UNIVERSITY OF CALIFORNIA,
IRVINE

Compressing Language Models using Low-Rank Decomposition and Characterizing the Accuracy - Efficiency Trade-offs

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Electrical and Computer Engineering

by

Chakshu Moar

Thesis Committee:
Assistant Professor Hyoukjun Kwon, Chair
Assistant Professor Sitao Huang
Assistant Professor Yanning Shen

2024

DEDICATION

I dedicate this thesis to my parents and my brother for their endless love, support and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I am incredibly grateful to Professor Hyoukjun Kwon, my advisor and thesis committee chair, for his invaluable guidance and unwavering support throughout this project. His expertise laid the foundation for my understanding of the subject, and his constant encouragement motivated me during challenging times.

My sincere thanks go to Michael Pellauer, Research Scientist at NVIDIA, for collaborating with me and steering the direction of this research. His insightful advice on problem selection, methodology, and appropriate tools proved invaluable throughout the process.

I am also grateful to my thesis committee members, Professor Sitao Huang and Professor Yanning Shen, for taking the time to review my thesis report and providing valuable feedback and insightful suggestions.

I extend my gratitude to my fellow researchers in the ISA Lab (led by Professor Hyoukjun Kwon) for their valuable feedback on my research. I also acknowledge the University of California, Irvine, for providing the resources necessary to conduct this research.

Finally, I express my deepest gratitude to my friends and family for their unwavering support and encouragement throughout my journey. This accomplishment would not have been possible without them.

ABSTRACT OF THE THESIS

Compressing Language Models using Low-Rank Decomposition and Characterizing the Accuracy - Efficiency Trade-offs

By

Chakshu Moar

Master of Science in Electrical and Computer Engineering

University of California, Irvine, 2024

Assistant Professor Hyoukjun Kwon, Chair

Large language models (LLMs) have emerged and presented their general problem-solving capabilities with one model. However, the model size has increased dramatically with billions of parameters to enable such broad problem-solving capabilities. In addition, due to the dominance of matrix-matrix and matrix-vector multiplications in LLMs, the compute-to-model size ratio is significantly lower than that of convolutional neural networks (CNNs). This shift pushes LLMs from a computation-bound regime to a memory-bound regime. Therefore, optimizing the memory footprint and traffic is an important optimization direction for LLMs today.

Model compression methods such as quantization and parameter pruning have been actively explored for achieving the memory footprint and traffic optimization. However, the accuracy-efficiency trade-off of rank pruning (i.e., low-rank decomposition) for LLMs is not well-understood yet. Therefore, in this work, we characterize the accuracy-efficiency trade-off of a low-rank decomposition method, Tucker decomposition, on recent language models including an open-source LLM, Llama 2.

We formalize the low-rank decomposition design space and show that the decomposition design space is huge (e.g., $O(2^{37})$ for Llama2-7B). To navigate such a huge design space,

we characterize the design space and prune ineffective design space utilizing the learning from our characterization results (e.g., we can reduce the pruned ranks to 1 without a noticeable model accuracy drop). On the pruned design space, we perform thorough case studies of accuracy-efficiency trade-offs using six widely used LLM benchmarks on BERT and Llama 2 models. Our results show that we can achieve a 9% model size reduction with minimal accuracy drops, which range from 4%p to 10%p, depending on the difficulty of the benchmark, without any retraining to recover accuracy after decomposition. The results show that low-rank decomposition can be a promising direction for LLM-based applications that require real-time service in scale (e.g., AI agent assist and real-time coding assistant), where the latency is as important as the model accuracy.

Chapter 1

Introduction

Large language models (LLMs) such as GPT-4 [2] have opened a new era of artificial intelligence (AI) technologies, based on broad problem-solving capabilities and even encompassing generative tasks [3] interfaced with natural languages. Natural language-based interfaces provide intuitive access for users to the latest AI technology and open a plethora of automation opportunities (e.g., code generation [4], AI agent [5], etc.). Such a success was mainly driven by a massive amount of training data [2], which leads to a large number of model parameters to learn effectively. The number of parameters in the state-of-the-art models reaches up to 70 billion parameters on a popular open LLM, Llama 2 [1], which translates to 140 GB of memory in FP16 data format just for model parameters (i.e., weights). Such a large memory requirement is beyond typical on-board memory sizes in a single GPU (e.g., 80GB in NVIDIA A100 and H100), which presents a major challenge for providing services like ChatGPT at scale.

Although LLM variants in smaller scales (e.g., Llama2-7B [1]) exist, their memory requirements are still high compared to those of previously popular convolutional neural networks (CNNs). For example, Llama 2-7B has 268.5 $\times$ more parameters compared to ResNet50 [6]. As an additional challenge, these rising footprints have actually been paired with decreased

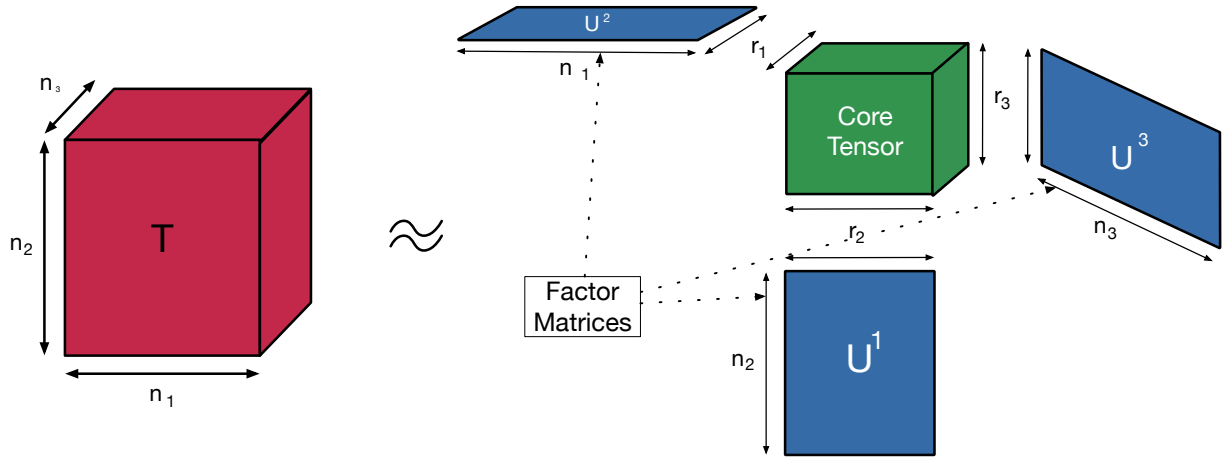


Figure 1.1: An illustration of Tucker Decomposition. A three-dimensional tensor T can be decomposed into one core tensor and three factor matrices, U^1, U^2, and U^3. the dimension of the core tensor corresponds to the rank of the decomposition.

data reuse compared to CNNs. This is because state-of-the-art LLMs are based on the Transformer [7] architecture, and its operators have a significantly low compute-to-model size ratio, as summarized in Table 1.1. This low compute-to-model size ratio and large memory footprint together indicate that optimizations for LLM inferences need to focus on the memory side, rather than focusing on increasing peak throughput.

As a result, recent research has been exploring various methodologies for reducing memory footprint and traffic, which includes sparsity [8] and quantization [9, 10, 11]. Among such approaches, low-rank decomposition (or factorization) is an emerging approach that performs a dimensionality analysis of a tensor and prunes minor components in the decomposed dimensions (i.e., prunes the rank of a tensor). One of the low-rank decomposition methodologies, Tucker decomposition, can be seen as a generalized approach of principal component analysis (PCA) for high-dimensional tensors [12].

As illustrated in Figure 1.1, low-rank Tucker decomposition method decomposes a tensor into a series of tensor contractions (or matrix multiplications if the base tensors are two-dimensional). When performing the conversion, we prune the rank of the decomposed tensors by removing unimportant dimensions similar to the dimension reduction methods

based on PCA. The pruned ranks lead to smaller memory requirements compared to the non-decomposed tensor, but this also leads to approximate tensor reconstruction, which has an impact on task performance [1] (e.g., accuracy).

The application of low-rank decomposition has been actively explored in the computer vision domain [13, 14, 15, 16]. Based on such successful application cases, researchers are exploring the case on large language models [17]. However, unlike weight pruning (sparsity) and quantization (data precision), the trade off among task performance (e.g., model accuracy), computational performance (e.g., latency), and energy efficiency of low-rank decomposition targeting large language models is not yet well-understood. In addition, we demonstrate that low-rank decomposition has a large design space originating from many possible choices (e.g., the number of pruned ranks, the choice of decomposed layers and tensors, and so on), so understanding the trade-off is a difficult task. For example, when we apply Tucker decomposition on the Llama2-7B model, our design-space formulation (Section 3.1) reveals that there exist $\mathbf{O}(\mathbf{2^{37}})$ possible ways of applying Tucker decomposition, even if we apply the same set of tensors with the same pruned ranks across all the layers.

Therefore, in this work, we first formalize the design space of the low-rank decomposition on recent language models based on Transformers, then characterize the trade-off space among task performance, computational performance, and energy efficiency. We perform a thorough profiling of the latency, energy, accuracy, and memory usage for running Llama2 and BERT after applying Tucker decomposition on 4× NVIDIA A100 GPUs with 80GB of memory for each. We measure the metrics on five broadly adopted benchmarks for LLMs: AI2 Reasoning Challenge (ARC) [18], HellaSwag [19], Massive Multitask Language Understanding (MMLU) [20], TruthfulQA [21], WinoGrande [22]. The benchmarks include a variety of tasks oriented for LLMs, which includes reasoning, truthfulness check, sentence completion,

[1] Performance is an overloaded term by the ML algorithm and hardware/system communities. To distinguish them, we refer to the quality of the outputs of the model as task performance, and the speed of the execution to be the computational performance.

Table 1.1: The model size and the number of computations (multiply-and-accumulate; MAC) of Transformer-based language models and convolutional neural network-based computer vision models. Language model data are based on the batch size of 1 and the sequence length of 128.

Models	Model Type	Model Size (FP16)	# Computations (MACs)	Compute to model size ratio
ResNet50 [6]	Computer Vision	51.1 MB	8.21 B	160.7
BERT-Base [25]	Language Model	219.0 MB	11.2 B	51.1
Llama2-7B [1]	Large Language Model	13.4 GB	850.0 B	63.4

commonsense reasoning, etc. In our case studies, we show that low-rank decomposition can reduce the model size by 9% without losing considerable model accuracy (4.5% points to 10% points). In addition to the model size reduction, we observe a considerable reduction in the end-to-end latency by 4% and energy by 5%, which is related to the lower compute-to-model size ratios as presented in Table 1.1. As part of this contribution, we study how to apply low-rank decomposition to the language models, performing a thorough characterization on the number of pruned ranks and decomposition location (layer and weight tensors within an attention block), showing that we can aggressively reduce the rank to be one without losing significant accuracy and should carefully select the decomposition location to minimize the accuracy degradation. Finally, we present a trade-off study between the model size reduction and resulting metrics (latency, energy, etc.). The results show that we can reduce the model size by 9% without losing considerable accuracy (on average, 10% loss of accuracy), which provides 4% latency and 5% energy savings. Our case study results show that low-rank factorization is a promising option to enable low-cost LLM-based services, such as Virtual Agent [5], real-time animation generation [3], real-time coding assistant [23], AI assistant [24], and so on.

We summarize our contributions as follows:

- We thoroughly explore the accuracy/performance trade-off of low-rank decomposition on recent language models including a large language model, Llama 2.
- We demystify the design space of low-rank decomposition on large language models by

formally defining its dimensions.

- Beyond the simple performance analysis, we also profile energy consumption and show the low-rank decomposition is an effective approach to enhancing energy efficiency.
- We analyze the sensitivity of low-rank decomposition and provide insights on how to best apply low-rank decomposition to language models.

Chapter 2

Background and Motivation

2.1 Low-Rank Decomposition

Tensor Decomposition. We briefly introduce tensor decomposition, specifically, Tucker Decomposition [26], and how we apply it to compress LLMs. Tucker Decomposition decomposes a tensor into smaller core tensor and a set of matrices equal to the order of the tensor. As shown in Figure 1.1, for a 3^{rd} order tensor $T \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, Tucker Decomposition is computed as

$$T \approx \Gamma \times_1 U^1 \times_2 U^2 \times_3 U^3 = K$$

where $\Gamma \in \mathbb{R}^{r_1 \times r_2 \times r_3}$, and $U^1,\ U^2,\ U^3$ belong to $\mathbb{R}^{r_1 \times n_1}, \mathbb{R}^{r_2 \times n_2}, \mathbb{R}^{r_3 \times n_3}$ respectively. Here, $r_1,\ r_2,\ r_3$ represent the decomposition rank of tensor T. The i mode product, $i = 1,\ 2,\ 3$, of the core tensor Γ and the factor matrices $U^{1,\ 2,\ 3}$ is defined as:

$$(\Gamma \times_1 U^1)(n_1, r_2, r_3) = \sum_{i_1=1}^{r_1} \Gamma(i_1, r_2, r_3) U^1(i_1, n_1)$$

$$(\Gamma \times_1 U^2)(r_1, n_2, r_3) = \sum_{i_2=1}^{r_2} \Gamma(r_1, i_2, r_3) U^1(i_2, n_2)$$

$$(\Gamma \times_1 U^2)(r_1, r_2, n_3) = \sum_{i_3=1}^{r_3} \Gamma(r_1, r_2, i_3) U^1(i_3, n_3)$$

Since tensor decomposition approximates the original tensor, the error between the original tensor T and the reconstructed tensor K depends on the decomposition rank $r_1,\ r_2,\ r_3$. For a given set of decomposition ranks, the relative error between the original and the reconstructed tensors satisfies

$$\left\| T - (\Gamma \times_1 U^1 \times_2 U^2 \times_3 U^3) \right\| \leq \epsilon \left\| T \right\|$$

where $\|T\|$ is the *norm* of T. The goal of Tucker-decomposition is to minimize ϵ, and it can be formulated as

$$\underset{\Gamma, U^1, U^2, U^3}{\arg\min} \left\| T - (\Gamma \times_1 U^1 \times_2 U^2 \times_3 U^3) \right\|$$

Algorithm 1: Tucker Decomposition via Higher Order Orthogonal Iteration (HOI)

Input : input tensor T, decomposition rank $(r_1,\ r_2,\ r_3)$
Output : core tensor Γ, factor matrices $U^1,\ U^2,\ U^3$
Initialize $U^2,\ U^3$ with orthonormal columns
while *convergence criteria not met* **do**
  $P = T \times_2 (U^2)^T \times_3 (U^3)^T$
  $U^1 = SVD(r_1, P_{(1)})$
  $Q = T \times_1 (U^1)^T \times_3 (U^3)^T$
  $U^2 = SVD(r_2, Q_{(2)})$
  $R = T \times_1 (U^1)^T \times_2 (U^2)^T$
  $U^3 = SVD(r_3, R_{(3)})$
end while
$\Gamma = T \times_1 (U^1)^T \times_2 (U^2)^T \times_3 (U^3)^T$
Return $\Gamma,\ U^1,\ U^2,\ U^3$
`/* Here, SVD means Singular Value Decomposition, and A = SVD(k, B)`
`means compute` k^{th}`-order truncated SVD of` B `and set` $A = [a_1, a_2, ..., a_k]$`,`
`where` $a_1, a_2, ..., a_k$ `are the` k `largest left singular vectors of` B `*/`

Algorithm 1 describes how we can compute the core tensor and the factor matrices using Higher Order Orthogonal Iteration (HOI). HOI is an iterative algorithm to approximate the

factor matrices, while it can be shown [27] that the optimal core tensor Γ can be computed as

$$\Gamma = T \times_1 (U^1)^T \times_2 (U^2)^T \times_3 (U^3)^T$$

Generally, a higher decomposition rank results in a better approximation. While the lower bound of $r_1,\ r_2,\ r_3$ is 1, the upper bound is usually taken as $r_i = n_i,\ i = 1, 2, 3$ for a good approximation. In our experiments, we prune the decomposition rank $r_1 = r_2 = r_3 \in [1, \min(n_1, n_2, n_3)]$.

2.2 Challenges for LLMs

Low compute-to-model size ratio in LLMs. We analyze the BERT-Base model with 110 million parameters fine-tuned on SQuAD dataset [28], and Llama-2-7B model with 7 billion parameters fine-tuned for chat. Self-Attention is a key operator in Transformer-based LLMs. The linear and batched matrix multiplication (BMM) are major operators in the self-attention module, which accounts for the majority of the model's memory footprint. The multi-layer perceptron (MLP) layer in Llama and the intermediate/output layers in BERT are also based on matrix multiplication. Overall, over 80% of the model parameters in BERT and over 96% of the model parameters in Llama-2 are present in the encoder and decoder layers respectively, and most of them are used for variants of matrix multiplications (e.g., linear and BMM). Because of the relatively low dimensionality of such operators compared to convolutions, which reduces the data reuse across different dimensions and effectively lowers the number of computations for each data, the overall computation-to-model size ratio of language models is lower than that of convolutional neural networks, as shown in Table 1.1. Therefore, Transformer-based language model workloads tend to be in the memory-bound region in the roof-line model [29], which motivates optimizations focusing on the memory footprint and traffic.

2.3 Understanding Tucker Decomposition in Language Models

Low-Rank Decomposition is a promising technique for compressing LLMs and increasing their arithmetic intensity. Tucker Decomposition can be viewed as a generalized case of Singular Value Decomposition (SVD) [12] with pruned low-rank factor matrices to approximate the original matrix. Therefore, second-order Tucker Decomposition can be computed as multiplication of core and factor matrices:

$$T(n_1, n_2) \approx U^1(n_1, pr) \times \Gamma(pr, pr) \times U^2(pr, n_2) = K$$

where $pr = pruned\ rank$.

As shown in Figure 3.1(c), a 2D tensor $T \in R^{H \times W}$ can be approximated by the product of a core tensor and two factor tensors. The core tensor $C \in R^{PR \times PR}$, where PR is the pruned rank, while the factor tensors $A \in R^{H \times PR}$ and $C \in R^{W \times PR}$. If $PR \ll \min(H,\ W)$, or $PR < (\frac{\sqrt{(H+W)^2+4\times H\times W}-(H+W)}{2})$ more specifically, the total number of parameters reduce because of decomposition, and the compression ratio can be computed as:

$$Number\ of\ parameters\ before\ decomposition = H \times W$$

$$Number\ of\ parameters\ after\ decomposition = H \times PR + PR \times PR + PR \times W$$

$$Compression\ Ratio = \frac{H \times W}{H \times PR + PR \times PR + PR \times W}$$

We apply tucker decomposition to the weights of a pre-trained LLM and decompose a fully-connected layer into three smaller fully-connected layers. With a pruned rank much smaller than the dimension of the weight tensor, we can significantly reduce the size of the model. As we reduce the total number of parameters, the overall number of computations decreases

thereby improving the inference latency. However, the design space of low-rank decomposition is huge, and therefore we need a formal methodology to perform low-rank decomposition.

Chapter 3

Decomposition Design Space Formalization and Characterization

As discussed in Section 2.3, Tucker decomposition with rank pruning reduces the computational and memory overhead. However, when applying Tucker decomposition to a language model, we need to consider the impact on the model's accuracy. That is, the model accuracy degradation needs to be confined within an acceptable range, while maximizing latency and energy benefits. To summarize the optimization goal, we formalize the goal assuming equal importance on latency and energy (i.e., targeting energy-delay-product as the objective metric) as follows:

Definition 3.1. ***Design Goal of low-rank decomposition***

Given an accuracy threshold τ, find a decomposition configuration γ such that

$$\underset{\gamma: max(Accuracy_{Original} - Accuracy(\gamma), 0) < \tau}{\arg\min} Latency(\gamma) \times Energy(\gamma)$$

In Definition 3.1, τ refers to an accuracy drop tolerance (e.g., 3%). The accuracy condition restricts the accuracy loss but allows potential accuracy gains. The optimized variable, γ,

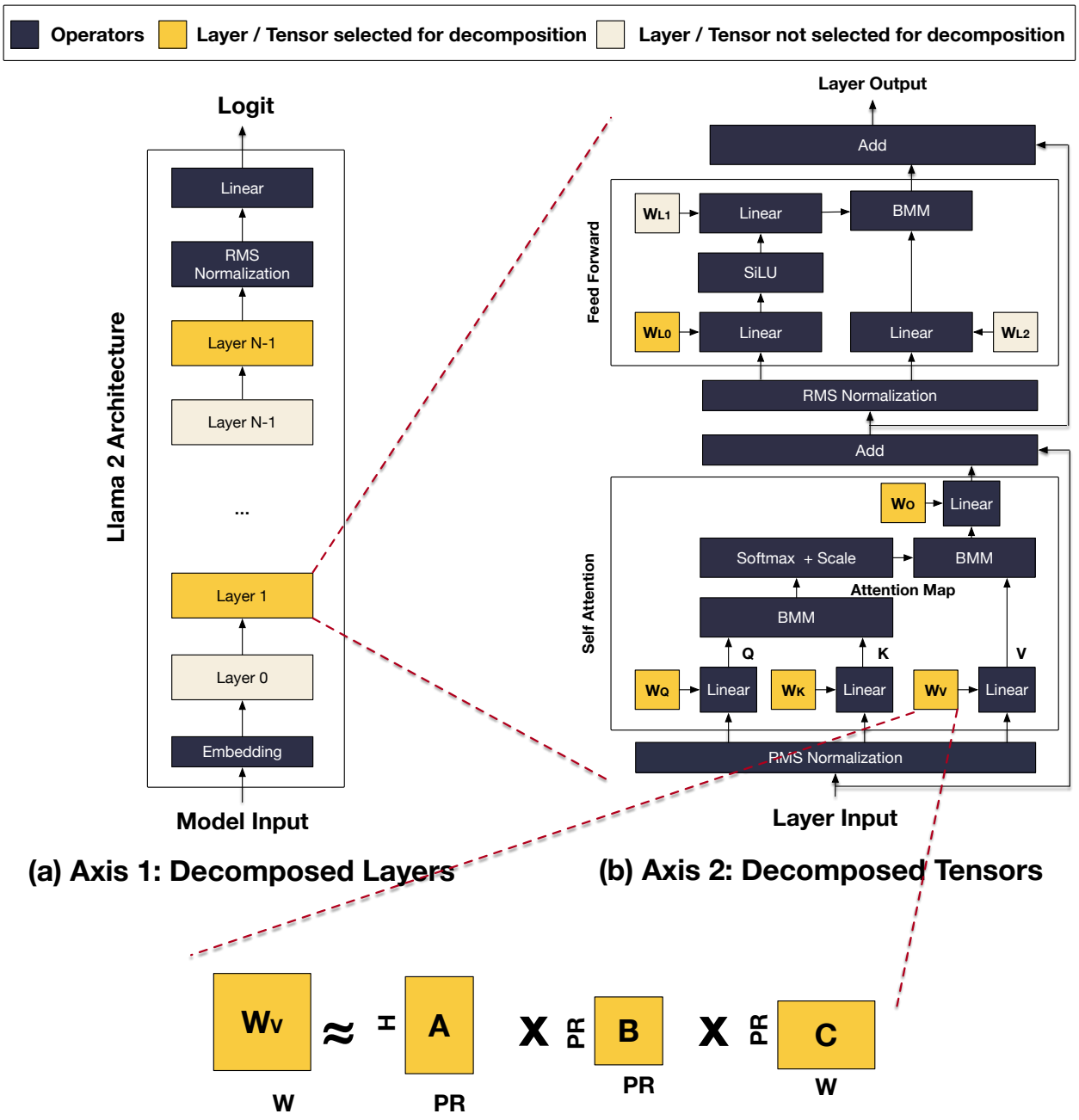


Figure 3.1: An illustration of three axes of the decomposition configurations discussed in Definition 3.4. (a): Choice of the layers to decompose, (b): Choice of tensors within each layer to decompose, (c): The choice of pruned rank (PR) to be used for each decomposed tensor.

refers to how we decompose a language model, which consists of three choices illustrated in Figure 3.1. (We discuss the formal definition of γ later in Definition 3.4.) γ enables us to describe any partial decomposition configuration of a model, which helps to navigate the

accuracy-efficiency trade-off.

However, because recent language models are complex and large, the number of possible ways of decomposition (i.e., decomposition design space) is massive (e.g., $\mathbf{O(2^{37})}$ ways for the smallest Llama 2 variant, Llama2 - 7B). Therefore, we first formalize the decomposition design space and perform a characterization of the design space to identify effective and ineffective decomposition configurations. Based on the characterization results, we prune the ineffective decomposition configurations and reduce the decomposition design space to a tractable size (e.g., for an LLM, Llama2-7B, the design space is reduced from $O(2^{37})$ to O(32).

3.1 Decomposition Space

Because of the uniformity of the building blocks (i.e., the same block is repeated) in language models, as illustrated in Figure 3.1 (a), we target homogeneous decomposition schemes for each layer. That is, we prune the same number of ranks and select the same set of weight tensors to be decomposed within each layer. Combined with those two choices (number of ranks after pruning and tensors to be decomposed), describing the layers to be decomposed provides a complete description of one decomposition configuration. We formalize the decomposition configuration and design space next to provide a precise definition of them.

Definition 3.2. ***Decomposed Layers and Tensors***
For a given model m, which has $N_{Layers}(m)$ layers and $N_{Tensors}(m)$ of decomposable weight tensors in each layer, the choices of decomposed layers $Decomp_{Layers}(m)$) and tensors ($Decomp_{Tensors}(m)$) are defined as follows:

$$Decomp_{Layers}(m) = \{DL_0, DL_1, ...DL_L\}$$

$$Decomp_{Tensors}(m) = \{DT_0, DT_1, ...DT_K\}$$

where

$$(L, K \in \mathbb{Z}) \wedge (0 \leq L < N_{Layers}(m)) \wedge (0 \leq K < N_{Tensors}(m))\square$$

In Definition 3.2, we describe the choice of decomposed layers and tensors as sets of corresponding layers and tensor IDs represented in integers. If the K and L are set to zeros, corresponding $Decomp_{Layers}$ and $Decomp_{Tensors}$ become empty sets, which allows to express the original model without any decomposition.

Definition 3.3. ***Pruned ranks***

For a given model m, the rank after pruning (or pruned ranks), $PR(m)$ is defined as follows:

$$\begin{aligned} PR(m) = \{(l, k, p) \mid (l, k, p \in \mathbb{Z}) \wedge (0 \leq k < N_{Layers}(m)) \\ \wedge (0 \leq l < N_{Tensors}(m)) \wedge (0 < p \leq rank(l, k))\} \end{aligned}$$

where $rank(l, k)$ refers to the rank of a weight tensor k in layer l. $\square$

The formulation in Definition 3.3 indicates that the pruned rank (the rank after pruning) cannot exceed the original rank. Also, Definition 3.3 allows us to describe the decomposition without rank pruning by setting the pruned rank (p) the same as the original rank. Using Definition 3.2 and Definition 3.3, we define a complete low-rank decomposition configuration as follows.

Definition 3.4. ***Low-rank Decomposition Configuration(γ)***

For a given model m, which has $N_{Layers}(m)$ layers, $N_{Tensors}(m)$ of decomposable weight tensors in each layer, and $Dim(m, ID_{Layer}, ID_{Tensor})$ dimensions, a decomposition configuration for model m ($\gamma(m)$) is defined as follows:

$$\gamma(m) = (PR(m), Decomp_{Layers}(m), Decomp_{Tensors}(m))\square$$

In Definition 3.4, we define the decomposition configuration as a tuple of N_{PR}, $Decomp_{Layers}$, and $Decomp_{Tensors}$, which are defined in Definition 3.3 and Definition 3.2. The tuple captures the three major decomposition axes illustrated in Figure 3.1. Before we define the low-rank decomposition design space, we first define the validity of a decomposition configuration.

Proposition 3.1. ***Validity of a Decomposition Configuration(***$Val(\gamma)$***)***

For a given model m, which has $N_{Layers}(m)$ layers, $N_{Tensors}(m)$ of decomposable weight tensors in each layer, and $Dim_{Min}(m)$ to be the smallest weight matrix dimension (i.e., number of columns), a decomposition configuration for model m ($\gamma(m) = (PR(m), Decomp_{Layers}(m), Decomp_{Tensors}(m))$), $\gamma(m)$ is valid if the following conditions are met:

$$\forall (l, k, p) \in PR(m), l \in Decomp_{Layers}(m) \wedge k \in Decomp_{Tensors}(m)$$
$$\wedge |PR(m)| = (|Decomp_{Layers}| - 1) \times (|Decomp_{Tensors}| - 1) + 1 \square$$

Because the individual validity of PR, $Decomp_{Layers}$, and $Decomp_{Tensors}$ are checked in their definitions in Definition 3.3 and Definition 3.2, we need to check the validity as their combination. Note that $Decomp_{Layers}$ and $Decomp_{Tensors}$ are independent, based on their definition in Definition 3.2 (i.e., selection of the decomposed layers and tensors within each decomposed layer is independent). However, the definition of the pruned ranks in Definition 3.3 contains the layer and tensor IDs within its definition. Therefore, we need to make sure the pruned ranks cover all the decomposed (layer, tensor) combinations, and Proposition 3.1 states that condition.

Using the definitions and proposition, we can define the decomposition design space as follows.

Definition 3.5. ***Low-rank Decomposition Design Space (***S_{LR}***)***

For a given model m, the decomposition design space($S_{LR}(m)$) is defined as follows:

$$S_{LR}(m) = \{\gamma_i \mid Val(\gamma_i) \wedge i \in \mathbb{Z} \wedge i \geq 0\}\square$$

Definition 3.5 states that the decomposition design space is a set of all valid decomposition configurations. One useful property of the design space for understanding the complexity is the scale of the design space. We discuss how we can obtain a little-o illustration.

Theorem 3.1. ***The size of the Decomposition Design Space ($|S_{LR}|$)***
For a given model m and its decomposition design space($S_{LR}(m)$),

$$|S_{LR}(m)| = (2^{N_{Tensors}(m)} - 1) \times (2^{N_{Layers}(m)} - 1) \times rank(l,k) + 1$$

Proof. The size of the decomposition design space $|S_{LR}(m)|$ is the number of elements within $S_{LR}(m)$. Because $S_{LR}(m)$ is a set of valid decomposition configurations γ, we count all the possible $\gamma = (PR, Decomp_{Layers}, Decomp_{Tensors})$.

(1) The number of possible choices for pruned ranks ($|PR|$)

Based on Definition 3.3, $\forall(l,k,p) \in PR(m), p \leq rank(l,k)$. That is, the number of available choices for p is $rank(l,k)$.

(2) The number of possible choices for decomposed layers

We can select to decompose 0 to $N_{Layers}(m)$ layers. However, in addition to the total number of decomposed layers, we need to specify which layers are decomposed (e.g., decomposing two layers, we can select layers 0 and 1, 1 and 2, 0 and 3, etc.), which can be represented using combinations. Therefore, the number of possible choices is as follows:

$$\Sigma_{l=0}^{N_{Layers}(m)} \binom{N_{Layers}(m)}{l}$$

(3) The number of possible choices for decomposed tensors

Following the same method as (2), the number of possible choices is represented as follows:

$$\Sigma_{k=0}^{N_{Tensors}(m)} \binom{N_{Tensors}(m)}{k}$$

(4) The number of valid combinations of decomposed layers and tensors

All possible combinations of decomposed layers and tensors can be counted as (2) × (3):

$$All\ Combinations = \Sigma_{l=0}^{N_{Layers}(m)} \binom{N_{Layers}(m)}{l} \Sigma_{k=0}^{N_{Tensors}(m)} \binom{N_{Tensors}(m)}{k}$$

However, the above equation counts the cases where the model is not decomposed ($l = 0$ or $k = 0$) multiple times, while they should only be counted once. Therefore, all the valid combinations of decomposed layers and tensors are counted as:

$$Valid\ Combinations = \Sigma_{l=1}^{N_{Layers}(m)} \binom{N_{Layers}(m)}{l} \Sigma_{k=1}^{N_{Tensors}(m)} \binom{N_{Tensors}(m)}{k} + 1$$

Because we count all the possible and valid $\gamma = (PR, Decomp_{Layers}, Decomp_{Tensors})$ (i.e., all the possible combinations of (1) and (4)),

$$\begin{aligned} |S_{LR}(m)| &= (4) \times (1) \\ &= \left(\Sigma_{l=1}^{N_{Layers}(m)} \binom{N_{Layers}(m)}{l} \Sigma_{k=1}^{N_{Tensors}(m)} \binom{N_{Tensors}(m)}{k} + 1 \right) rank(l,k) \\ &= (2^{N_{Layers}(m)} - 1) \times (2^{N_{Tensors}(m)} - 1) \times rank(l,k) + 1 \end{aligned}$$

where, $rank(l, k)$ is the pruned target rank for a uniform decomposition of all tensors.

□

Table 3.1: A summary of the model parameters and corresponding design space size of recent language models.

Model	# of layers	# of decomposable Tensors	Decomposition Design Space Scale
BERT-Base	12	6	$O(2^{18})$
BERT-Large	24	6	$O(2^{30})$
Llama 2 - 7B	32	5	$O(2^{37})$
Llama 2 - 70B	80	5	$O(2^{85})$

Based on Theorem 3.1, we can estimate the size of the decomposition space as $O(2^{N_{Layers}(m)+N_{Tensors}(m)})$, even if we apply uniform decomposition as assumed in this subsection. Using the complexity we can analyze the design space of recent language models in Table 3.1. With the data, we observe the decomposition design space size for large language models such as Llama2 [1] is intractably huge. Therefore, we perform design space characterization to identify ineffective decomposition configurations and prune the space.

3.2 Decomposition Space Characterization

To explore decomposition design space pruning opportunities, we investigate the impact of each decomposition axis: (1) pruned rank, (2) tensors, and (3) layers.

3.2.1 Characterization Methodology

We use LLama2-7B [1] for the characterization studies. We run six benchmarks listed in Table 4.1 and track how the accuracy changes when we change each axis of the design space individually. We run the workloads on a system with AMC EPYC 7763 processor, 1TB of main memory, and four NVIDIA A100 GPUs with 80GB HBM2e memory for each. To measure the latency of the multi-GPU system, we measure GPU runtime utilizing *torch.cuda.event* APIs. We also utilize NVIDIA's System Management Interface (*nvidia-smi*) to measure the power consumption and the memory usage. We calculate the area under the power-time graph using *nvidia-smi*-reported average power information to estimate the GPU energy consumption.

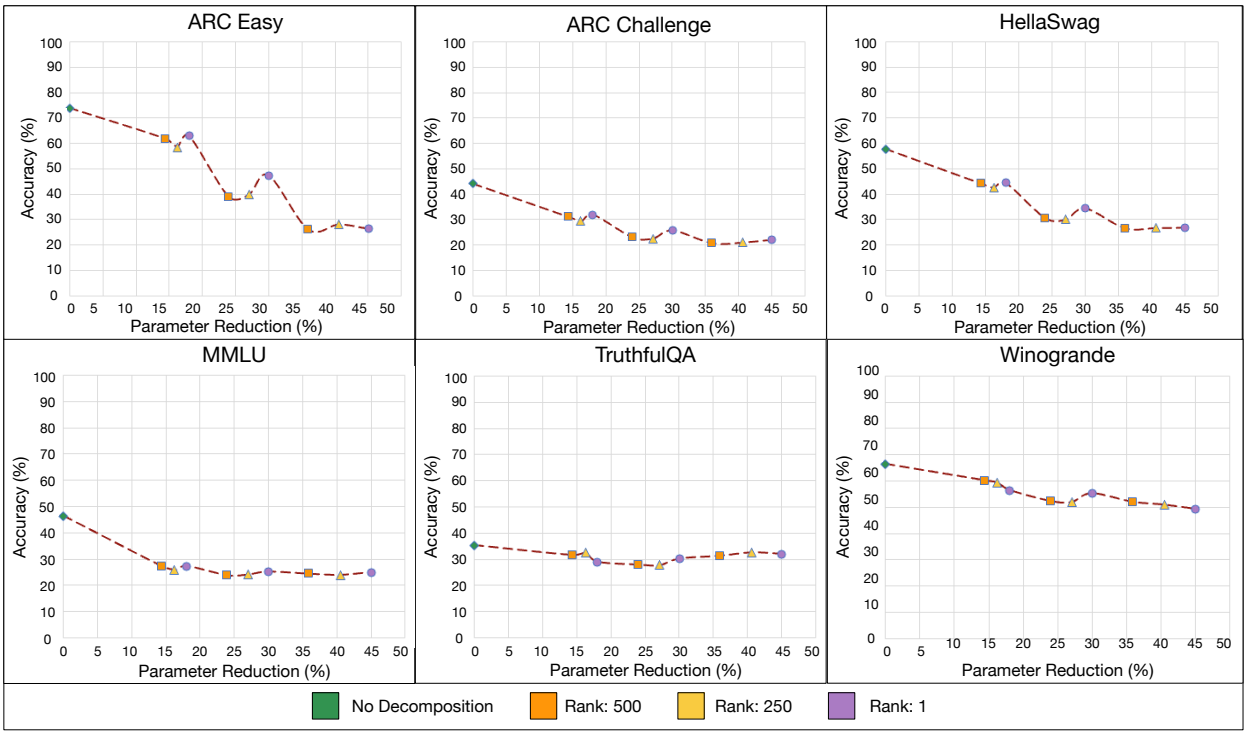


Figure 3.2: Impact of Rank on Accuracy. We prune ranks from the original (4096) to 500, 250, and 1. By pruned rank (PR), we refer to the remaining rank after rank pruning. The accuracy with no decomposition is based on the reported accuracy in the original Llama2 publication [1].

3.2.2 Characterizing the Impact of Pruned Rank Choices

We decompose all the tensors illustrated in Figure 3.1 (b) in each layer varying the pruned ranks to observe the impact of pruned rank choices on accuracy. We select different combinations of layers as shown in Table 4.2 to observe the trend on different parameter reduction rates. We choose three values of pruned ranks for our characterization: 1, 250, and 500. We do not increase the rank above 500 to ensure a meaningful model size reduction rate.

Observation. As small differences across orange, yellow, and purple data points in Figure 3.2 show, we observe that the rank of decomposition has minimal effect on the accuracy score, compared to that of the parameter reduction. We observe similar trends in all of the six benchmarks, which show the average variation in accuracy of 1.5%. With that observation, we select rank-1 decomposition for the main case study because rank-1 decomposition provides the most model size (parameter) reduction.

(a) BERT Layer Architecture **(b) Llama 2 Layer Architecture**

Figure 3.3: The layer architectures of BERT and Llama 2. We highlight decomposable weight tensors using yellow boxes.

3.2.3 Characterizing the Impact of Decomposed Tensor Choices

Figure 3.3 shows six and seven weight tensors in BERT and Llama 2 layers, respectively. In BERT, there are six weight tensors that are used to calculate the output of each layer, the Query (W_Q), Key (W_K), Value (W_V), self-attention output projection (W_{SO}), intermediate fully-connected layer (W_{Int}), and output fully connected layer (W_O). In Llama-2-7B, there are seven weight tensors used to calculate the output of each layer, the Query (W_Q), Key (W_K), Value (W_V), self-attention output projection (W_{SO}), Multi-Layer Perceptron (MLP) Gate projection (W_G), MLP Up projection (W_U) and MLP Down projection (W_D). We

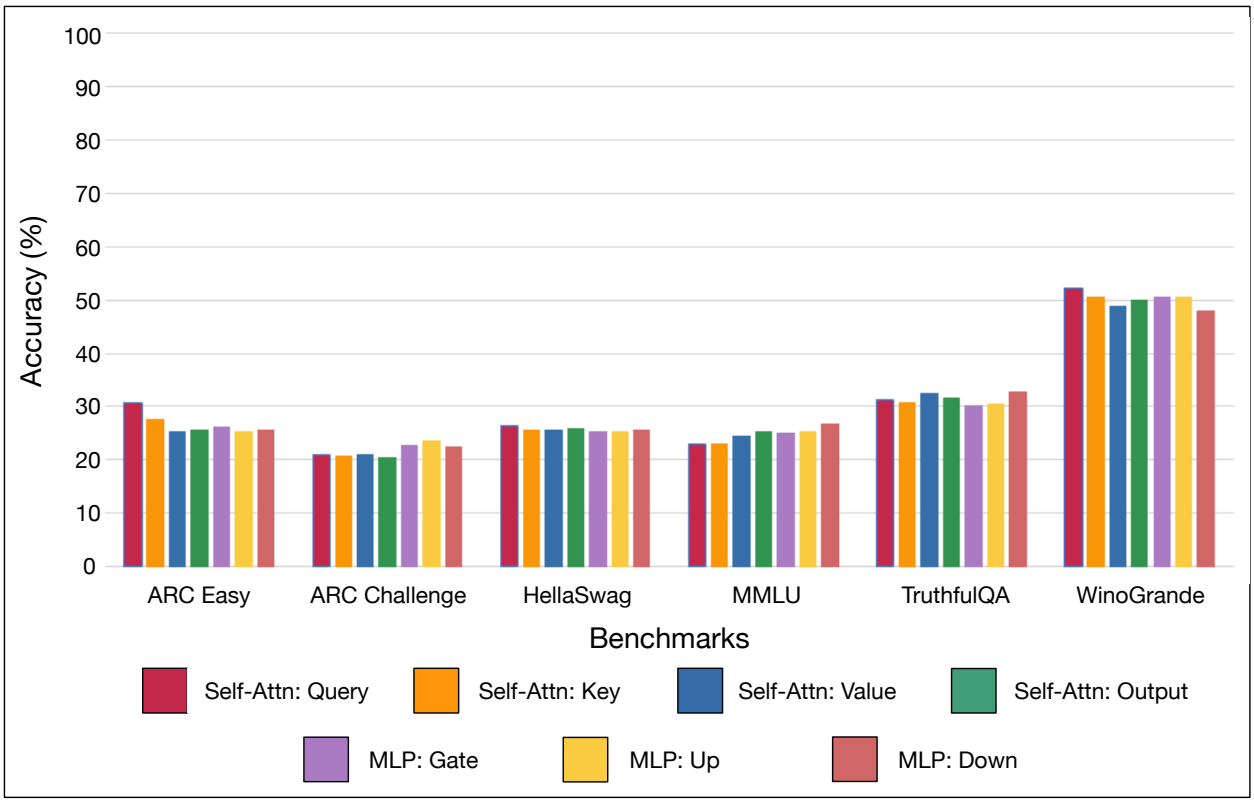


Figure 3.4: The impact of decomposed tensor choices on the accuracy on Llama2-7B.

decompose each tensor separately to analyze its impact on the overall model's accuracy. We also decompose these tensors in groups to investigate the accuracy impact of various combinations. We present the results in figures 3.4 and 3.5, and observe the following:

Observation 1) All the tensors in self-attention and MLP modules are equally sensitive to decomposition when compared within their same group. The sensitivity varies between different groups. We decompose the tensors using the pruned rank of 1 in self-attention and MLP modules of Llama-2-7B individually in one or all the decoder layers. From Figure 3.4 we observe no specific trend in the sensitivity among the decomposed tensors for Llama2-7B, which shows a weak sensitivity on the choice of decomposed tensors within the same overall decomposition rate. However, for BERT, we observe that the weight tensor of the intermediate fully-connected layer (W_{Int}) is the most sensitive under decomposition.

Observation 2) For same parameter reduction, decomposing all the tensors in a decoder layer is better than decomposing tensors in multiple layers. When

we decompose the Query tensor in all the 32 decoder layers of Llama-2-7B, we achieve a parameter reduction of 8% compared to the full model. However, that approach led to more than 50%p of accuracy loss while it only provided 8% parameter reduction. In contrast, if we decompose all the tensors and reduce the number of decomposed layers to keep the overall parameter reduction ratio identical, we observe much smaller accuracy drop (3%p). The results are shown in Figure 3.5 in two cases (a) when we target 8% parameter reduction and (2) 21% parameter reduction. We observe similar trends across those two cases, and for BERT as well. The results indicate that we should decompose more tensors rather than decomposing more layers with the same parameter reduction rate.

3.2.4 Characterizing the Impact of Decomposed Layer Choices

We decomposed each encoder layer in BERT and each decoder layer in Llama-2-7B and found out that the first few and the last few layers are more sensitive to decomposition compared to the layers in the middle. In the case of Llama-2-7B, for example, the first two layers and the last layer are more sensitive to decomposition, as shown in Figure 3.6.

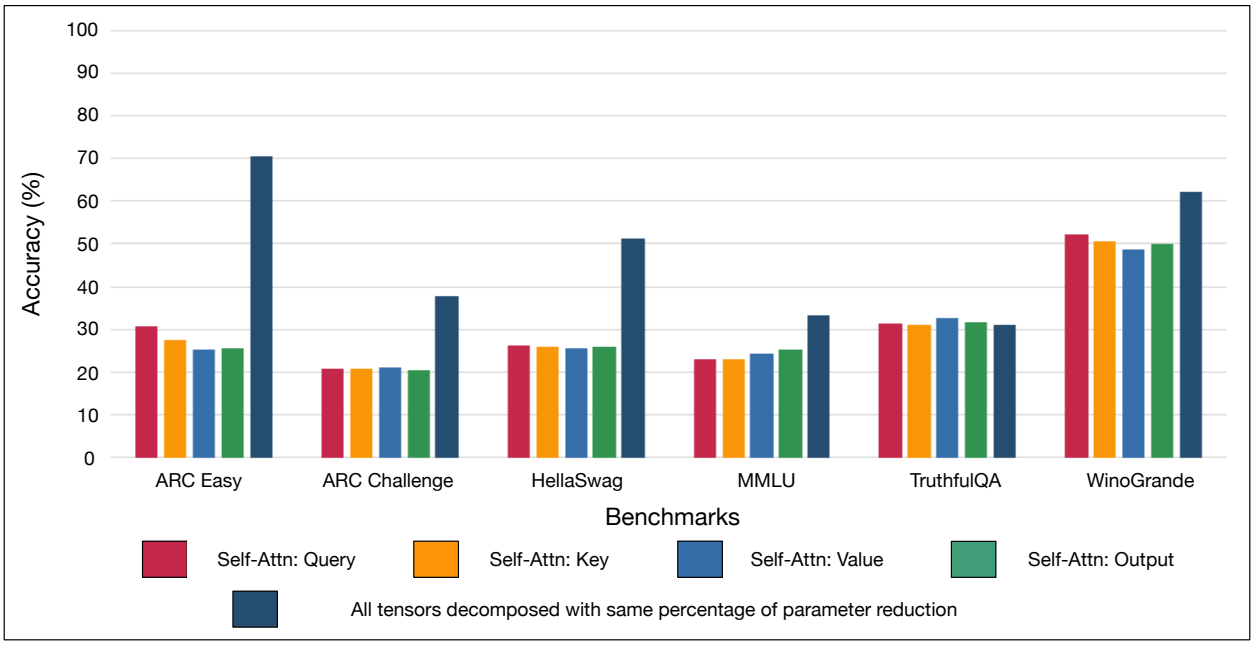
We also analyzed the effect of decomposing encoder/decoder layers that are closer to each other versus layers that are far apart. We observed that the impact on accuracy is less when the distance between decomposed layers is more. In other words, it is better to perform decomposition in layers that are far apart compared to layers that are closer to each other. From Figure 3.7, for example, we note that the accuracy score is much better if decomposition is performed in every sixth layer as compared to in the consecutive layers for all the benchmarks except TruthfulQA.

3.3 Insights from the Characterization

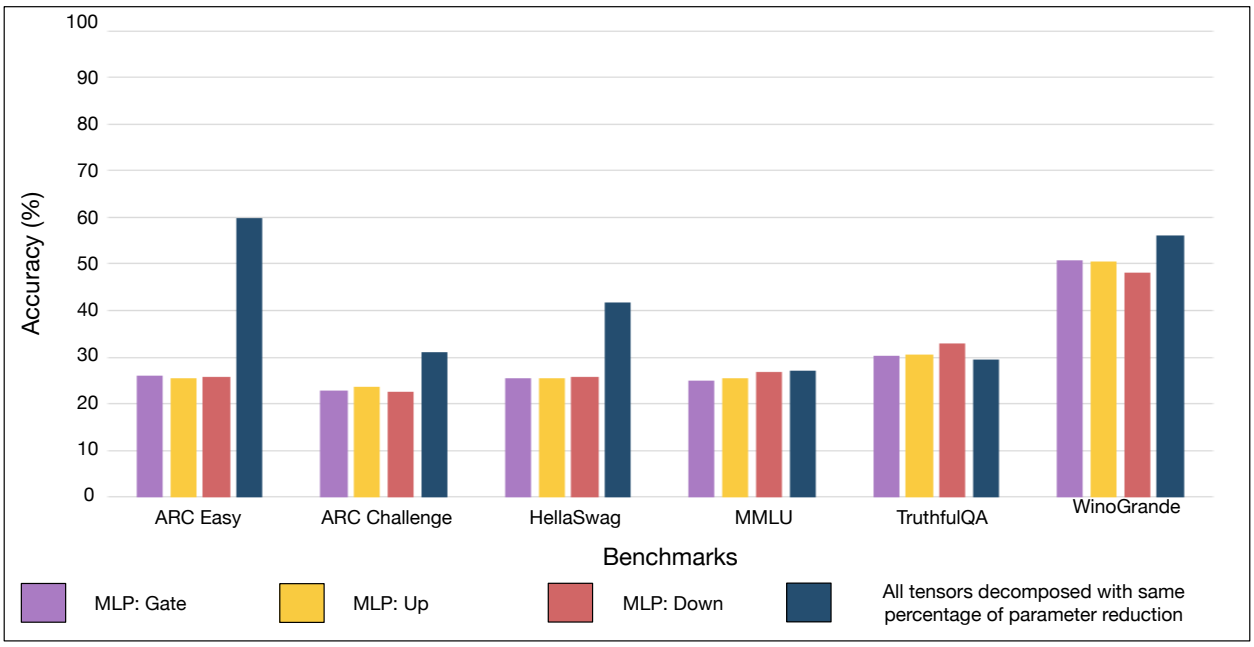
Based on the analysis presented in this section, we can draw the following conclusions:

- Rank-1 decomposition seems to be producing similar results compared to higher-rank decomposition. Therefore, it is better to use rank-1 decomposition as it results in higher parameter reduction with the same loss of accuracy as other higher-rank decomposition.
- Decomposing all the tensors inside an encoder/decoder layer or inside a single self-attention/MLP module is better compared to decomposing the same number of tensors in different layers or modules.
- It is best to avoid decomposition in the first few layers and the last few layers as these layers are more sensitive to decomposition.
- It is best to decompose layers uniformly spread apart, as far as possible, as compared to consecutive layers or layers closer to each other.

Based on these observations, we decomposed the Llama-2-7B model with varying decomposition percentages to observe the model's performance and the hardware gains because of the model compression. We present our case study in the next section.



(a) Case 1: Target parameter reduction rate is 8%



(b) Case 2: Target parameter reduction rate is 21%

Figure 3.5: To understand the impact of decomposed tensor choices, we compare two different ways for achieving similar parameter reduction rates: (1) decompose a specific tensor in many layers (2) decompose all tensors and select less number of layers to be decomposed. The right-most black bar corresponds to (2), and all the other bars correspond to (1). We compare those two approaches using two different parameter reduction targets: (a) 8% and (b) 21%.

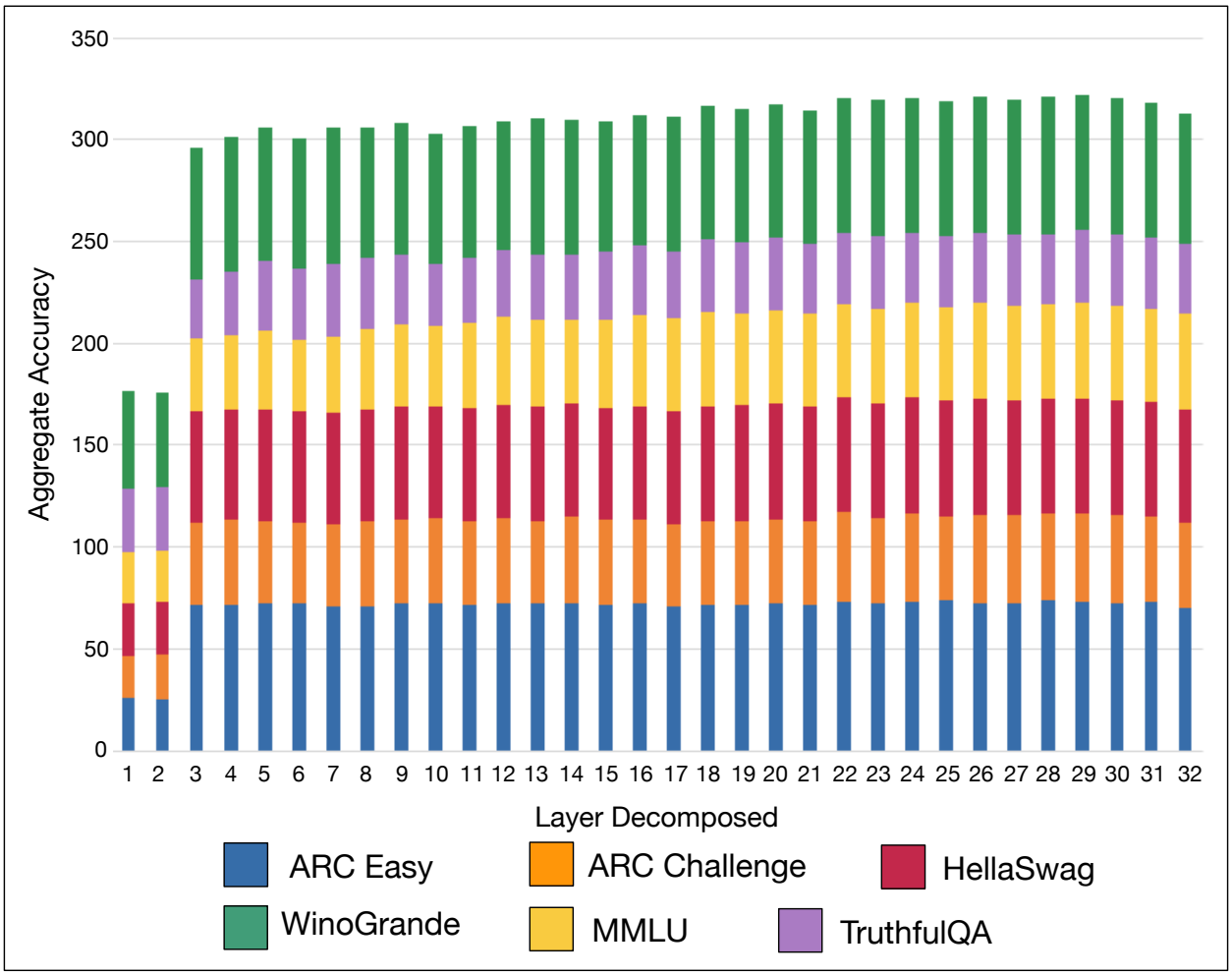


Figure 3.6: The aggregate accuracy across six benchmarks when we decompose different layer of Llama2-7B. We select one layer to be decompose and plot the correlation between the location of the selected layer and accuracy.

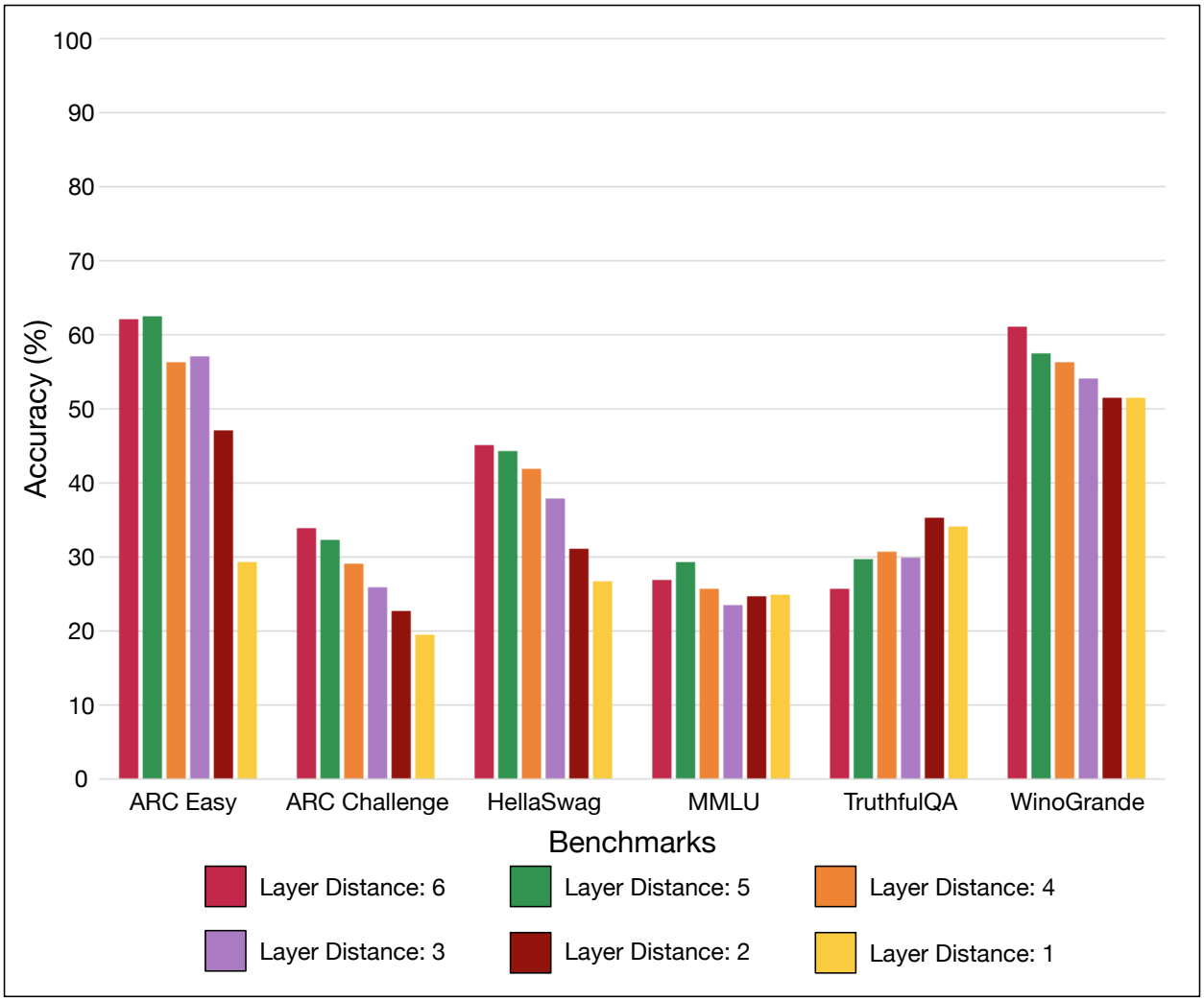


Figure 3.7: The impact of the distance between decomposed layers on model accuracy.

Chapter 4

Case Study

4.1 Experimental Setup

We use the same methodology and hardware platform as the initial characterization study, which is discussed in Subsection 3.2.1. Beyond the initial characterization, we perform a more thorough exploration of the accuracy-efficiency trade-off in this section with the reduced design space pruned by insights from characterization in Section 3.2.

Table 4.1: Six benchmarks used for our characterization and case studies.

Benchmark	**Task**	**# of Samples**
ARC Easy	Commonsense Reasoning (Q&A) - Easy	5.2K
ARC Challenge	Commonsense Reasoning(Q&A) - Challenging	2.59K
HellaSwag	Commonsense Reasoning(Sentence Completion) - Challenging	10K
MMLU	Multitask Language Understanding	15.9K
TruthfulQA	Truthfulness	1634
WinoGrande	Commonsense Reasoning (Q&A) - Moderate	44K
GSM8K	Mathematical Reasoning	8.5K

4.1.1 Benchmarks and Evaluation Methodology

To measure the pure inference time, energy, and memory usage on GPUs, we run each benchmark multiple times for more than two minutes. This approach ensures that we measure the statistics at the GPUs' steady state, which is aligned with the cloud that services LLM inferences. Following the common practice that exploits throughput-optimized GPUs, we use the maximum batch size for each GPU for each inference and utilize all of four NVIDIA A100 GPUs in parallel. We use EluetherAI's [30] `lm-evaluation-harness` framework to get the accuracy of Llama-2-7B on the benchmarks.

We evaluate our low-rank decomposition scheme on all the standard benchmarks used by the HuggingFace Open LLM Leaderboard, which includes ARC (Easy and Challenge), HellaSwag, MMLU, TruthfulQA, WinoGrande, and GSM8K. These benchmarks span a variety of domains where LLMs are often used and provide a good basis for testing the performance of our low-rank decomposition scheme on LLMs comprehensively. We discuss how we utilize each benchmark for four LLM tasks as follows:

- **Commonsense Reasoning**: We report the zero-shot accuracy of ARC Easy and Challenge, HellaSwag, and WinoGrande. While the ARC and WinoGrande datasets use question-answering format to test common sense reasoning, HellaSwag tests the sentence completion ability of a model.
- **Multitask Accuracy:** We report 0-shot accuracy of MMLU dataset which tests a model's accuracy on multiple tasks like world knowledge and problem solving.
- **Truthfulness (safety benchmark):** We report the accuracy on TruthfulQA benchmark which tests if a model can generate reliable outputs which agree with factuality and common sense.
- **Mathematical Reasoning:** We report 8-shot exact match scores of GSM8K (Grade

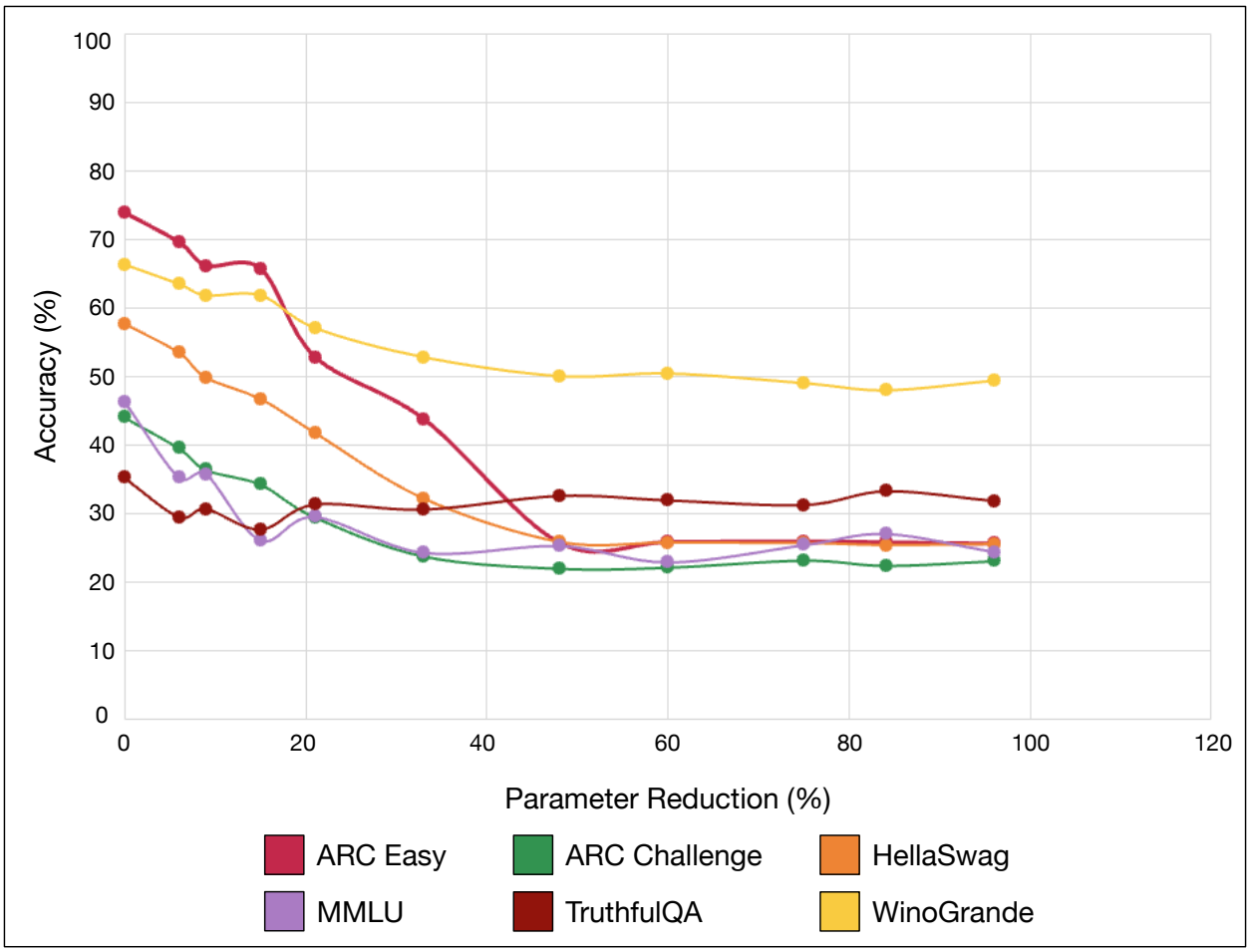


Figure 4.1: The correlation between the model size (parameter) reduction ratio by low-rank decomposition and resulting model accuracy.

School Math 8K) benchmark which tests a model's performance on linguistically diverse grade school math word problems that require multi-step reasoning.

4.2 Decomposition Methodology

As baselines without decomposition, we collect publicly available models on HuggingFace [31] and profile them by logging the inference accuracy, time, power consumption, energy consumption, and memory usage. We decompose those models using the layer configurations listed in Table 4.2. Note that the layer configuration in Table 4.2 is driven by the learning in Subsection 3.2.4, which motivates us to avoid decomposing the first two layers and nearby layers to prevent severe accuracy drop. We use the pruned rank of 1 and decompose all

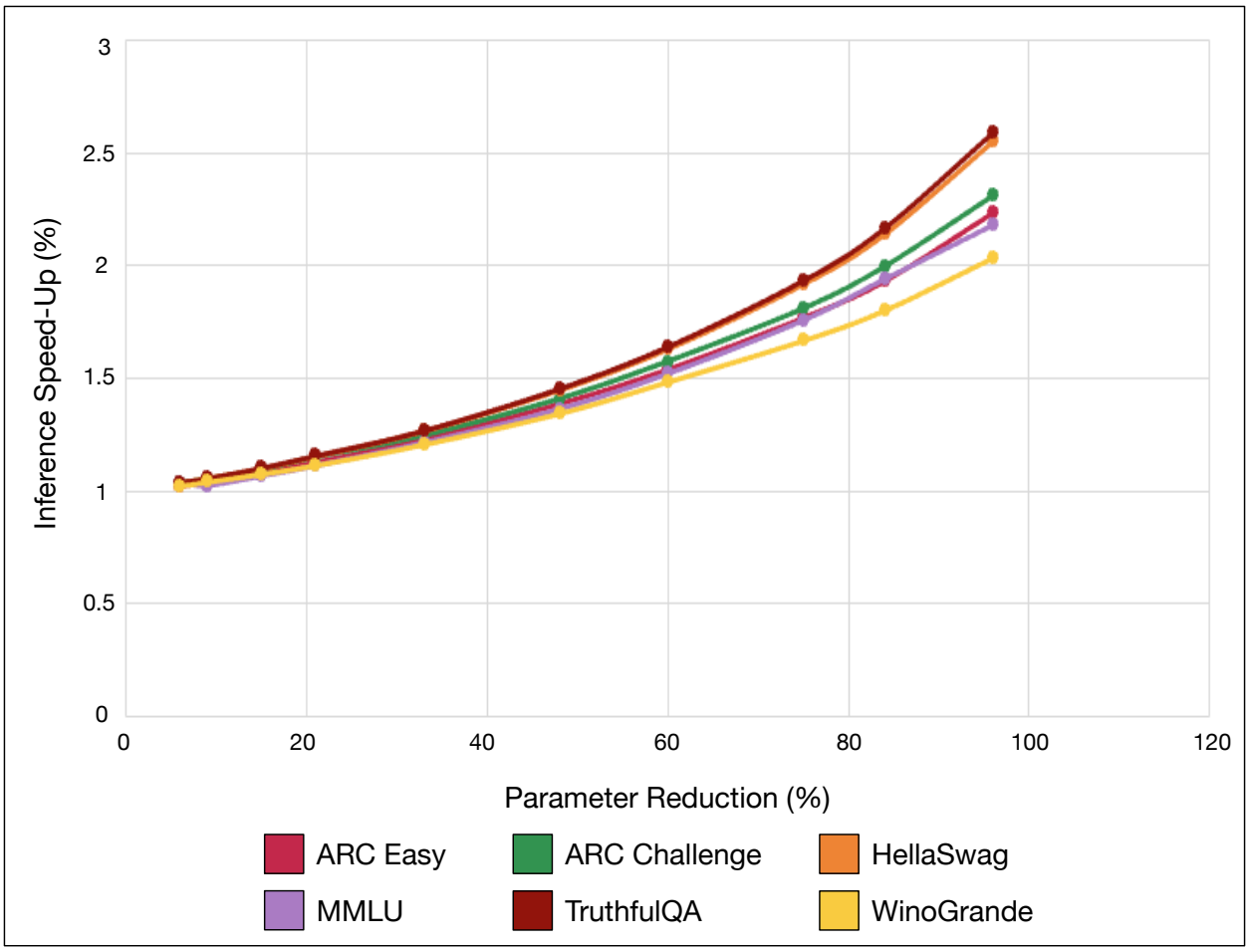


Figure 4.2: The correlation between the model size (parameter) reduction ratio by low-rank decomposition and speedup.

the tensors within selected layers based on the learning from our characterization study in Section 3.2.

4.3 Results and Discussion

4.3.1 Accuracy-efficiency trade-off

Accuracy. Figure 4.1 shows the accuracy impact of tensor decomposition at various levels of model compression. As the compression increases–that is, as the parameters are reduced–the accuracy starts to drop while inference time, energy consumption, and memory usage decrease as well. If we classify common-sense reasoning benchmarks based on the inference accuracy, we

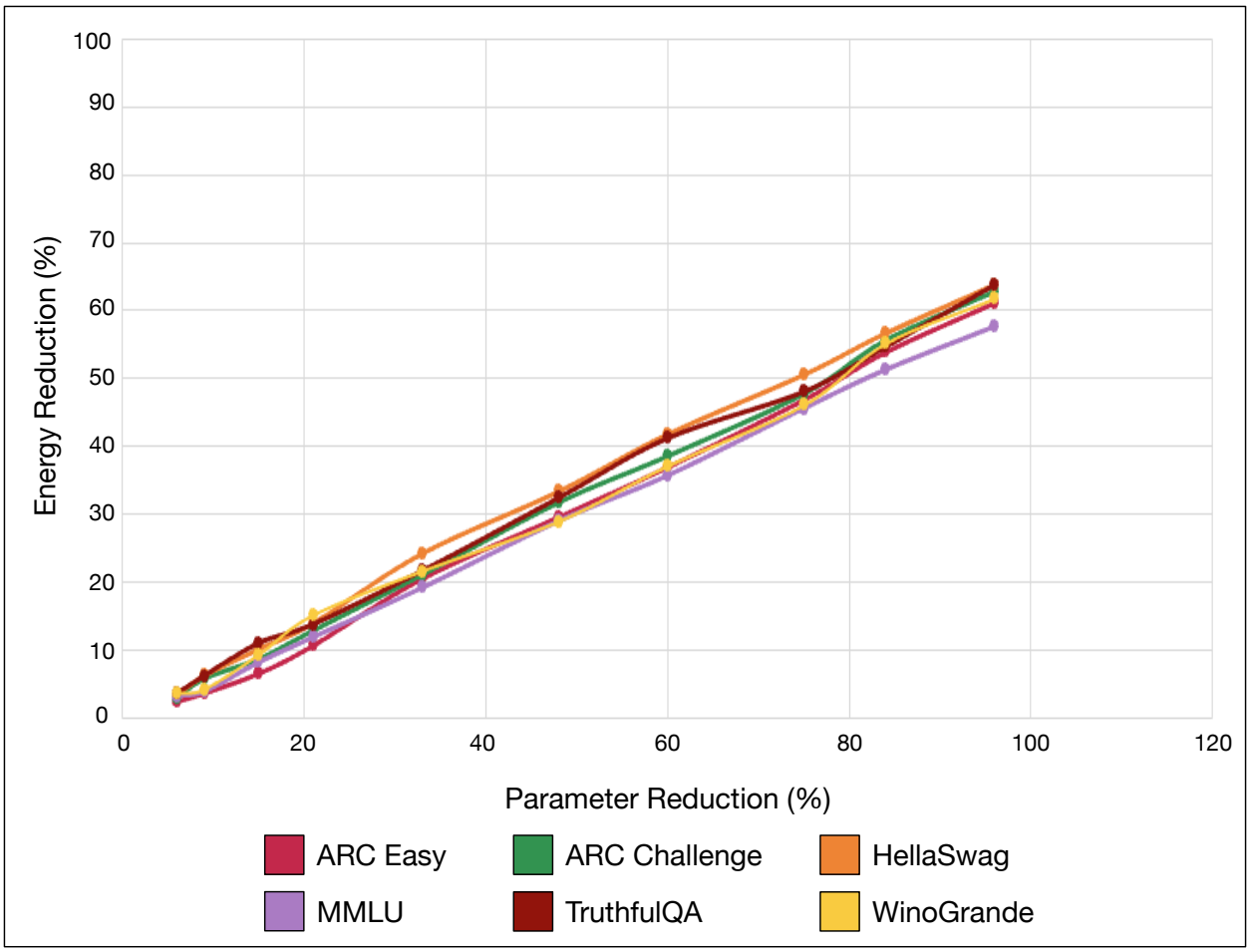


Figure 4.3: The correlation between the model size (parameter) reduction ratio by low-rank decomposition and energy reduction.

can consider benchmarks like ARC Easy and WinoGrande as simpler benchmarks compared to ARC Challenge, HellaSwag, MMLU, and GSM8K, as they have a higher inference accuracy. For such relatively easy benchmarks, the decline in accuracy is smaller as the decomposition percentage increases compared to more challenging benchmarks.

For ARC Easy, parameter reduction via decomposition resulted in considerable accuracy drop, which led to 7.4%p accuracy drop per 1% reduction in parameter. In the case of WinoGrande, we observe the least degradation of accuracy. In the case of TruthfulQA, the results are interesting because the accuracy drops from 35 to 29 with a 6% reduction in parameters. However, as we further reduce the number of parameters by 96%, we observe the accuracy is improves to 32%. Therefore, TruthfulQA shows a reverse trend as compared

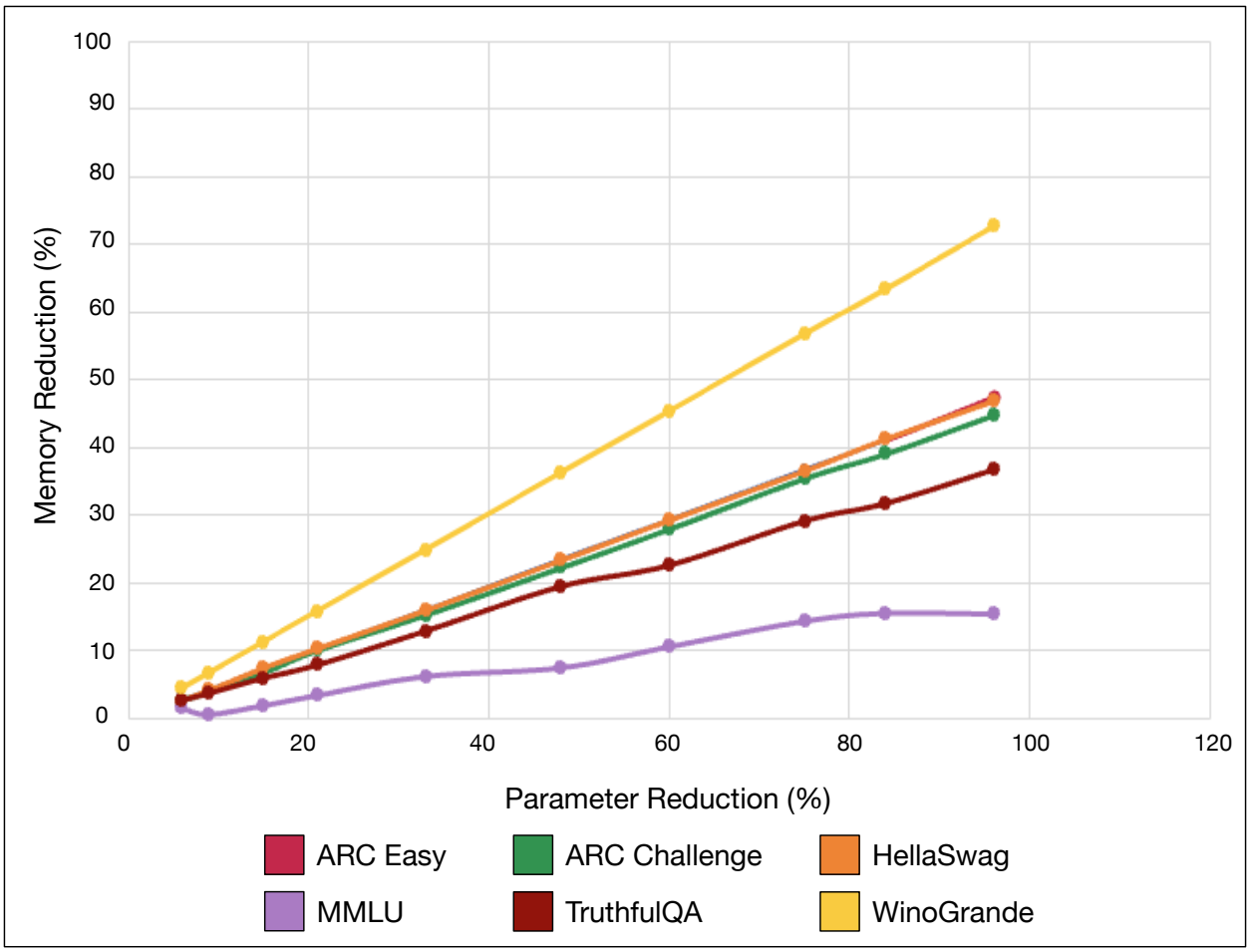


Figure 4.4: The correlation between the model size (parameter) reduction ratio by low-rank decomposition and memory footprint.

to other benchmarks. In case of difficult benchmarks like ARC Challenge, HellaSwag, and MMLU, the accuracy drops between 4 to 11 %p for up to a 9% reduction in parameters, and the accuracy drop increases for a higher reduction in parameters, as shown in Figure 4.1.

Efficiency. As we reduce the size of the model by low-rank decomposition, the memory usage decreases. This results in faster inference time and consequently less energy consumption. In terms of inference time, we observe a steady reduction in inference time as model size reduces. For every 1% reduction in model size, we see an average of 0.5% reduction in inference time. Since the LLMs utilize the GPU at 100% utilization, the power consumption of the GPU is always the maximum (300W in the case of NVIDIA A100 80GB). Therefore, savings in inference latency result in a proportional saving in energy consumption. We observe a similar

Table 4.2: Decomposed layer choices used in our case studies and corresponding parameter reduction rates.

Parameter Reduction (%)	**Decomposed Layers**
6%	3, 30
9%	3, 18, 32
15%	3, 9, 15, 21, 27
21%	5, 9, 13, 17, 21, 25, 29
33%	3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 32
48%	1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31
60%	2, 4, 6, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 23, 25, 27, 29, 31
75%	2, 4, 6, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30
84%	1, 3, 5, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32
96%	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32

ratio in energy savings where for every percentage reduction in parameters results in an approximate 0.5% reduction in energy consumption. Across all the benchmarks we evaluate, the relative standard deviation of inference time and energy consumption is 2.6% and 2.9%, respectively. The absolute numbers are shown in Figure 4.2 and Figure 4.3 for inference latency and energy consumption, respectively. A similar trend is observed in the total memory usage of the GPU, where a 1% reduction in model parameters results in approximately 0.4% reduction in total memory usage of the GPU. The exact memory usage reduction is for each benchmark is presented in Figure 4.4.

4.3.2 Insights from the Results

Based on the results obtained in our case study, the main insights of this work are as follows:

- We explored the accuracy-efficiency trade-off of low-rank decomposition in LLMs.
- We show that it is possible to compress LLMs by reducing the number of parameters

by up to 9% with minimal accuracy loss of 4%p to 10%p depending on the benchmark, without retraining or fine-tuning the model after decomposition.

- We also show that for every 1% reduction in model's parameters, there is a proportional decrease of 0.5% in inference latency and the energy consumption. The memory usage also decreases by 0.4% for the same amount of reduction in parameters.

- We observe that the loss of accuracy tapers-off when more than 48% of the parameters are reduced.Therefore, using techniques like knowledge-distillation and fine-tuning, it might be possible to recover the accuracy and further compress the model reducing its size significantly with an acceptable accuracy loss.

- We also note that a decomposed model with fewer parameters than the original model consumes less energy and has lower inference latency. The inference latency and the energy consumption scale linearly with the decomposed model size, and this holds true across all the benchmarks we examined.

Chapter 5

Related Works

Low-Rank Decomposition in Deep Learning. Although low-rank decomposition has been actively explored as a model compression technique, there are few studies about their implication on recent large language models. In [32] the authors decompose the BERT model using SVD and perform feature distillation to recover accuracy. However, the authors use the BERT model and GLUE benchmark which are not representative of bigger language models with diverse sets of tasks. Further, the paper does not present a decomposition methodology in case fine-tuning on training datasets is not possible. In [33], authors proposed a way to decompose the input embedding layers using Tensor Train decomposition. The authors in [34] present low-rank decomposition for compressing Convolutional Neural Networks (CNNs) using Canonical Polyadic (CP) decomposition. They also show that a combination of Tucker and CP decomposition might be better for convolutional layers of CNNs. In [35] however, the authors present a methodology to decompose both the fully-connected layers and the convolutional layers and recover accuracy by retaining the decomposed model using knowledge transfer from the original model. In GroupReduce [36] the authors present how language models with big vocabulary sizes which have more than 90% of the parameters in the embedding layer can be compressed using low-rank decomposition of the embedding layer. However, common models that do not use exceptionally large vocabulary still have a majority

of their parameters in the encoder/decoder layers, therefore we do not decompose embedding layers. In TIE [37] the authors developed a computation-efficient inference scheme for DNNs decomposed using Tensor Train decomposition.

Fine-tuning for LLMs. LoRA [38] proposed a low-rank adapter for fine-tuning LLMs. LoRA freezes all the pre-trained weights and trains small low-rank decomposed weights using extra training data prepared for fine-tuning. After completing the fine-tuning, the low-rank decomposed weights are contracted to generate a tensor with the same size as the original weight. As the contracted tensor (i.e., fine-tuned weight) has the same shape as the original weights, the contracted tensor can be added to the original weight to eliminate inference time overhead.

Performance Characterization of LLMs. Because the large language model emerged recently, the computational performance characterization on LLM inference is not well-explored. Megatron-LM analyzed the computational performance of LLM training targeting data centers [39]. However, the work focused on the training workload without model compression techniques like low-rank decomposition.

Chapter 6

Conclusion and Future Work

In this work, we explored the accuracy-efficiency trade-off of low-rank decomposition, which was not well understood previously. By formalizing the decomposition design space, we showed that the space is huge, which makes it intractable to navigate. To address the challenge, we perform characterization focusing on three axes (number of pruned ranks, the choice of decomposed layers, and decomposed tensors) identified based on our formulation and extract useful insights helpful for reducing the design space: we can prune the rank down to 1 with minimal accuracy impact, we should not decompose early layers, and we should not decompose adjacent layers. Such insights can be adopted to future algorithm-level research for developing high-accuracy and -efficiency low-rank decomposition methods for LLMs. In particular, recovering accuracy using fine-tuning after low-rank decomposition is a promising direction. Our early investigation shows that we can recover the accuracy of 15% compressed model to that of a 9% model within a single epoch of fine-tuning, which motivates future studies in this domain.

Our characterization code base will be open-sourced to facilitate such future studies. In particular, unlike many other works, we include energy profiling, which will enable a more thorough consideration of the accuracy-efficiency trade-off in future LLM algorithm research.

Bibliography

[1] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[2] OpenAI, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[3] H. Huang, F. D. L. Torre, C. M. Fang, A. Banburski-Fahey, J. Amores, and J. Lanier, "Real-time animation generation and control on rigged models via large language models," in *NeurIPS 2023*, December 2023. Spotlight Paper.

[4] A. Ni, S. Iyer, D. Radev, V. Stoyanov, W.-t. Yih, S. Wang, and X. V. Lin, "Lever: Learning to verify language-to-code generation with execution," in *International Conference on Machine Learning*, pp. 26106–26128, PMLR, 2023.

[5] G. Cloud, "Agent assist." `https://cloud.google.com/agent-assist?hl=en`, 2024.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[8] E. Frantar and D. Alistarh, "Sparsegpt: Massive language models can be accurately pruned in one-shot," in *International Conference on Machine Learning*, pp. 10323–10337, PMLR, 2023.

[9] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-bert: Integer-only bert quantization," in *International conference on machine learning*, pp. 5506–5518, PMLR, 2021.

[10] Z. Yao, R. Yazdani Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He, "Zeroquant: Efficient and affordable post-training quantization for large-scale transformers," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27168–27183, 2022.

[11] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale," *Advances in Neural Information Processing Systems*, vol. 35, pp. 30318–30332, 2022.

[12] A. Zare, A. Ozdemir, M. A. Iwen, and S. Aviyente, "Extension of pca to higher order data structures: An introduction to tensors, tensor decompositions, and tensor pca," *Proceedings of the IEEE*, vol. 106, no. 8, pp. 1341–1358, 2018.

[13] S. Kong and C. Fowlkes, "Low-rank bilinear pooling for fine-grained classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 365–374, 2017.

[14] X. Yu, T. Liu, X. Wang, and D. Tao, "On compressing deep models by low rank and sparse decomposition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7370–7379, 2017.

[15] J. Gusak, M. Kholiavchenko, E. Ponomarev, L. Markeeva, P. Blagoveschensky, A. Cichocki, and I. Oseledets, "Automated multi-stage compression of neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

[16] H. Wang, S. Agarwal, Y. Tanaka, E. Xing, D. Papailiopoulos, *et al.*, "Cuttlefish: Low-rank model training without all the tuning," *Proceedings of Machine Learning and Systems*, vol. 5, 2023.

[17] M. Xu, Y. L. Xu, and D. P. Mandic, "Tensorgpt: Efficient compression of the embedding layer in llms based on the tensor-train decomposition," *arXiv preprint arXiv:2307.00526*, 2023.

[18] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the ai2 reasoning challenge," *arXiv preprint arXiv:1803.05457*, 2018.

[19] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[20] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[21] S. Lin, J. Hilton, and O. Evans, "TruthfulQA: Measuring how models mimic human falsehoods," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (S. Muresan, P. Nakov, and A. Villavicencio, eds.), (Dublin, Ireland), pp. 3214–3252, Association for Computational Linguistics, May 2022.

[22] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "Winogrande: an adversarial winograd schema challenge at scale," *Commun. ACM*, vol. 64, p. 99–106, aug 2021.

[23] Github, "Github copilot." `https://github.com/features/copilot`, 2023.

[24] Microsoft, "Announcing microsoft copilot, your everyday ai companion." https://blogs.microsoft.com/blog/2023/09/21/announcing-microsoft-copilot-your-everyday-ai-companion/, 2023.

[25] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of naacL-HLT*, vol. 1, p. 2, 2019.

[26] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika 31, 279–311 (1966)*, 1996.

[27] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.

[28] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ questions for machine comprehension of text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (J. Su, K. Duh, and X. Carreras, eds.), (Austin, Texas), pp. 2383–2392, Association for Computational Linguistics, Nov. 2016.

[29] S. Kim, C. Hooper, T. Wattanawong, M. Kang, R. Yan, H. Genc, G. Dinh, Q. Huang, K. Keutzer, M. W. Mahoney, *et al.*, "Full stack optimization of transformer inference: a survey," *arXiv preprint arXiv:2302.14017*, 2023.

[30] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou, "A framework for few-shot language model evaluation," 12 2023.

[31] H. Face, "Models - hugging face." https://huggingface.co/models, 2024.

[32] M. Ben Noach and Y. Goldberg, "Compressing pre-trained language models by matrix decomposition," in *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing* (K.-F. Wong, K. Knight, and H. Wu, eds.), (Suzhou, China), pp. 884–889, Association for Computational Linguistics, Dec. 2020.

[33] O. Hrinchuk, V. Khrulkov, L. Mirvakhabova, E. Orlova, and I. Oseledets, "Tensorized embedding layers," in *Findings of the Association for Computational Linguistics: EMNLP 2020* (T. Cohn, Y. He, and Y. Liu, eds.), (Online), pp. 4847–4860, Association for Computational Linguistics, Nov. 2020.

[34] A.-H. Phan, K. Sobolev, K. Sozykin, D. Ermilov, J. Gusak, P. Tichavskỳ, V. Glukhov, I. Oseledets, and A. Cichocki, "Stable low-rank tensor decomposition for compression of convolutional neural network," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX 16*, pp. 522–539, Springer, 2020.

[35] S. Lin, R. Ji, C. Chen, D. Tao, and J. Luo, "Holistic cnn compression via low-rank decomposition with knowledge transfer," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 12, pp. 2889–2905, 2018.

[36] P. Chen, S. Si, Y. Li, C. Chelba, and C.-J. Hsieh, "Groupreduce: Block-wise low-rank approximation for neural language model shrinking," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[37] C. Deng, F. Sun, X. Qian, J. Lin, Z. Wang, and B. Yuan, "Tie: Energy-efficient tensor train-based inference engine for deep neural network," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pp. 264–277, 2019.

[38] E. J. Hu, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, *et al.*, "Lora: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022.

[39] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, *et al.*, "Efficient large-scale language model training on gpu clusters using megatron-lm," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2021.