ADAPTIVE RANK ALLOCATION: SPEEDING UP MODERN TRANSFORMERS WITH RANA ADAPTERS

Anonymous authors
Paper under double-blind review

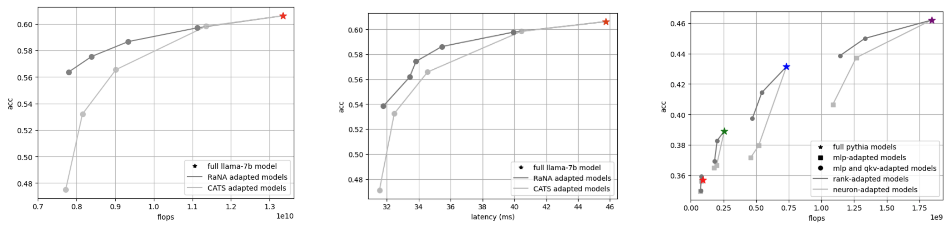
ABSTRACT

Large Language Models (LLMs) are computationally intensive, particularly during inference. Neuron-adaptive techniques, which selectively activate neurons in Multi-Layer Perceptron (MLP) layers, offer some speedups but suffer from limitations in modern Transformers. These include reliance on sparse activations, incompatibility with attention layers, and the use of costly neuron masking techniques. To address these issues, we propose the Adaptive Rank Allocation framework and introduce the Rank and Neuron Allocator (RaNA) adapter. RaNA adapters leverage rank adapters, which operate on linear layers by applying both low-rank matrix decompositions and adaptive masking to efficiently allocate compute without depending on activation sparsity. This enables RaNA to be generally applied to MLPs and linear components of attention modules, while eliminating the need for expensive maskers found in neuron-adaptive methods. Notably, when compared to neuron adapters, RaNA improves perplexity by up to 7 points and increases accuracy by up to 8 percentage-points when reducing FLOPs by $\sim$44% in state-of-the-art Transformer architectures. These results position RaNA as a robust solution for improving inference efficiency in modern Transformer architectures.

1 INTRODUCTION

As Large Language Models (LLMs) have grown in popularity and size, they have begun consuming a non-trivial amount of compute and time for training and inference (Kim et al. (2023), Pope et al. (2022)). Adaptive compute methods seek to speed up the inference stage of Transformers (Vaswani et al. (2023)), the *de facto* LLM architecture, by identifying and avoiding redundant computations to save I/O and floating-point operations (FLOPs). Commonly, these methods apply neuron adapters to the Multi Layer Perceptron (MLP) layers of the Transformer architecture, which dynamically ignore neurons depending on the input of the layer (Lee et al. (2024), Mirzadeh et al. (2023), Liu et al. (2023) Zhang et al. (2024)). Utilizing these adapters leads to inference speedups, as the amortized time complexity of an MLP layer reduces from $\mathcal{O}(d_{in} \cdot d_{hidden})$ to $\mathcal{O}(d_{in} \cdot d_{adapted})$, where d_{in} represents the input dimension of the MLP, d_{hidden} is the hidden dimension, and $d_{adapted}$ is the average number of active neurons. For modern Transformer architectures, this results in practical speedups at the expense of negligible model quality decrease for a sufficient amount of average active neurons.

Unfortunately, adaptive compute methods using neuron adapters suffer from rapid performance degradation in modern Transformer architectures (Figs. 1a, 1c). We observe this issue arises from the limitations of the neuron-adaptive framework, which enforces dynamic neuron allocation based on activation values. Concretely, we believe this decline is attributed to this framework's lack of generalization across different layer types, reliance on sparse activation functions, and costly neuron masking techniques. First, neuron adapters, often tailored to MLPs, can not be directly applied to non-MLP layers, such as Query, Key, Value, and attention layers, which lack neurons. In addition, these methods frequently depend on activation-induced sparsity, like that attained by ReLU (Agarap (2019)), making them ineffective with non-sparse activation functions like SwiGLU (Shazeer (2020), Zhang et al. (2024)). Further, while recent approaches like CATS (Lee et al. (2024)) or ReLUfication (Mirzadeh et al. (2023)) attempt to handle non-sparse activations, they inefficiently compute neuron-activations before determining which neurons to exclude. This is concerning as Transformers architectures like Llama (Touvron et al. (2023)), Gemma (Team et al. (2024)) or Mis-



(a) Llama2-7b Accuracy v.s. FLOPs. (b) Llama2-7b Accuracy v.s. Latency. (c) Pythia Suite Accuracy v.s. FLOPs.

Figure 1: **RaNA improves accuracy-compute tradeoff over neuron adapters**. y-axis shows accuracy averaged over multiple downstream tasks (Sect. 5.1); for Figs. 1a and 1c, x-axis shows average FLOPs for a forward pass with sequence length 512; for Fig. 1b x-axis shows average per-token decoding latency over a sequence of 492 tokens with initial context lengths ranging from 1 to 1000. We compare RaNA-adapted models to neuron-adapted versions at various compression rates for (left) Llama2-7b and (right) Pythia models. Notably, RaNA accuracies decay slower as compression rates increase compared to neuron-adapters.

tral (Jiang et al. (2023)) rely on non-sparse activation functions, which makes sparsity-based neuron adapters ineffective for them, pushing us to rely on more computationally intensive neuron adapters.

To address these gaps in the neuron-adaptive setup, we propose the Adaptive Rank Allocation framework, together with the Rank and Neuron Allocator (RaNA) adapter. We note that this rank-adaptive framework is a more powerful generalization of the neuron-adaptive one. Moreover, we develop RaNA in this framework, which unlike neuron adapters, can be directly applied to any linear layer, where it does not rely on activation function sparsity. In addition, when applied to modern MLP layers with SwiGLU activations, it allows us to better distribute compute across the different linear layers of the MLP, unlike previous neuron adapters. Concretely, we devise the Adaptive Rank Allocation framework from the observation that any linear layer can be decomposed into the product of two low-rank matrices and an adaptive router/masker, namely Linear$(x) = Wx \approx A(r(x) \odot Bx)$. Then, RaNA is a specific adapter following this setup, where we devise specific A and B decompositions and a masker $m(x)$ for Transformer layers layers like QKV or MLPs.

We empirically validate the rank-adaptive framework and the RaNA adapter. We demonstrate that, similar to neuron-adaptive setups, the ranks of the AB matrix decomposition in the proposed RaNA adapters have sparse importances, depending on the input (Figs. 2a, 2b), allowing us to dynamically prune them. We also show that RaNA adapters attain the lowest error in MLP layers when recovering the full MLP outputs, outperforming neuron-adaptive methods by 6.7, 18.1 and 7.4 percentage-points on Llama2-7b, Gemma-2b and Pythia-160M respectively. Further, we show the effectiveness of RaNA in modern Transformer architectures by applying it to Llama (Touvron et al. (2023)) and Gemma (Team et al. (2024)). Notably, RaNA closes the gap between full-model and adapted-model performance, as it outperforms the state-of-the-art method CATS (Lee et al. (2024)) on multiple compression rates (Tabs. 1, 2). Concretely, at a $\sim$45% FLOP compression rate, RaNA achieves an average improvement of 4 perplexity-points and 8 percentage-points in benchmark accuracy for Llama2-7b, while for Gemma-2b, it improves perplexity by 7 points and accuracy by 5 percentage-points, compared to prior adapters. Finally, to assess the effectiveness of RaNA's applicability to different types of layers and activations, we apply it to the Pythia suite (Biderman et al. (2023)), a set of varied-sized GPTNeox (Black et al. (2022)) models. Here we also observe that, when comparing their performance to conventional neuron-adaptive methods, it consistently attains better perplexity and downstream task performance (Figs. 1c, 4).

2 RELATED WORK AND PRELIMINARIES

Here we discuss key work in neuron-adaptive methods and examine neuron adapters for Transformers with sparse activations and with modern non-sparse activations.

Neuron-adaptive framework: Adaptive compute methods (Han et al. (2021)) are a popular approach to speed up inference in Transformer (Vaswani et al. (2023)) architectures. A concrete instance of them are neuron adapters, which dynamically allocate neurons of the MLP layers to different inputs using a masker. These neuron-adaptive methods for Transformers stem from the observation that neuron importance is sparse, and while Transformers allocate equal resources to all tokens, some tokens require less compute based on context (Li et al. (2023b)). Additionally, redundant computations are common due to model over-parameterization (Frankle & Carbin (2019)).

Nevertheless, the neuron-adaptive framework is constraining, as it only allows us to dynamically allocate neurons, which are only present on MLP layers of Transformers. Additionally, neuron adapters do not generalize well to different activation functions in the MLP layers (Zhang et al. (2024)). Hence, previous work has focused on creating adapters specific to certain activation functions, namely sparse activation functions and non-sparse activation functions.

Neuron adapters for sparse activations: In the case of Transformers leveraging sparse activation functions like ReLU (Agarap (2019)), neuron adapters rely on the abundance of 0-valued neuron activations (Li et al. (2023a)). These adapters commonly work by using a small MLP masker that predicts whether a neuron is going to be active for a given input, then only computing that neuron if its prediction is positive. Concretely, consider a conventional MLP layer with a ReLU activation:

$$\text{MLP}_{ReLU}(x) = W_{down}(\text{ReLU}(W_{up}x))) \tag{1}$$

where $W_{up} \in \mathbb{R}^{d,h}$ and $W_{down} \in \mathbb{R}^{h,d}$ are the Up-Projection and Down-Projection matrices of the MLP layer. Then, the neuron adapted version of this MLP follows:

$$\text{MLP}'_{ReLU}(x) = W_{down}(m(x) \odot \text{ReLU}(W_{up}x)) \tag{2}$$

where $m(x) : \mathbb{R} \to \{0,1\}^h$ is the binary masker, which often is parameterized by a small MLP. While effective in these type of models, given their strong reliance on the sparsity of neuron activations, this adapters have unfortunately shown poor performance when applied to Transformers with non-sparse activations (Zhang et al. (2024)).

Neuron adapters for non-sparse activations: Modern Transformer architectures, like those used in popular models like Llama (Touvron et al. (2023)), Gemma (Team et al. (2024)), Mistral (Jiang et al. (2023)) or GPTNeox (Black et al. (2022)), leverage non sparse activation functions like GeLU (Hendrycks & Gimpel (2023)) for GPTNeox and the more popular SwiGLU (Shazeer (2020)) for the others. Concretely, an MLP layer with a SwiGLU activation follows:

$$\text{MLP}_{SwiGLU}(x) = W_{down}(\text{SiLU}(W_{gate}x) \odot W_{up}x) \tag{3}$$

where $W_{up} \in \mathbb{R}^{d,h}$, $W_{down} \in \mathbb{R}^{h,d}$ and $W_{gate} \in \mathbb{R}^{d,h}$ are the Up-Projection, Down-Projection and Gate-Projection matrices of the MLP layer.

Recent work has sought to improve the performance of conventional neuron adapters on non-sparse activations by developing methods specifically tailored towards them, such as CATS (Lee et al. (2024)) and ReLUfication (Mirzadeh et al. (2023)). While these approaches have demonstrated good performance in SwiGLU-based Transformers, they suffer from inefficiencies. The inefficiency stems from their reliance on calculating exact activation values prior to applying thresholding. For instance, CATS adapters compute the entire output of the Gate-Projection layer before selecting which neurons to retain. At large compression rates, this enforces a sub-optimal FLOP allocation imbalance across the Up, Down and Gate projection layers of the MLP, where the Gate-Projection layer receives most of the FLOPs.

3 Adaptive Rank Allocation Framework

We begin by presenting the Adaptive Rank Allocation Framework, which addresses the shortcomings of the neuron-adaptive approach, specifically its limitation to neuron adaptation and its frequent reliance on neuron sparsity. Adaptive compute frameworks generally involve two key parts: dynamically allocated components and a function to determine their use based on the input. We make the observation that such an adaptive compute setup can be applied at the granularity of linear layers, where the dynamically allocated components are the ranks of a weight matrix, while the function that determines which ranks are leveraged by a given input is a router or masker. This is the main observation that motivates our Adaptive Rank Allocation Framework, which can be applied to *arbitrary linear layers without relying on activation function sparsity*.

Concretely, consider any linear layer Linear$(x) = Wx$, where $W \in \mathbb{R}^{o,i}$. Under the Adaptive Rank Allocation framework, we replace this linear layer by:

$$\text{Linear'}(x) = A(r(x) \odot Bx) \tag{4}$$

This simple setup has two parts. First, we have a set of static matrices A and B, where $A \in \mathbb{R}^{o,d}$ and $B \in \mathbb{R}^{d,i}$. Second, we have the adaptive component $r(x)$, where $r : \mathbb{R}^i \to \mathbb{R}^d$. Ideally we want $r(x)$ to be sparse, as that allows us to save I/O and floating-point operations. If $r(x)$ is sparse, our rank adapted layer from Eqn. 4 becomes essentially a low-rank matrix multiplication, where the the low-rank matrix $(A \text{ diag}(r(x)) \ B)$ has rank= $\|r(x)\|_0$. Notably, the FLOPs consumed by such a matrix multiplication are proportional to the rank of such matrix, hence why sparsity is desirable.

Generalization of Neuron-Adaptive Methods: We note that the Adaptive Rank Allocation framework is a strict generalization of the neuron-adaptive one, where neuron adapters can be viewed as a specific instance of rank adapters with appropriate choices of A, B and $r(x)$. In Prop. 1, we illustrate this for ReLU-based MLPs, but we note the proof easily extends to other activation functions:

Proposition 1. *Consider an MLP$(x)_{ReLU}$ layer (Eqn. 1) and its neuron adapted version MLP$'(x)_{ReLU}$ (Eqn. 2). Then, there exists a rank adapted MLP$^*_{ReLU}$ (i.e. an MLP whose linear layers have been rank adapted) s.t. MLP$^*_{ReLU}(x)$ = MLP$'(x)_{ReLU}$ for all $x \in \mathbb{R}^i$.*

Proof. Let $\text{MLP}^*_{ReLU}(x) = A_{down}(r_{down}(x) \odot B_{down}(\text{ReLU}(A_{up}(r_{up}(x) \odot B_{up}x))))$.

Then, by setting $A_{down} := W_{down}$, $B_{down} := \mathbf{I}$, $A_{up} := \mathbf{I}$, $B_{up} := W_up$, $r_{down} := m_{down}$, and $r_{up} := \mathbf{1}$, it follows that:

$$\begin{aligned}\text{MLP}^*_{ReLU}(x) &= W_{down}(m_2(x) \odot \mathbf{I}(\text{ReLU}(\mathbf{I}(\mathbf{1} \odot W_{up}x)))) \\ &= W_{down}(m_2(x) \odot \text{ReLU}(W_{up}x)) = \text{MLP}'_{ReLU}(x).\end{aligned}$$

In summary, this result shows that rank adaptation can generalize neuron adaptation, highlighting the increased versatility and applicability of the Adaptive Rank Allocation framework.

4 Rank and Neuron Allocator Adapters

Here, we introduce the Rank and Neuron Allocator (RaNA) adapters, which serve as specific instances of the Adaptive Rank Allocation framework. At their core, RaNA adapters leverage Linear Layer Rank adapters that operate at the granularity of linear layers, which we also introduce in this section. Concretely, we begin in Sect. 4.1 by diving into the Linear Layer Rank adapter, a versatile adapter which employs approximately optimal A and B matrices, along with a dynamic rank allocation mechanism, adjusted based on the input distribution. As stand-alone adapters, while effective in QKV, Up-Projection, and Gate-Projection layers, Linear Layer Rank adapters struggle in Down-Projection layers. To overcome this limitation, we propose RaNA adapters.

RaNA adapters operate at the level of QKV and MLP layers, combining Linear Layer Rank adapters with neuron-thresholding techniques, a setup that has demonstrated the best empirical performance when adaptively compressing layers (Figs. 3a, 3b, 3c, 3d). Notably, these adapters address the inefficiencies of previous non-sparse neuron adapters by avoiding the costly exact computation of specific linear layer outputs prior to applying neuron thresholding. Furthermore, they empirically demonstrate their effectiveness in model accuracy and perplexity compared to other adapters (Tabs. 1, 2).

4.1 Linear Layer Rank Adapters

The Adaptive Rank Allocation framework, unlike the neuron-adaptive one, does not impose limitations on the A or B matrices or the routing function $r(x)$ when adapting a linear layer through Linear$'(x) = A(m(x) \odot Bx) \approx Wx$. We leverage these properties to introduce the Linear Layer Rank Adapters. Here, we want to find the optimal A, B and $r(x)$ that, in expectation over the input distribution, best recover the original output of the linear layer. Formally, we want:

$$\text{argmin}_{A,B,r} E_x(\|Wx - A(r(x) \odot Bx)\|_F^2) \tag{5}$$

Evidently, optimally finding A, B and r is non-trivial. Hence, we choose to address this problem by breaking it into two steps. First, we propose a set of A and B matrices that solve a similar problem to Eqn. 5, which ignores the routing function. Then, with our election of A and B matrices, we propose an optimal input-dependent masker, which serves as a router, for our Linear Layer Rank Adapter.

A and B matrices: First, we devise a set of A and B matrices for our rank-adapted layer Linear'$(x) = A(r(x) \odot Bx)$. We start off by relaxing the optimization problem from Eqn. 5 so that we can obtain good candidate A and B matrices analytically. Concretely, we remove the router from the picture and frame the problem as finding fixed low-rank matrices A_r and B_r, where r denotes the rank of the matrices, that minimize our objective. Namely, the relaxed objective becomes:

$$\text{argmin}_{A_r,B_r} E_x(\|Wx - A_r B_r x\|_F^2) \tag{6}$$

Note that this relaxed problem is not only convenient, but also a decent choice, as it is equivalent to solving the original problem from Eqn. 5 with a fixed router $r(x)$ that always outputs 1's for the first r elements of the output vector and 0's for the rest. In practice, we do not have access to the actual distribution of the inputs x, hence we reframe the problem to its empirical version:

$$\text{argmin}_{A_r,B_r} \|WX - A_r B_r X\|_F^2 \tag{7}$$

where each column of $X \in \mathbb{R}^{i,k}$, namely x_i, is a hidden-state input that our linear layer observes in practice. Notably, we used $k = 32{,}000$ samples in our experiments, as we empirically found that going beyond that did not impact our objective.

We can then analytically find the optimal A_r and B_r matrices that solve this problem.

Theorem 1. *Let $U_r \in \mathbb{R}^{o,r}$ be the first r singular vectors of WX, then by the Eckart-Young theorem (Eckart & Young (1936)), the A_r and B_r that optimize the objective from Eqn. 7 are:*

$$A_r := U_r, B_r := U_r^T W.$$

The proof for Theorem 1 is provided in Appendix A.1. Therefore, we pick our A,B matrices as $A := U$, $B := U^T W$. Next, we demonstrate how to pick a masker/router to effectively leverage these matrices in an input-adaptive manner.

B-Masker: With our $A = U$ and $B = U^T W$ matrices at hand, we devise a sparse router $r(x)$ function that minimizes the adapter's error (Eqn. 5). For simplicity, we opt for a binary $r(x)$, i.e. a masker $r(x) = m(x) : \mathbb{R}^i \to \{0,1\}^i$ that allows us to allocate different ranks (and hence different amount of FLOPs) to different input hidden states. Formally, we want $m(x)$ that optimizes:

$$\begin{aligned} &\text{argmin}_{m(x)} \sum_i^k \|Wx_i - U(m(x_i) \odot U^T W x_i)\|_F^2, \quad \text{s.t. } \mathbb{E}_x[\|m(x)\|_0] = r \\ &\equiv \text{argmax}_{m(x)} \sum_i^k \|m(x_i) \odot U^T W x_i\|_F^2, \quad \text{s.t. } \mathbb{E}_x[\|m(x)\|_0] = r \end{aligned} \tag{8}$$

The constraint in Eqn. 8 enforces that expected rank of the matrix $W' = U\ \text{diag}(m(x))\ U^T W$ is r. This constraint directly controls the FLOPs that the rank-adaptive model will consume on average.

Observe that, for any hidden state x, we can compute the contribution of each rank in our adapter to the output of the original linear layer. To do this, we examine the contribution of each column vector in $A = U$ (equivalently, the contribution of each rank of $A\ B$) to the output $o = \text{Linear}(x) \approx A(m(x) \odot Bx)$. Specifically, we note that the contribution to the Frobenius norm $||o||_F^2$ from the i-th column vector of A, u_i, is given by $(Bx)_i^2$, due to the orthogonality of the columns in $A = U$. This allows us to identify the important ranks of our A, B decomposition for a given input x and create a sparse masker $m(x)$ for it (Figs. 2a, 2b), which just keeps the most descriptive ranks. We call this the B-Masker, as it uses the B matrix to select the most important ranks for x. Formally:

$$m(x)_i = \text{B-masker}(x)_i = \begin{cases} 1 & \text{if } (Bx)_i^2 \geq t, \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

In Eqn. 9, the threshold t is picked so that, on average, a desired amount of FLOPs is consumed in our adapted layer. We note that the B-masker is efficient to use when the matrix from our linear layer $W \in \mathbb{R}^{(o,i)}$ has an output dimension o that is bigger than its inner dimension i, as this makes computing $(Bx)^2$ cheap. Hence, it is convenient to use this masker in practice for the Up-Projection and Gate-Projection layers of our models, where $o \approx 4i$, or the QKV layers, where $o \approx 3i$.

MLP-Sigmoid Masker: An alternative option to the B-masker, which has the potential to be more efficient, and that works for any linear layer, is a small MLP masker with a Sigmoid activation function, just like that used commonly in the neuron-adaptive literature (Liu et al. (2023), Zhang et al. (2024)). Concretely, we use the parametrization $m(x) = \sigma(CDx)$ where $C \in \mathbb{R}^{r,r'}$, $D \in \mathbb{R}^{r',i}$, i is the dimension of the hidden states x, r' is the inner dimension of the predictive masker, and r is the rank of the matrix W from the linear layer the masker corresponds to. In our experiments, we train this masker on a binary cross-entropy loss to match the output of the B-masker.

4.2 Fusing Rank Adapters and Neuron Thresholding for RaNA

Building on the insights from the previous section, we observe that Linear Layer Rank Adapters independently perform well for QKV, Up-Projection, and Gate-Projection layers due to their use of tall and narrow matrices. However, they may encounter difficulties with Down-Projection layers, which involve short and wide matrices. To overcome this limitation, RaNA combines Linear Layer Rank Adapters with neuron-thresholding adapters, specifically for the Down-Projection layers. Further, RaNA implements a FLOP allocation procedure across the various components of the adapter, addressing the imbalance found in previous neuron-adaptive methods, where FLOP distribution is heavily skewed toward specific components.

RaNA in QKV layers: For the QKV layers, we just replace the linear QKV layers with Linear Layer Rank Adapters, namely:

$$\text{QKV}(x) = Wx \approx \text{QKV}'(x) = A(m(x) \odot Bx) \tag{10}$$

where $m(x)$ is a B-masker, as we find it outperforms the MLP-Sigmoid masker (Fig. 3d).

RaNA in MLP layers: For MLP layers, we demonstrate how RaNA is applied to SwiGLU-based MLPs and note that its application to MLPs with other activation functions follows the same approach, excluding the Gate-Projection adapter. Namely, we use Linear Layer Rank Adapters for the Up-Projection and Gate-Projection matrices and use neuron-thresholding for the Down-Projection matrices. Formally, our RaNA adapted MLP'(x) is described by the following:

$$\begin{aligned} \text{MLP'}(x) &= \text{Down'}(\text{SiLU}(\text{Gate'}(x)) \odot \text{Up'}(x))) \\ \text{where} \quad \text{Up'}(x) &= A_{up}(m_{up}(x) \odot B_{up}x), \\ \text{Gate'}(x) &= A_{gate}(m_{gate}(x) \odot B_{gate}x), \\ \text{Down'}(x) &= W_{down}(m_{down}(x) \odot x) \end{aligned} \tag{11}$$

where $m_{up}(x)$ and $m_{gate}(x)$ are B-maskers and $m_{down}(x)$ is a simple neuron thresholding masker:

$$m(x)_i = \text{neuron-thresholding}(x)_i = \begin{cases} 1 & \text{if } |x_i| \odot ||W^{down}_{i,:}||_F \geq t, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

RaNA FLOP Allocation: Prior neuron-adapters for non-sparse activation functions are forced to ineffectively allocate FLOPs across their components (Sect. 2). On the other hand, RaNA's flexibility permits us to freely distribute FLOPs across the adapter components, therefore we propose a FLOP allocation strategy specifically for RaNA. At the Linear Layer Rank Adapter level, we perform a simple line search to balance FLOPs between the B-Masker and the target sparsity, selecting the configuration that minimizes the output error with respect to the original linear layer. Similarly, at the MLP level, we conduct a grid search to distribute FLOPs among the Up', Gate', and Down' layers, with each component further balancing FLOPs between its masker and target sparsity. We then retain the configuration that achieves the greatest error reduction in the MLP output.

Please refer to Appendix A.2 for a pseudocode implementation of RaNA.

5 Experiments and Results

In this section, we present experiments and results for the following claims:

1. **The contribution of ranks in Linear Layer Rank Adapters is sparse**. Just like this is a desirable property for neuron-adaptive approaches to work well, it is also one for the Linear Layer Rank Adapters, which we use in RaNA. Notably, the distribution of the contributions of each rank in the AB decomposition (to the recovery of the original output of the linear layer) of our rank adapters is concentrated at 0, and is heavy tailed (Figs. 2a, 2b). This validates Linear Layer Rank adapters and practically means that we can effectively mask out many of the ranks with near-0 contributions.

2. **RaNA adapters attain lowest errors on Transformer layers when replicating original layer outputs**. When compared to other adapters, RaNA attains the lowest mean squared error when comparing its outputs to those of the original layers (Figs. 3a, 3b, 3c, 3d). Concretely, we conduct a study of the error in the Llama2-7b, Gemma-2b and Pythia-160M models using multiple adaptive methods, targeting a ∼50% reduction in the FLOPs of the compressed layers. Notably, RaNA achieves average errors of 10.5% and 2.18%, surpassing the 17.22% and 20.31% attained by CATS on MLP layers for Llama2-7b and Gemma-2b respectively. Likewise, in Pythia-160M, RaNA achieves average errors of 7.87% and 0.36% in MLP and QKV layers respectively, outperforming the 15.23% and 1.29% errors obtained with neuron adapters and SVD-based adapters.
3. **RaNA outperforms neuron adapters in perplexity and downstream tasks**. Using RaNA, we attain lower perplexity and higher average accuracy on multiple NLP benchmarks in modern Transformers compared to neuron-adapters. Notably, RaNA outperforms the state-of-the art CATS adapter across multiple FLOP compression levels (Tabs. 1, 2, Fig. 1a) for SwiGLU based architectures. In the case of Llama2-7b, we achieve an improvement of 8 percentage-points in accuracy and 4 perplexity-points on a 42% model compression rate. Similarly, for Gemma-2b, we attain an improvement of 5 percentage-points in accuracy and 7 perplexity-points on a 45% model compression rate. We also show that RaNA not only preserves a theoretical advantage in the accuracy-FLOP trade-off but also practically in the accuracy-latency trade-off (Fig. 1b). Further, RaNA outperforms conventional neuron-adapters in GeLU-based Pythia models across multiple model sizes and compression levels (Figs. 1c, 4).

5.1 Experimental Setup

Here we describe the settings under which we run our experiments.

Models and Adapters: We conduct experiments using Llama2-7b, Gemma-2b, and multiple GPT-Neox models from the Pythia suite. In Sect. 5.3, we evaluate the following adapters, which are mainly implemented in PyTorch.

- *RaNA*: Our proposed adapter from Sect. 4.2.
- *CATS*: A state-of-the-art adapter for MLP layers leveraging SwiGLU activations. We refer to their work (Lee et al. (2024)) for more details.
- *SliceGPT*: A recent structured pruning method, which compresses linear layers by rotating and slicing them, including MLP and QKV layers. We refer to their work (Ashkboos et al. (2024)) for more details.
- *Neuron-Adaptive*: A standard neuron adapter for MLP layers with a small MLP masker, such as that leveraged by Zhang et al. (2024) or Liu et al. (2023), using 6% of MLP FLOPs for the masker, as done by Zhang et al. (2024).
- *Linear-Layer-Rank-Adapters with MLP maskers (LLRA)*: An adapter that applies Linear Layer Rank adapters leveraging an MLP-based masker (Sect. 4.1) to all linear layers in QKV and MLPs.

For output error assessments in Sect. 5.3, we additionally use a fixed low-rank singular value decomposition (SVD) comparison.

Datasets: We use the RedPajama (Computer (2023)) dataset for Llama2-7b and Gemma-2b, and the Pile (Gao et al. (2020)) dataset for Pythia models when evaluating rank contribution sparsity (Sect. 5.2), output errors (Sect. 5.3), and perplexity (Sect. 5.3), and for devising any data-dependent adapter component (e.g. A and B matrices in RaNA, the activation threshold in CATS and the slicing and rotating procedure of SliceGPT). Downstream-task performance is assessed using HellaSwag (Zellers et al. (2019)), PIQA (Bisk et al. (2019)), WinoGrande (Sakaguchi et al. (2019)), Arc-Easy (Clark et al. (2018)), Arc-Challenge (Clark et al. (2018)), RACE (Lai et al. (2017)) and OBQA (Mihaylov et al. (2018)).

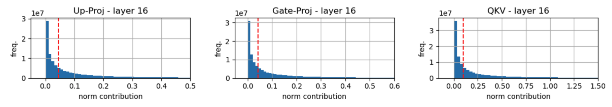
Fine-tuning: To assess accuracy and perplexity (Sect. 5.3), we fine-tune adapted models using the Huggingface library (Wolf et al. (2020)) and LoRA adapters (Hu et al. (2021)) for ∼31M tokens on Llama2-7b and Gemma-2b, with an AdamW optimizer, where learning rates were determined from various options, depending on the model's performance following 6M tokens of training. In a similar setup, Pythia models are fine-tuned for ∼61M tokens with the exception of not leveraging LoRA adapters. In addition, for SliceGPT, we leveraged a fixed 1e-5 learning rate and trained in float16 precision.

Performance Evaluations: We use LM-evaluation harnesses (Gao et al. (2023)) for downstream-task performance evaluation in a zero-shot setting for Llama2-7b and Pythia models, and a five-shot setting for Gemma-2b. Perplexity is measured on a held-out subset of each model's fine-tuning dataset. Further, FLOP compression is assessed by measuring the average FLOPs required to decode 512-token sequences.

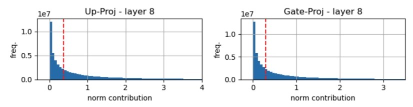
Latency Evaluations: For our latency evaluations, we leverage 100 sequences from the RedPajama dataset, where adapted models are timed in the task of decoding a sequence of 492 tokens with an initial context ranging from 1 to 1000 tokens. Evaluations are performed on an NVIDIA L40S GPU.

5.2 Rank Contribution Sparsity

A key property of adaptive compute methods is having sparse contributions from the pruned components to the module's output. In RaNA, we adaptively prune the ranks of the AB matrix in the decomposition $Wx \approx A(m(x) \odot Bx)$, which effectively means pruning the column vectors of the A matrix (Sect. 4.1). Concretely, we aim for sparsity in the contribution of each column vector of the A matrix to the output of the linear layer, which we obtain by measuring $(Bx)_i^2$. To measure this, we study the histograms of these contributions in Llama2-7b, Gemma-2b, and Pythia-160M models, shown in Figs. 2a, 2b. The distributions exhibit heavy tails with concentrations near zero, allowing us to mask out irrelevant values and retain the most impactful ones.



(a) Llama2-7b.



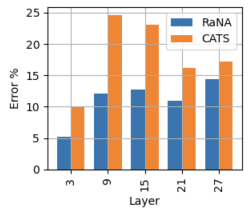
(b) Gemma-2b.

Figure 2: **The contribution of ranks in Linear Layer Rank Adapters is sparse for multiple layer types** (Sect. 5.2). Histograms outline the contribution of different column-vectors from the A matrix in the Linear Layer Rank Adapter decomposition $Wx \approx A(m(x) \odot Bx)$ to the original layers for Llama2-7b (left) and Gemma-2b (right). Red dashed line indicates a 50% sparsity threshold.

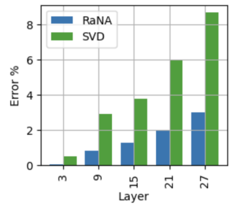
5.3 RaNA Evaluations

Output Errors: To assess the compression capabilities of RaNA, we apply it to the MLP and QKV layers of Llama2-7b, Gemma-2b and Pythia-160M. Further, we measure the error that RaNA and other adapters induce in these layers when adapting them to consume ∼50% of their FLOPs, as it is a common compression ratio in the pruning literature (Ma et al. (2023), Ashkboos et al. (2024)). For the MLPs, we measure the normalized error $\frac{|\text{MLP}(x) - \text{MLP}'(x)|_2^2}{|\text{MLP}(x)|_2^2}$, where MLP' is the adapted MLP; we do the analogous measurement for the QKV layers too. Intuitively, an effective adapter produces small errors, as that allows it to recover the model's original behavior better. Notably, from Figs. 3a, 3b, 3c, 3d, we can observe that RaNA attains the lowest error across all layers when compared to neuron-adapters and other adapter types. Concretely, in the case of Llama2-7b, RaNA attains an average error of 10.5%, while CATS attains an average error of 17.22% in MLP layers. Further, for Gemma-2b's MLPs, RaNA achieves an error of 2.18%, while CATS attains an error of 20.31%. Similarly, for Pythia-160M, RaNA achieves average errors of 7.87% and 0.36% across MLP and QKV layers respectively, while other approaches attain average errors of 15.23% and 0.36%. This demonstrates RaNA's capacity to effectively compress modern Transformer layers, which we further show translates into practical downstream task performance and perplexity improvements.

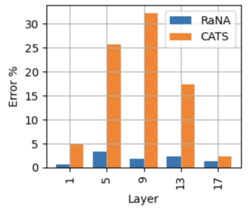
Downstream Task Performance and Perplexity: To examine the performance of RaNA adapters beyond their compression capabilities, we measure the perplexities and downstream task performance when applied to Llama2-7b, Gemma-2b, and a set of varied sized Pythia models. For these evaluations, we first apply the given adapter to the MLP and/or QKV layers of the model at hand, targeting a specific FLOP reduction ratio. Concretely, for Llama and Pythia models we apply RaNA to MLP and QKV layers, while for Gemma we only apply it to MLP layers. Further, we fine-tune



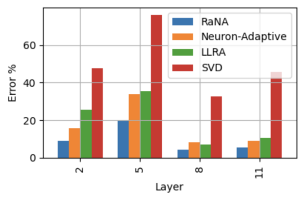
(a) Llama2-7b MLP layers.



(b) Llama2-7b QKV layers.



(c) Gemma-2b MLP layers.



(d) Pythia-160M MLP layers.

Figure 3: **RaNA adapters attain lowest errors on Transformer layers when replicating original layer outputs.** y-axis shows the error percentage; x-axis shows layer number. Errors induced by different adapters are compared when compressing layers of Llama2-7b, Gemma-2b and Pythia-160M to 50% FLOPs. RaNA attains the lowest error consistently across model layers (Sect. 5.3).

Method	FLOP Compression Rate	Arc Easy	Arc Challenge	HellaSwag	PIQA	RACE	WinoGrande	Avg Acc	PPL
Llama2-7b	0%	76.26%	43.52%	57.08%	78.07%	39.62%	69.14%	60.61%	6.39
RaNA	42%	68.81%	36.86%	51.96%	74.76%	39.71%	66.14%	**56.37%**	**8.04**
CATS	42%	56.31%	27.13%	41.01%	68.28%	35.12%	57.22%	47.51%	12.36
SliceGPT	42%	40.15%	21.84%	35.35%	59.85%	34.35%	55.49%	41.17%	18.39
RaNA	30%	72.85%	39.76%	54.63%	77.42%	40.48%	66.85%	**58.67%**	**7.29**
CATS	32%	69.49%	36.01%	53.16%	76.28%	38.76%	65.59%	56.55%	7.55
SliceGPT	31%	48.19%	26.19%	40.24%	65.07%	36.84%	59.59%	46.02%	13.95
RaNA	17%	73.86%	41.55%	56.09%	77.80%	39.52%	69.38%	59.70%	6.63
CATS	15%	75.17%	41.55%	56.96%	77.48%	39.52%	68.27%	**59.82%**	**6.37**
SliceGPT	17%	58.29%	31.31%	46.73%	70.73%	38.56%	63.22%	51.47%	10.65

Table 1: **RaNA outperforms neuron-adapters in perplexity and accuracy in Llama2-7b**. Perplexity is measured on ∼300K tokens of RedPajama. Average accuracy is aggregated over the listed benchmarks. The compression rate outlines the average FLOP compression rate for decoding a 512-token long sequence.

the adapted models. Finally, we measure their perplexity on a held-out subset of the dataset used for fine-tuning and measure their accuracies for multiple NLP benchmarks.

From Tabs. 1, 2, we observe that RaNA outperforms the state-of-the-art CATS adapters across a varied set of compression rates in SwiGLU based Transformers. Notably, not only does it outperforms when applied to MLP and QKV layers, as shown for Llama2-7b (Tab. 1), but it also does when only applied to the MLP layers as shown for Gemma-2b (Tab. 2). Notably, we opted for not adapting QKV layers in Gemma-2b for simplicity, as they constitute just a small proportion of FLOPs (∼5%) relative to the MLP layers. Particularly, for Llama2-7b, RaNA improves over neuron-adaptive methods by attaining 4 less perplexity-points and 8 more percentage-points in accuracy when reducing FLOPs by 42% on the overall model. Moreover, both RaNA and CATS outperform SliceGPT in both perplexity and downstream task performance (Tab. 1, Fig. 5) for all FLOP compression rates, highlighting the benefits of adaptive compression methods compared to static ones. Similarly, at a 45% compression, for Gemma-2b, RaNA achieves 7 less perplexity-points and 5 more percentage-points in accuracy than prior neuron adapters.

We attribute RaNA's strong performance to its notable compression capacity, its ability to more evenly distribute FLOPs across the Up-Project, Down-Project, and Gate-Project layers, and its direct applicability to QKV and MLP layers, all of which posed challenges for previous neuron adapters. In addition, from Figs. 1c and 4, we observe that RaNA applied to MLP and QKV layers outperforms neuron adapters across the varied sized set of GeLU based Pythia models in both average accuracy and perplexity. This highlights RaNA's capacity to be generally applicable to modern Transformer models with non-sparse activations, like SwiGLU in the case

Method	FLOP Compression Rate	Arc Easy	Arc Challenge	HellaSwag	PIQA	RACE	WinoGrande	Avg Acc	PPL
Gemma-2b	0%	59.47%	29.95%	46.90%	70.24%	36.17%	56.83%	49.93%	11.05
RaNA	44%	58.84%	29.18%	42.34%	69.31%	34.83%	54.46%	**48.16%**	**13.83**
CATS	47%	51.43%	24.57%	35.07%	65.94%	28.42%	52.64%	43.01%	21.02
RaNA	32%	61.74%	31.66%	45.38%	70.40%	35.89%	51.62%	**49.45%**	**11.74**
CATS	34%	59.55%	29.61%	44.51%	70.95%	34.55%	54.38%	48.92%	12.60
RaNA	19%	59.26%	29.78%	46.42%	69.31%	35.98%	55.01%	49.29%	**11.18**
CATS	20%	62.79%	32.94%	47.84%	72.25%	35.50%	55.41%	**51.12%**	11.41

Table 2: **RaNA outperforms neuron-adapters in perplexity and accuracy in Gemma-2b**. Perplexity is measured on ∼300K tokens of RedPajama. Average accuracy is aggregated over the listed benchmarks. The compression rate outlines the average FLOP compression rate for decoding a 512-token long sequence.

of Llama and Gemma or GeLU in the case of Pythia. Together, these results show RaNA's effectiveness beyond individual layer compression, demonstrating practical improvements over state-of-the-art neuron adaptive techniques in perplexity and downstream task evaluations. Further, we attribute RaNA's and CATS' strong performance compared to SliceGPT, a static structured pruning approach, to their adaptive nature. The adaptiveness in RaNA and CATS allows them to dynamically determine weight matrices on the fly as a function of the input, as opposed to always using static ones like conventional structured pruning approaches; in practice, we observe adaptiveness allows them to obtain a better performance-compute trade-off.

Notably, from Fig.1b, we can observe how RaNA is capable of realizing practical speedups for varied compression rates in the case of Llama2-7b, as well as attaining a better accuracy-latency trade off than previous neuron adapters. This positions RaNA as a practical adapter to speed up Transformer inference when adaptively compressing models.

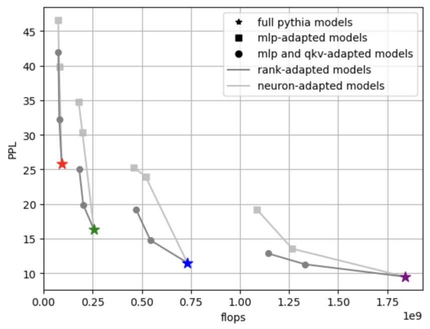


Figure 4: Perplexities are measured across adapted Pythia models, as a complementary measurement to accuracies outlined in Fig 1c. y-axis shows perplexity measured over 300K tokens of the Pile dataset (Sect. 5.1); x-axis shows average FLOPs for a forward pass with sequence length 512.

6 Conclusion and Future Work

We present the Adaptive Rank Allocation framework and RaNA adapters, designed to address the limitations of neuron-adaptive methods in modern Transformer architectures. By moving beyond neuron-based adaptation, RaNA is effectively applicable to both MLP and QKV layers, leveraging low-rank matrix decompositions and adaptive routers. Empirical results demonstrate that RaNA achieves greater performance over existing neuron-adaptive methods like CATS, providing improvements in perplexity and accuracy across benchmarks when reducing FLOPs. For instance, RaNA yields 4 fewer perplexity-points and an improvement of 8 percentage-points in accuracy for Llama2-7b when compressing FLOPs by 42%.

Future work seems promising. First, exploring the applicability of RaNA to other architectures, such as vision transformers or those leveraging different activation functions than the ones studied in this work could extend its impact. Additionally, investigating alternative matrix decomposition techniques and router configurations within Linear Layer Rank adapters could further improve RaNA's effectiveness. Finally, exploring a FLOP allocation strategy at the model level, rather than focusing solely on individual layers, presents a promising opportunity to improve RaNA's overall capacity.

REFERENCES

Abien Fred Agarap. Deep learning using rectified linear units (relu), 2019. URL https://arxiv.org/abs/1803.08375.

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns, 2024. URL https://arxiv.org/abs/2401.15024.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pp. 2397–2430. PMLR, 2023.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: reasoning about physical commonsense in natural language. *CoRR*, abs/1911.11641, 2019. URL http://arxiv.org/abs/1911.11641.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. GPT-NeoX-20B: An open-source autoregressive language model. In *Proceedings of the ACL Workshop on Challenges & Perspectives in Creating Large Language Models*, 2022. URL https://arxiv.org/abs/2204.06745.

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, 2018. URL https://arxiv.org/abs/1803.05457.

Together Computer. Redpajama: an open dataset for training large language models, 2023. URL https://github.com/togethercomputer/RedPajama-Data.

Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936. doi: 10.1007/BF02288367. URL https://doi.org/10.1007/BF02288367.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2019. URL https://arxiv.org/abs/1803.03635.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020. URL https://arxiv.org/abs/2101.00027.

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

Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey, 2021. URL https://arxiv.org/abs/2102.04906.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023. URL https://arxiv.org/abs/1606.08415.

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, 2021.

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

Sehoon Kim, Coleman Hooper, Thanakul Wattanawong, Minwoo Kang, Ruohan Yan, Hasan Genc, Grace Dinh, Qijing Huang, Kurt Keutzer, Michael W. Mahoney, Yakun Sophia Shao, and Amir Gholami. Full stack optimization of transformer inference: a survey, 2023. URL `https://arxiv.org/abs/2302.14017`.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations, 2017. URL `https://arxiv.org/abs/1704.04683`.

Je-Yong Lee, Donghyun Lee, Genghan Zhang, Mo Tiwari, and Azalia Mirhoseini. Cats: Contextually-aware thresholding for sparsity in large language models, 2024. URL `https://arxiv.org/abs/2404.08763`.

Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J. Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, and Sanjiv Kumar. The lazy neuron phenomenon: On emergence of activation sparsity in transformers, 2023a. URL `https://arxiv.org/abs/2210.06313`.

Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J. Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, and Sanjiv Kumar. The lazy neuron phenomenon: On emergence of activation sparsity in transformers, 2023b. URL `https://arxiv.org/abs/2210.06313`.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, and Beidi Chen. Deja vu: Contextual sparsity for efficient llms at inference time, 2023. URL `https://arxiv.org/abs/2310.17157`.

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

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering, 2018. URL `https://arxiv.org/abs/1809.02789`.

Iman Mirzadeh, Keivan Alizadeh, Sachin Mehta, Carlo C Del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. Relu strikes back: Exploiting activation sparsity in large language models, 2023. URL `https://arxiv.org/abs/2310.04564`.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference, 2022. URL `https://arxiv.org/abs/2211.05102`.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WINOGRANDE: an adversarial winograd schema challenge at scale. *CoRR*, abs/1907.10641, 2019. URL `http://arxiv.org/abs/1907.10641`.

Noam Shazeer. Glu variants improve transformer, 2020. URL `https://arxiv.org/abs/2002.05202`.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, Justin Mao-Jones, Katherine Lee, Kathy Yu, Katie Millican, Lars Lowe Sjoesund, Lisa Lee, Lucas Dixon, Machel Reid, Maciej Mikuła, Mateo Wirth, Michael Sharman, Nikolai Chinaev, Nithum Thain, Olivier Bachem, Oscar Chang, Oscar Wahltinez, Paige Bailey, Paul Michel, Petko Yotov, Rahma Chaabouni, Ramona Comanescu, Reena Jana, Rohan Anil, Ross McIlroy, Ruibo

Liu, Ryan Mullins, Samuel L Smith, Sebastian Borgeaud, Sertan Girgin, Sholto Douglas, Shree Pandya, Siamak Shakeri, Soham De, Ted Klimenko, Tom Hennigan, Vlad Feinberg, Wojciech Stokowiec, Yu hui Chen, Zafarali Ahmed, Zhitao Gong, Tris Warkentin, Ludovic Peran, Minh Giang, Clément Farabet, Oriol Vinyals, Jeff Dean, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Douglas Eck, Joelle Barral, Fernando Pereira, Eli Collins, Armand Joulin, Noah Fiedel, Evan Senter, Alek Andreev, and Kathleen Kenealy. Gemma: Open models based on gemini research and technology, 2024. URL `https://arxiv.org/abs/2403.08295`.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL `https://arxiv.org/abs/1706.03762`.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45. Association for Computational Linguistics, 2020. URL `https://aclanthology.org/2020.emnlp-demos.6`.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *CoRR*, abs/1905.07830, 2019. URL `http://arxiv.org/abs/1905.07830`.

Zhengyan Zhang, Yixin Song, Guanghui Yu, Xu Han, Yankai Lin, Chaojun Xiao, Chenyang Song, Zhiyuan Liu, Zeyu Mi, and Maosong Sun. Relu^2 wins: Discovering efficient activation functions for sparse llms, 2024. URL `https://arxiv.org/abs/2402.03804`.

A Appendix

A.1 Proof of Theorem 1

Here we prove Theorem 1.

Proof. Let M_r be a rank r matrix. The matrix M_r that obtains the lowest possible error ($\|WX - M_r\|_F^2$) is the rank-r singular value decomposition of WX, namely $M_r = U_r \Sigma_r V_r^T$. Now, solving for A_r, B_r:

$$A_r B_r X = M_r = U_r \Sigma_r V_r^T \implies A_r B_r = U_r \Sigma_r V_r^T X^+$$

where X^+ is the pseudo-inverse of X. Hence, we have found the optimal $A_r B_r$ for eqn. 7, namely:

$$A_r B_r = U_r \Sigma_r V_r^T X^+ = U_r (U_r^T W).$$

A.2 Algorithm

We provide a pseudocode implementation of RaNA below:

Algorithm 1 RANA Layer Compression

```
1: procedure CompressLayer(layer, decomposition, prune_ratio)
2:     // Find optimal rank and thresholds for compression
3:     $rank, threshold \leftarrow$ ComputeOptimalParameters$(decomposition, prune\_ratio)$
4:     // Create compressed layer using low-rank approximation
5:     $compressed\_layer \leftarrow$ DecomposeToRankN$(layer, rank)$
6:     // Apply adaptive thresholding for dynamic sparsity
7:     $masked\_layer \leftarrow$ ApplyThresholdMask$(compressed\_layer, threshold)$
8:     return $masked\_layer$
9: end procedure
```

Algorithm 2 RANA MLP Transformation

```
1: procedure TransformMLP(mlp, input_data, prune_ratios)
2:     // Transform each component with different pruning ratios
3:     $up\_masked \leftarrow$ CompressLayer$(mlp.up\_proj, prune\_ratios.up)$
4:     $gate\_masked \leftarrow$ CompressLayer$(mlp.gate\_proj, prune\_ratios.gate)$
5:     // Compute activation threshold for downstream pruning
6:     $act\_threshold \leftarrow$ ComputeActivationThreshold$(mlp, input\_data, prune\_ratios.down)$
7:     // Construct efficient MLP with dynamic sparsity
8:     $efficient\_mlp \leftarrow$ CreateDynamicMLP$(up\_masked, gate\_masked, mlp.down\_proj, act\_threshold)$
9:     return $efficient\_mlp$
10: end procedure
```

Algorithm 3 RANA Forward Pass

```
1: procedure ForwardPass(input, compressed_layer)
2:     // Dynamic pruning based on activation magnitudes
3:     $activations \leftarrow$ ComputeActivations$(input)$
4:     $pruned\_activs \leftarrow$ ApplyDynamicMask$(activations)$
5:     // Efficient forward computation using compressed weights
6:     $output \leftarrow$ ComputeOutput$(pruned\_activs)$
7:     return $output$
8: end procedure
```

A.3 Ablations and Additional Results

Compressing MLPs only vs. MLPs + QKVs:

As an ablation study, we look at how compressing both MLPs and QKVs compare to only compressing MLPs to the same FLOP compression ratio in Llama2-7b. In addition, we study how our FLOP-allocation algorithm for MLP layers impacts performance by evaluating an adapted model that leverages the FLOP allocation procedure in the MLP layers and evaluating the same adapted model with the exception that it uniformly allocates FLOPs across the components of each individual MLP layer.

Concretely, we compress three Llama2-7b models by ∼31% of their FLOPs and evaluate their perplexity on ∼300K tokens of the RedPajama dataset, without fine-tuning. First, we look at a vanilla RaNA adapted model, which leverages the FLOP allocation algorithm at the MLP level and that also compresses QKV layers. Second, we look at a model with the same setup, except without compressing QKV layers, but still compressing ∼31% of the overall total FLOPs. Third, we look at a model that compresses both QKV and MLP layers but does not leverage the FLOP allocation procedure for MLP layers.

Model Version	FLOP Compression Rate	PPL
Llama2-7b - MLP + QKV + FLOP Allocation	31%	8.40
Llama2-7b - MLP + FLOP Allocation	31%	8.79
Llama2-7b - MLP + QKV (No FLOP Allocation)	32%	9.10

Table 3: Perplexity Evaluation for Different RaNA Settings.

As we can observe in Tab. 3, the best model is the one combining both MLP + QKV compression and the FLOP allocation procedure in the MLPs, while the two other versions fall behind in perplexity.

Accuracy-FLOPs tradeoff including SliceGPT:
Here, in Fig. 5, we include the SliceGPT curve for the Accuracy-FLOPs trade-off in Llama2-7b.

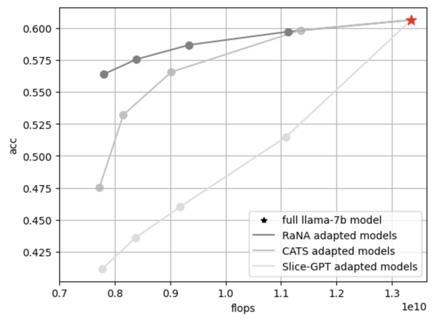


Figure 5: Llama2-7b Accuracy v.s. FLOPs.

A.4 FLOP Compression Breakdown

Here we leave in Tab. 4 the FLOP compression breakdown across MLP and QKV layers of the main adapted models.

Model	Total FLOP Compression	MLP FLOP Compression	QKV FLOP Compression
Gemma-2b-RaNA	44%	61%	0%
Gemma-2b-CATS	47%	65%	0%
Gemma-2b-RaNA	32%	45%	0%
Gemma-2b-CATS	34%	48%	0%
Gemma-2b-RaNA	19%	27%	0%
Gemma-2b-CATS	20%	28%	0%
Llama2-7b-RaNA	42%	47%	46%
Llama2-7b-CATS	42%	65%	0%
Llama2-7b-RaNA	30%	34%	33%
Llama2-7b-CATS	32%	50%	0%
Llama2-7b-RaNA	17%	19%	18%
Llama2-7b-CATS	15%	23%	0%

Table 4: FLOP Compression Comparison for Different Models. Total, MLP, and QKV FLOP compression rates are reported for the Gemma-2b and Llama2-7b models with RaNA and CATS adapters.