.8579**	**0.9105**
0.2	SVD-LLMv2	role-based	0.8055	0.8866	0.8420	0.8490	**0.9092**
	NIDA-SVD	index-based	**0.8088**	**0.8870**	**0.8561**	**0.8497**	0.9082
0.3	SVD-LLMv2	role-based	0.7742	0.8756	0.7761	0.8317	0.9036
	NIDA-SVD	index-based	**0.7832**	**0.8766**	**0.8004**	**0.8320**	**0.9059**
0.4	SVD-LLMv2	role-based	0.7016	0.8396	0.6421	0.7591	0.8841
	NIDA-SVD	index-based	**0.7132**	**0.8467**	**0.6615**	**0.7694**	**0.8876**
0.5	SVD-LLMv2	role-based	0.5592	0.8040	0.5335	**0.5101**	0.8463
	NIDA-SVD	index-based	**0.5727**	**0.8053**	**0.5387**	0.4919	**0.8532**

TABLE 4. **Comparison of our method (NIDA-SVD) against the previous state-of-the-art (SVD-LLMv2) for MobileBERT base model under different compression ratios (CR). Each method is complemented with its own rank allocation algorithm. In 22 out of 25 cases (88%), our method outperforms SVD-LLMv2, especially in high compression ratios. The best results are in bold.**

CR	Method	Rank Allocation	mrpc	qnli	qqp	sst2	stsb
N/A	MobileBERT base	N/A	0.8888	0.9068	0.8670	0.9128	0.8773
0.1	SVD-LLMv2	role-based	**0.8625**	0.8224	0.7856	0.8830	0.8191
	NIDA-SVD	index-based	0.8603	**0.8506**	**0.8042**	**0.8841**	**0.8452**
0.2	SVD-LLMv2	role-based	0.8277	0.7940	0.7437	0.8337	0.7741
	NIDA-SVD	index-based	**0.8290**	**0.8065**	**0.7664**	**0.8451**	**0.7972**
0.3	SVD-LLMv2	role-based	0.8218	0.7298	0.6971	0.8027	0.7570
	NIDA-SVD	index-based	**0.8284**	**0.7346**	**0.7339**	**0.8130**	**0.7773**
0.4	SVD-LLMv2	role-based	0.8183	0.5976	0.6750	0.7717	0.6691
	NIDA-SVD	index-based	**0.8210**	**0.5991**	**0.6782**	**0.7740**	**0.7053**
0.5	SVD-LLMv2	role-based	0.6895	0.5081	**0.5834**	0.7419	**0.4664**
	NIDA-SVD	index-based	**0.6919**	**0.5092**	0.5735	**0.7488**	0.3867

TABLE 5. **Comparison of our method (NIDA-SVD) against the previous state-of-the-art (SVD-LLMv2) for TinyBERT model under different compression ratios (CR). Each method is complemented with its own rank allocation algorithm. In 17 out of 18 cases (94%), our method outperforms SVD-LLMv2, especially in high compression ratios. The best results are in bold.**

CR	Method	Rank Allocation	mnli-mm	mrpc	qnli	qqp	sst2	stsb
N/A	TinyBERT base	N/A	0.7955	0.8881	0.837	0.8512	0.8704	0.8731
0.1	SVD-LLMv2	role-based	0.7759	0.8722	0.8107	0.8405	0.8612	0.8647
	NIDA-SVD	index-based	**0.778**	**0.8736**	**0.8125**	**0.8421**	**0.8646**	**0.8672**
0.2	SVD-LLMv2	role-based	0.6456	0.8278	0.6761	0.7837	**0.8486**	0.8202
	NIDA-SVD	index-based	**0.6554**	**0.8291**	**0.6904**	**0.7891**	0.8451	**0.8233**
0.3	SVD-LLMv2	role-based	0.3721	0.8122	0.5215	0.5406	0.5676	0.1146
	NIDA-SVD	index-based	**0.3802**	**0.8122**	**0.5216**	**0.5422**	**0.5745**	**0.115**

TinyBERT, compressed at 0.5 and 0.3 compression ratio, respectively. Even after introducing the fine-tuning step, NIDA-SVD generally maintains better results than SVD-LLMv2. For MobileBERT, NIDA-SVD registers meaningful gains in *qnli* and *sst2* (both achieving an approximate +1.26% improvement over SVD-LLMv2). Similarly, on TinyBERT, NIDA-SVD captures an advantage in the inference tasks (*mrpc* and *qnli*), with the *qnli* metric recovering +0.92%

TABLE 6. Single-epoch fine-tuning comparison of NIDA-SVD and SVD-LLMv2 on MobileBERT at 0.5 compression ratio and TinyBERT at 0.3 compression ratio. The results (F1/Accuracy on MRPC, QNLI, SST-2) show that fine-tuning after NIDA-SVD compression achieves better accuracy in 4 out of 6 cases (66.6%), demonstrating robust performance despite the substantial compression level.

CR	Method	mrpc	qnli	sst2
N/A	MobileBERT	0.8888	0.9068	0.9036
0.5	SVD-LLMv2	**0.8145**	0.8281	0.8543
	NIDA-SVD	0.8096	**0.8407**	**0.8669**
N/A	TinyBERT	0.8881	0.837	0.8704
0.3	SVD-LLMv2	0.816	0.6148	**0.8325**
	NIDA-SVD	**0.8222**	**0.624**	0.8176

better than the baseline. Overall, across the six task-model pairs evaluated, the NIDA-SVD compressed models yield better accuracy in four instances, underscoring the robustness of our data-aware, importance-driven initialization.

C. ABLATION STUDIES

In Table 7 we provide experimental evidence that under uniform rank allocation, naively combining FWSVD with SVD-LLMv2 (NIDA-SVD (PI)) in 26 out of the 40 cases (more than half of them), the compressed model's task performance is inferior compared to when not using PI at all (in which case the algorithm is equivalent to vanilla SVD-LLMv2). We note that NIDA-SVD (PI) is equivalent to Algorithm 1 but with $\hat{\boldsymbol{I}}$ computed as discussed in Section III-E, whereas in NIDA-SVD (NI), $\hat{\boldsymbol{I}}$ is computed as discussed in Section V-A. Overall, in 38 out of 40 cases (95%), NI is more effective than PI, and in 35 it improves upon SVD-LLMv2, while remaining competent in the rest of them.

Tables 8 and 9 summarize experimental results regarding the efficacy of our proposed index-based rank allocation, compared to other strategies that were previously considered. The experiments show that regardless of the compression algorithm being SVD-LLMv2 or NIDA-SVD, index-based allocation is a better choice for the majority of the cases (more than 67.5% of them). Furthermore, a row-wise comparison of Tables 8 and 9 reveals that, in most cases, NIDA-SVD outperforms SVD-LLMv2 under any rank allocation algorithm. While for uniform rank allocation this was discussed in the previous section, for role-based allocation, NIDA-SVD outperforms SVD-LLMv2 in 34 cases (85%) while for index-based allocation in 33 (82.5%). In many cases the improvement is substantial, especially at high compression ratios: by up to +4.54% **absolute improvement** in role-based rank allocation and +7.97% for index-based rank allocation.

D. RANK ALLOCATION ANALYSIS

The rank allocation analysis indicates that the index-based strategy (NIDA-SVD) offers advantages over the role-based approach (SVD-LLMv2), mainly because the role-based method handles matrix ranks in a more uniform manner across all transformer layers (especially in Figure 2 (a),(b) and (e), resulting in a flatter rank profile that fails to capture the intrinsic heterogeneity in network's sensitivity among layers. This results in role-based ranks being consistently closer to uniform values, reflecting a more uniform compression approach across the network. In contrast, the index-based strategy, as detailed in Section V-C and Algorithm 2, groups all compressible weight types by individual layer index, allowing for a more targeted distribution of ranks that adaptively varies according to the specific redundancy and importance of each layer. This approach enables the index-based method to exploit layer-specific redundancies more effectively, as demonstrated in the plots of Figure 2 (a)-(f). Particularly in Figure 2 (e), the role-based ranks align much more closely with the uniform values when compared to the index-based ranks.

In Figure 2 (a) and (b), the two images illustrate the rank allocations for the down projection and value weight matrices across the 12 layers of the BERT model under varying compression ratios (0.1 to 0.5) using the index-based strategy. For all compression ratios, the behavior of ranks within the same weight type follows a consistent pattern. Specifically, for the $\boldsymbol{W}_d$ (Figure 3 (a)), the earlier layers (e.g. 0-5) are more sensitive to compression, requiring more parameters to maintain performance, in contrast to the later layers (e.g. 6-11). Also, for the $\boldsymbol{W}_V$ (Figure 3 (b)), the earlier layers are less sensitive to compression than the later layers. As we can see, there are different patterns for different weight types across the layers.

Generally, we cannot assume that a weight will exhibit a specific behavior in rank compression across layers, as the sensitivity to compression can vary depending on the weight type and its index within the model. Our rank allocation algorithm identifies these distinct patterns by analyzing the intrinsic redundancy and importance of each layer, dynamically adjusting the ranks to optimize performance.

E. MEMORY AND COMPUTATIONAL ANALYSIS

The primary motivation for model compression is to minimize memory and inference time, especially for deployment on resource-constrained devices. To quantify the advances achieved by our low-rank approximation methods, we conducted a complexity analysis measuring both the memory requirements, represented by total parameters (in millions), and the computational cost, measured in MFLOPS/token. This analysis conducted in DistilBERT and MobileBERT for comrpession ratio 0.5, and TinyBERT for compression ratio 0.3, comparing the uncompressed base models against the two SVD-based (SVD-LLMv2 and NIDA-SVD) compression approaches.

The results presented in Table 10 clearly demonstrate the reduction in model size. For DistilBERT, compression reduces the total parameters from 66.9 million to approximately 33.4 million, achieving the targeted 50 reduction. Similarly the memory size of the MobileBERT reduced from 24.5 million to about 12.1 million. This parameter reduction directly translated to a massive decrease in computational cost, as the MFLOPS/token for DistilBERT drops from 86 MFLOPS to just 19 MFLOPS, corresponding to a speedup factor of approximately 4.5. The most dramatic

TABLE 7. Ablation study under different compression ratios (CR): NIDA-SVD (ours) using Parameter Importance (PI) as suggested in FWSVD, and our proposed Neuron Importance (NI). In 38 out of 40 cases (95%), NI is more effective than PI and in 35 (87.5%) it improves upon SVD-LLMv2. In 26 out of 40 cases (65%), using PI makes the results worse than not using it at all (NIDA-SVD (PI) vs SVD-LLMv2). The best results are in bold, while the second-best are underlined.

CR	Method	Rank Allocation	cola	mnli-m	mnli-mm	mrpc	qnli	qqp	sst2	stsb
	SVD-LLMv2		0.5548	0.8407	**0.8423**	0.9075	0.9115	0.8740	**0.9174**	0.8838
0.1	NIDA-SVD (PI)	uniform	0.5482	0.8416	0.8389	0.9109	0.9108	0.8734	0.9139	0.8830
	NIDA-SVD (NI)		**0.5600**	**0.8434**	0.8418	**0.9157**	**0.9119**	**0.8747**	0.9160	**0.8846**
	SVD-LLMv2		**0.5254**	0.8349	0.8335	0.9071	0.9009	0.8652	0.9151	0.8729
0.2	NIDA-SVD (PI)	uniform	0.4975	0.8334	0.8284	**0.9159**	0.9033	0.8658	0.9048	0.8704
	NIDA-SVD (NI)		0.5094	**0.8383**	**0.8352**	0.9128	**0.9068**	**0.8684**	**0.9185**	**0.8740**
	SVD-LLMv2		**0.4543**	0.8210	0.8152	0.8863	0.8755	0.8514	0.9025	0.8239
0.3	NIDA-SVD (PI)	uniform	0.3650	0.8141	0.8022	0.8970	0.8791	0.8489	0.8922	0.8285
	NIDA-SVD (NI)		0.4451	**0.8245**	**0.8203**	**0.9003**	**0.8898**	**0.8561**	**0.9048**	**0.8420**
	SVD-LLMv2		0.3206	0.7766	0.7691	0.8376	0.8088	0.8197	0.8876	0.7295
0.4	NIDA-SVD (PI)	uniform	0.2937	0.7687	0.7473	0.8652	0.8160	0.8174	0.8887	0.7433
	NIDA-SVD (NI)		**0.3271**	**0.7901**	**0.7851**	**0.8657**	**0.8359**	**0.8322**	**0.8979**	**0.7652**
	SVD-LLMv2		0.2079	0.6727	0.6678	0.8176	0.6875	0.7602	**0.8772**	0.6426
0.5	NIDA-SVD (PI)	uniform	0.1932	0.6685	0.6615	**0.8384**	0.6732	0.7612	0.8566	0.6024
	NIDA-SVD (NI)		**0.2394**	**0.7143**	**0.7160**	0.8330	**0.7113**	**0.7837**	0.8704	**0.6659**

TABLE 8. Ablation study: Our index-based rank allocation algorithm improves vanilla SVD-LLMv2 compared to other allocation strategies in 27 out of 40 cases (67.5%). The best results are in bold, while the second-best are underlined.

CR	Method	Rank Allocation	cola	mnli-m	mnli-mm	mrpc	qnli	qqp	sst2	stsb
		uniform	0.5548	0.8407	0.8423	0.9075	0.9115	**0.8740**	**0.9174**	0.8838
0.1	SVD-LLMv2	role-based	**0.5573**	0.8419	0.8429	0.9087	**0.9126**	0.8736	0.9139	0.8841
		index-based	0.5490	**0.8420**	**0.8435**	**0.9089**	0.9088	0.8737	0.9174	**0.8842**
		uniform	0.5254	0.8349	0.8335	0.9071	0.9009	0.8652	**0.9151**	**0.8729**
0.2	SVD-LLMv2	role-based	0.5325	0.8369	0.8340	**0.9134**	0.9002	0.8657	0.9110	0.8722
		index-based	**0.5334**	**0.8387**	**0.8356**	0.9115	**0.9028**	**0.8670**	0.9116	0.8727
		uniform	0.4543	0.8210	0.8152	0.8863	0.8755	0.8514	**0.9025**	0.8239
0.3	SVD-LLMv2	role-based	0.4556	0.8206	0.8171	**0.8907**	0.8766	0.8531	0.9002	**0.8242**
		index-based	**0.4796**	**0.8255**	**0.8179**	0.8874	**0.8810**	**0.8544**	0.9005	0.8237
		uniform	0.3206	0.7766	0.7691	0.8376	0.8088	0.8197	0.8876	**0.7295**
0.4	SVD-LLMv2	role-based	0.3276	0.7802	0.7715	0.8366	0.8103	0.8199	0.8944	0.7231
		index-based	**0.3446**	**0.7835**	**0.7770**	**0.8496**	**0.8222**	**0.8265**	**0.8948**	0.7234
		uniform	0.2079	0.6727	0.6678	0.8176	0.6875	0.7602	**0.8772**	0.6426
0.5	SVD-LLMv2	role-based	0.2039	0.6762	0.6750	0.8176	0.6917	**0.7698**	0.8727	0.6435
		index-based	**0.2336**	**0.6846**	**0.6797**	**0.8237**	**0.7054**	0.7657	0.8772	**0.6441**

TABLE 9. Ablation study: Our index-based rank allocation algorithm improves NIDA-SVD compared to other allocation strategies in 28 out of 40 cases (70%). The best results are in bold, while the second-best are underlined.

CR	Method	Rank Allocation	cola	mnli-m	mnli-mm	mrpc	qnli	qqp	sst2	stsb
		uniform	0.5600	0.8434	0.8418	**0.9157**	0.9119	0.8747	0.9160	0.8846
0.1	NIDA-SVD	role-based	**0.5656**	0.8450	0.8416	0.9141	0.9110	**0.8749**	**0.9174**	0.8852
		index-based	0.5522	**0.8456**	**0.8444**	0.9078	**0.9121**	0.8739	0.9162	**0.8866**
		uniform	0.5094	0.8383	0.8352	**0.9128**	**0.9068**	0.8684	**0.9185**	0.8740
0.2	NIDA-SVD	role-based	**0.5198**	0.8386	0.8363	0.9056	0.9066	0.8687	0.9185	0.8754
		index-based	0.5147	**0.8396**	**0.8387**	0.9090	0.9068	**0.8690**	0.9116	**0.8793**
		uniform	0.4451	0.8245	0.8203	0.9003	0.8898	0.8561	0.9048	0.8420
0.3	NIDA-SVD	role-based	0.4365	0.8247	0.8221	**0.9063**	0.8896	0.8569	**0.9059**	0.8433
		index-based	**0.4736**	**0.8283**	**0.8224**	0.9026	**0.8923**	**0.8576**	0.9013	**0.8608**
		uniform	0.3271	0.7901	0.7851	0.8657	0.8359	0.8322	**0.8979**	0.7652
0.4	NIDA-SVD	role-based	0.3325	0.7911	0.7879	0.8698	0.8367	0.8329	0.8967	0.7685
		index-based	**0.3486**	**0.7954**	**0.7907**	**0.8708**	**0.8458**	**0.8345**	0.8956	**0.8031**
		uniform	0.2394	0.7143	0.7160	0.8330	0.7113	0.7837	0.8704	0.6659
0.5	NIDA-SVD	role-based	0.2424	0.7152	0.7163	0.8348	0.7127	0.7849	**0.8715**	0.6648
		index-based	**0.2480**	**0.7255**	**0.7251**	**0.8358**	**0.7331**	**0.7890**	0.8704	**0.7076**

efficiency improvement is seen in TinyBERT, where a 30% parameter reduction (from 14.3M to 10.0M) results in a spectacular 90% reduction in computational cost, dropping from ≈ 9 MFLOPS/token down to less than 1 MFLOP/token.

Finaly, the comparison between our NIDA-SVD method and the SVD-LLMv2 method reveals that our importance-aware rank allocation strategy does not impose a measurable computational overhead. Both methods achieve nearly identical parameter counts and, consequently, highly similar MFLOPS/token values across all architectures. This is an expected outcome, as the fundamental change to the network's architecture (replacing a dense matrix $\boldsymbol{W}$ with two smaller matrices $\boldsymbol{A}$ and $\boldsymbol{B}$) is the same for both approaches, regardless of how the compression rate is distributed, since

TABLE 10. **Computational and memory analysis. Comparison of MLFLOPS/token and total parameters (in Millions) for base and compressed DistilBERT, MobileBERT, and TinyBERT models.**

Architecture	GLUE	CR	Compression Method	MLFLOPS/token	Total Parameters
distilBERT	mrpc	N/A	base	85.96	66.9
		0.5	SVD-LLMv2	18.99	33.4
			NIDA-SVD	18.99	33.4
	qnli	N/A	base	85.97	66.9
		0.5	SVD-LLMv2	18.99	33.4
			NIDA-SVD	19.07	33.5
	sst2	N/A	base	85.56	66.9
		0.5	SVD-LLMv2	18.59	33.4
			NIDA-SVD	18.73	33.5
mobileBERT	mrpc	N/A	base	41.17	24.5
		0.5	SVD-LLMv2	16.33	12.1
			NIDA-SVD	16.48	12.2
	qnli	N/A	base	41.18	24.5
		0.5	SVD-LLMv2	16.33	12.1
			NIDA-SVD	16.48	12.2
	sst2	N/A	base	40.89	24.5
		0.5	SVD-LLMv2	16.05	12.1
			NIDA-SVD	16.24	12.2
tinyBERT	mrpc	N/A	base	9.38	14.3
		0.3	SVD-LLMv2	0.75	10.0
			NIDA-SVD	0.73	10.0
	qnli	N/A	base	9.38	14.3
		0.3	SVD-LLMv2	0.75	10.0
			NIDA-SVD	0.73	10.0
	sst2	N/A	base	9.27	14.3
		0.3	SVD-LLMv2	0.64	10.0
			NIDA-SVD	0.62	10.0

the total computational budget for a fixed compression ratio remains constant. This confirms that in the rank allocation proccess, the performance efficacy of NIDA-SVD in contrast to other SVD-based methods is achieved solely through a smarter distribution of ranks (resources) across layers, not by increasing the computational expense.

VII. CONCLUSION

In summary, this work introduced NIDA-SVD, a novel framework to compress LLM matrices that integrates neuron importance with data-aware low-rank approximation, alongside a computationally efficient algorithm for dynamic rank allocation. Experimental results demonstrate that NIDA-SVD consistently outperforms prior state-of-the-art methods under diverse allocation strategies, while our proposed rank allocation algorithm also independently strengthens both NIDA-SVD and previous approaches. Together, these contributions establish a robust and versatile compression solution, achieving substantial performance gains, especially at high compression ratios, paving the way for a more effective deployment of large-scale language models in resource-constrained setups.

REFERENCES

[1] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," OpenAI, San Francisco, CA, USA, Tech. Rep., 2018.

[2] J. Devlin and M. Chang, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2018, pp. 4171–4186.

[3] M. Rivière et al., "Gemma 2: Improving open language models at a practical size," 2024, *arXiv:2408.00118*.

[4] A. Kamath et al., "Gemma 3 technical report," 2025, *arXiv:2503.19786*.

[5] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.

[6] L. Caruccio, S. Cirillo, G. Polese, G. Solimando, S. Sundaramurthy, and G. Tortora, "Claude 2.0 large language model: Tackling a real-world classification problem with a new iterative prompt engineering approach," *Intell. Syst. Appl.*, vol. 21, Jan. 2024, Art. no. 200336.

[7] M. Gupta and P. Agrawal, "Compression of deep learning models for text: A survey," *ACM Trans. Knowl. Discovery Data*, vol. 16, no. 4, pp. 1–55, Aug. 2022.

[8] X. Zhu, J. Li, Y. Liu, C. Ma, and W. Wang, "A survey on model compression for large language models," *Trans. Assoc. Comput. Linguistics*, vol. 12, pp. 1556–1577, Nov. 2024.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2025, pp. 5998–6008.

[10] A. Hoffman, "A generalization of the Eckart-Young-Mirsky matrix approximation theorem," *Linear Algebra Appl.*, vol. 88, pp. 317–327, Jan. 1987.

[11] L. T. Nguyen, J. Kim, and B. Shim, "Low-rank matrix completion: A contemporary survey," *IEEE Access*, vol. 7, pp. 94215–94237, 2019.

[12] Y.-C. Hsu, H. Ting, S. Chang, Q. Lou, Y. Shen, and H. Jin, "Language model compression with weighted low-rank factorization," in *Proc. Int. Conf. Learn. Represent.*, 2022, pp. 1–11.

[13] X. Wang, S. Alam, Z. Wan, H. Shen, and M. Zhang, "SVD-LLM v2: Optimizing singular value truncation for large language model compression," in *Proc. Conf. Nations Americas Chapter Assoc. Comput. Linguistics, Human Lang. Technol.*, 2025, pp. 4287–4296.

[14] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter," 2019, *arXiv:1910.01108*.

[15] D. Chen and J. Zhou, "LightMobileBert: A secondary lightweight model based on MobileBert," *J. Intell. Fuzzy Syst.*, vol. 44, no. 2, pp. 2117–2129, Jan. 2023.

[16] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "TinyBERT: Distilling BERT for natural language understanding," in *Proc. Findings Assoc. Comput. Linguistics, EMNLP*, 2020, pp. 4163–4174.

[17] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," 2019, *arXiv:1907.11692*.
[18] P. He, X. Liu, J. Gao, and W. Chen, "DeBERTa: Decoding-enhanced BERT with disentangled attention," in *Proc. Int. Conf. Learn. Represent.*, 2020.
[19] P. He, J. Gao, and W. Chen, "DeBERTaV3: Improving DeBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing," in *Proc. 11th Int. Conf. Learn. Represent.*, 2021.
[20] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peşte, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," *J. Mach. Learn. Res.*, vol. 22, no. 241, pp. 1–124, 2021.
[21] S. Ashkboos, M. L. Croci, M. G. Nascimento, T. Hoefler, and J. Hensman, "SliceGPT: Compress large language models by deleting rows and columns," in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2024.
[22] T. F. Van Der Ouderaa, M. Nagel, M. Van Baalen, and T. Blankevoort, "The LLM surgeon," in *Proc. 12th Int. Conf. Learn. Represent.*, 2023, pp. 1–9.
[23] E. Frantar and D. Alistarh, "SparseGPT: Massive language models can be accurately pruned in one-shot," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 10323–10337.
[24] L. Hou, Z. Huang, L. Shang, X. Jiang, X. D. Chen, and Q. Liu, "DynaBERT: Dynamic BERT with adaptive width and depth," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 9782–9793.
[25] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *Int. J. Comput. Vis.*, vol. 129, no. 6, pp. 1789–1819, 2021.
[26] G. Cai, J. Li, X. Liu, Z. Chen, and H. Zhang, "Learning and compressing: Low-rank matrix factorization for deep neural network compression," *Appl. Sci.*, vol. 13, no. 4, p. 2704, Feb. 2023.
[27] H. Yu and J. Wu, "Compressing transformers: Features are low-rank, but weights are not!" in *Proc. AAAI Conf. Artif. Intell.*, 2023, vol. 37, no. 9, pp. 11007–11015.
[28] P. Chen, H. Yu, I. S. Dhillon, and C. Hsieh, "DRONE: Data-aware low-rank compression for large NLP models," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 29321–29334.
[29] Z. Yuan, Y. Shang, Y. Song, D. Yang, Q. Wu, Y. Yan, and G. Sun, "ASVD: Activation-aware singular value decomposition for compressing large language models," 2023, *arXiv:2312.05821*.
[30] X. Wang, Y. Zheng, Z. Wan, and M. Zhang, "SVD-LLM: Truncation-aware singular value decomposition for large language model compression," in *Proc. 13th Int. Conf. Learn. Represent.*, 2024, pp. 1–7.
[31] M. B. Noach and Y. Goldberg, "Compressing pre-trained language models by matrix decomposition," in *Proc. 1st Conf. Asia–Pacific Chapter Assoc. Comput. Linguistics 10th Int. Joint Conf. Natural Lang. Process.*, 2020, pp. 884–889. [Online]. Available: https://aclanthology.org/2020.aacl-main.88
[32] Z. Liu, C. Zhao, I. Fedorov, B. Soran, D. Choudhary, R. Krishnamoorthi, V. Chandra, Y. Tian, and T. Blankevoort, "SpinQuant: LLM quantization with learned rotations," in *Proc. 13th Int. Conf. Learn. Represent.*, 2025, pp. 1–8.
[33] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate post-training quantization for generative pre-trained transformers," 2022, *arXiv:2210.17323*.
[34] J. Lin, J. Tang, H. Tang, S. Yang, G. Xiao, and S. Han, "AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration," in *Proc. Mach. Learn. Syst.*, vol. 28, 2025, pp. 12–17.
[35] H. Bai, W. Zhang, L. Hou, L. Shang, J. Jin, X. Jiang, Q. Liu, M. Lyu, and I. King, "BinaryBERT: Pushing the limit of BERT quantization," in *Proc. 59th Annu. Meeting Assoc. Comput. Linguistics 11th Int. Joint Conf. Natural Lang. Process.*, 2021, pp. 4334–4348.
[36] E. Kim, K.-H. Lee, and W.-K. Sung, "Optimizing spatial shift point-wise quantization," *IEEE Access*, vol. 9, pp. 68008–68016, 2021.
[37] M. Chen, W. Shao, P. Xu, J. Wang, P. Gao, K. Zhang, and P. Luo, "EfficientQAT: Efficient quantization-aware training for large language models," in *Proc. 63rd Annu. Meeting Assoc. Comput. Linguistics*, 2025, pp. 10081–10100.
[38] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[39] J. Baglama and L. Reichel, "Augmented implicitly restarted Lanczos bidiagonalization methods," *SIAM J. Sci. Comput.*, vol. 27, no. 1, pp. 19–42, Jan. 2005.
[40] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Rev.*, vol. 53, no. 2, pp. 217–288, Jan. 2011.
[41] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proc. EMNLP Workshop BlackboxNLP, Analyzing Interpreting Neural Netw. NLP*, 2018, pp. 353–355.

ATHANASIOS NTOVAS received the Diploma degree from the Computers and Communication Engineering Department, University of Thessaly (UTH), in 2019.

Since April 2020, he has been a Research Assistant with the Information Technologies Institute (ITI), Centre for Research and Technology Hellas (CERTH). His research interests include artificial intelligence, computer vision, and digital signal processing.

ALEXANDROS DOUMANOGLOU received the Diploma degree in electrical and computer engineering from the Aristotle University of Thessaloniki (A.U.Th.). He is currently pursuing the Ph.D. degree in explainable and interpretable artificial intelligence with the Department of Advanced Computing Sciences, Maastricht University, under the supervision of Prof. Kurt Driessens. He joined the Information Technologies Institute (ITI) in 2012, where he has since worked as a Research Assistant in the areas of computer vision, 3D graphics, and machine learning. Concurrently with his Ph.D. studies, he remains affiliated with ITI as a Senior Researcher. His current research interests include computer vision, unsupervised and representation learning, mechanistic interpretability, and explainable and interpretable methods for deep learning models.

PETROS DRAKOULIS received the B.Sc. degree in informatics from International Hellenic University and the M.Sc. degree in digital media and computational intelligence from the Aristotle University of Thessaloniki.

In 2018, he joined the Visual Computing Laboratory, CERTH-ITI, where he has been a Research Associate ever since. His main research interests include software engineering, visual computing, machine learning, and graphics.

DIMITRIS ZARPALAS received the Diploma degree in electrical and computer engineering from the Aristotle University of Thessaloniki (AUTH), the M.Sc. degree in electrical engineering (focusing on computer vision) from The Pennsylvania State University, and the Ph.D. degree in medical informatics from the Health Science School, Department of Medicine, AUTH.

He joined the Information Technologies Institute, in 2007. He is currently a Researcher (Grade C). His research interests include real-time tele-immersion applications (3D reconstruction of moving humans and their compression), 3D computer vision, 3D medical image processing, shape analysis of anatomical structures, 3D object recognition, and motion capturing and evaluation, while in the past he has also worked in indexing, search and retrieval, classification of 3D objects, and 3D model watermarking.