Partitioning Transformer Networks with Low Intercommunication

Matthew Thill* and Dariush Divsalar*

ABSTRACT. — We present a methodology for partitioning large transformer networks into disconnected components with limited intercommunication, allowing them to be hosted by a collection of machines in a space-based environment. We describe how this method would reduce the memory and power demands of large AI models, outlining a pathway toward their future use in space exploration.

I. Introduction

New advances in large neural network architectures have sparked a renaissance in artificial intelligence, leading to giant leaps in areas such as data analysis, scientific understanding, and robotics. These networks promise to revolutionize space exploration, but they require massive amounts of computational resources and power. As an example, the largest open-source Mixture-of-Experts (MoE) model is DeepSeek-AI's DeepSeek-V3 architecture [1], which contains about 671 billion parameters. The most current MoE models, such as OpenAI's GPT-5 series and xAI's Grok-5, are suspected to contain several trillion parameters. These models will typically have tens to hundreds of billions of parameters activated for each token processed. While we expect any AI model utilized in space would be trained on Earth, the cost of supporting a pre-trained model is still very high by space-based technological standards. A single instance of a model could require 8 to 16+ high-end NVIDIA H200 or Blackwell Graphics Processing Units (GPUs) to achieve the hundreds to thousands of gigabytes of Video Random Access Memory (VRAM) needed to run the model. These require high-speed interconnects to transmit data between the GPUs, and come with significant size, weight, and power demands. They are often hosted in large clusters of 72-GPU racks.

---

*Advanced Communications and Networking Section.

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

In the near future, a space environment will have far more limited resources. Rather than interconnected GPU racks, there will more likely be smaller GPU processors distributed over a network of machines such as spacecraft, rovers, landers, and CubeSats, as illustrated in Figure 1. They will have wireless radio and/or optical connections. In order for them to host such a large network, we need to find ways of partitioning it amongst them and coordinating the machines to implement it, We must also reduce the memory and power requirements, as well as the amount of data that must be transmitted between them, without harming the network performance. The inputs to the network will likely be based on instrument measurements from each machine, such as spectrometer readings and camera images. The network may be used to gain scientific understanding of the data, or to use the collective measurements to coordinate the movement of robotic agents, for instance. The fact that inputs to the network will come from separate machines should constrain how we partition it.

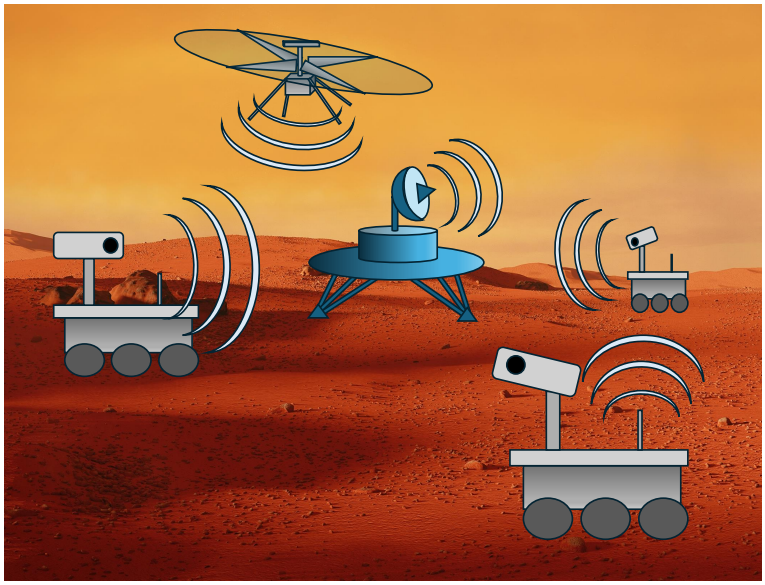


Figure 1. Intercommunicating machines that could simultaneously implement a large neural network based on their collective sensor measurements.

State-of-the-art AI models are based on the transformer architecture [2], shown in Figure 2, and its variations (e.g., the vision transformer [3]). These models function by tokenizing their inputs and embedding the tokens into a vector space based on both their token value and their position in the token sequence. Central to their success are their attention units, which map each of these embeddings to "query," "key," and "value" vectors, and compute the dot products of each query vector with the keys of all tokens. These are then used to weight a sum of all the value vectors. Some of these models can operate on context windows containing millions of tokens, and employ "multiheaded" attention mechanisms whereby multiple sets of query, key, and value vectors are produced for each token. For a network partitioned over M machines, this risks that a large amount of required intercommunication will be demanded between them.

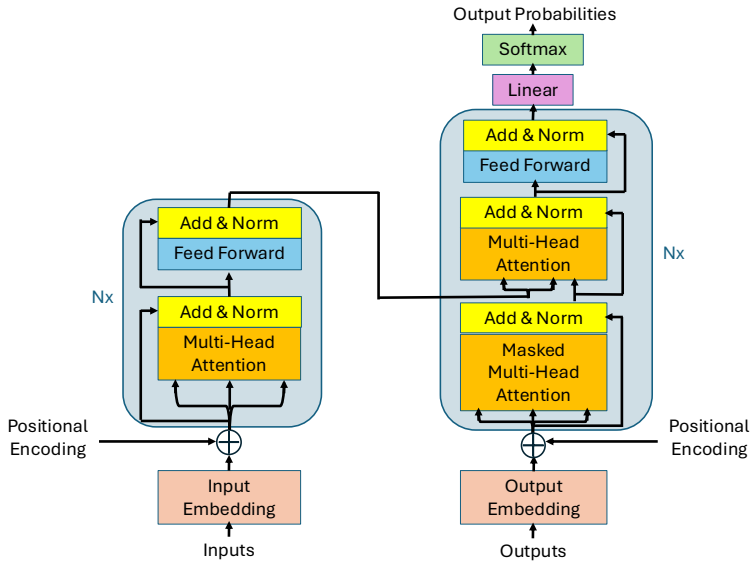


Figure 2. Diagram of a transformer network. Blocks in blue are repeated N_x times.

Another key feature of transformer models are the feed-forward components that follow the attention units. These contain a series of fully connected dense layers that contain a large portion of the parameters of the transformer model, and require a large number of floating-point operations (FLOPs). The size of these layers is driven by the model dimension, which has historically grown larger with more current models.

We will treat our problem in two stages: In Section II, we will discuss a method of partitioning the dense layers of the feed-forward sections of the transformer. We note that our methodology will be applicable to general networks with a series of dense layers, such as those arising in deep learning scenarios. In Section III, we will look at how to partition the attention mechanisms, focusing on preserving the query-key inner products which are central to their performance. Our methodology will be to identify features of the transformer network which merit low-rank approximations. This will allow us to reduce the number of parameters we need to store in memory, decrease the number of FLOPs needed to implement dense layers, and have low intercommunication between the machines involved in the partition while maintaining little distortion in the network performance. We will attempt to quantify the tradeoff between the amount of data communicated and the approximation error incurred as a result.

It is worth noting that low-rank compression schemes have been applied to transformers and other networks before [4, 5, 6, 7, 8, 9]. There has also been work to compress and reduce the computation involved in the attention query-key dot products [10]. But ours centers on the goal of equally partitioning the network parameters on a disjoint set of machines.

II. Partitioning Dense Layers

A dense layer of a neural network takes a vector of inputs $\mathbf{x}^{(0)} \in \mathbb{R}^{d_0}$ and returns

$$\mathbf{x}^{(1)} := \sigma\left(\mathbf{W}\mathbf{x}^{(0)} + \mathbf{b}\right) \in \mathbb{R}^{d_1}, \tag{1}$$

where $\mathbf{W} \in \mathbb{R}^{d_1 \times d_0}$ is a matrix of weights, $\mathbf{b} \in \mathbb{R}^{d_1}$ is a vector of biases, and $\sigma(x)$ is an activation function, which is applied elementwise to $\mathbf{W}\mathbf{x}^{(0)} + \mathbf{b}$. Common choices for $\sigma(x)$ are nonlinear functions such as the sigmoid function $[1 + \exp(-x)]^{-1}$, the hyperbolic tangent function $\tanh(x)$, or the rectified linear unit (ReLU) function $\max(0, x)$. In deep learning models, it is common for many of these types of layers to be concatenated:

$$\mathbf{x}^{(i+1)} := \sigma_i\left(\mathbf{W}^{(i)}\mathbf{x}^{(i)} + \mathbf{b}^{(i)}\right),\ i = 0, ..., n-1, \tag{2}$$

where $\mathbf{W}^{(i)} \in \mathbb{R}^{d_{i+1} \times d_i}$ and $\mathbf{b}^{(i)} \in \mathbb{R}^{d_{i+1}}$. As such, the weight matrices $\mathbf{W}^{(i)}$ can account for a large portion of the model's parameters. Advanced models might have thousands of inputs, and try to capture many more features of the data (represented by the intermediate output vectors $\mathbf{x}_i$). Typical transformer layers of large language models each have a feed-forward network with two dense layers ($n = 2$), commonly with $d_2 = d_0$ and $d_1 = 4 \cdot d_0$ or even $8 \cdot d_0$. One such configuration of a single feed-forward network of Google's PaLM model (with $d_0 = d_2 = 12,288$ and $d_1 = 98,304$) contains two weight matrices of roughly 1.2 billion parameters each, or around 4.8 GB at 32-bit floating-point precision. The full model contains 64 such layers, resulting in almost 155 billion parameters for the weight matrices of the dense layers alone—a whopping 618 GB.

A. Baseline Method: Partitioning the Weight Matrix

As we develop our methodology, it will be useful to have a baseline algorithm with which to compare certain performance metrics. We consider the case where we have M machines over which to distribute a dense layer's parameters and computation. Suppose each machine has access to a fraction $1/M$ of the inputs, and a chunk of the corresponding weight-matrix columns. We can rewrite Equation (1) in the form

$$\mathbf{x}^{(1)} = \sigma\left(\begin{bmatrix}\mathbf{W}_1 & \mathbf{W}_2 & \dots & \mathbf{W}_M\end{bmatrix} \cdot \begin{bmatrix}\mathbf{x}_1^{(0)} \\ \mathbf{x}_2^{(0)} \\ \vdots \\ \mathbf{x}_M^{(0)}\end{bmatrix} + \mathbf{b}^{(0)}\right) \tag{3}$$

$$= \sigma\left(\sum_{m'=1}^{M} \mathbf{W}_{m'}\mathbf{x}_{m'}^{(0)} + \mathbf{b}^{(0)}\right), \tag{4}$$

where $\mathbf{x}_{m'}^{(0)} \in \mathbb{R}^{d_0/M}$ is the portion of the inputs owned by machine m', for $m' = 1, ..., M$, and $\mathbf{W}_{m'} \in \mathbb{R}^{d_1 \times \frac{d_0}{M}}$ is the matching set of columns of $\mathbf{W}$.

With this partitioning, machine m' can naturally be responsible for the matrix-vector multiplication

$$\mathbf{w}_{m'} := \mathbf{W}_{m'}\mathbf{x}_{m'}^{(0)} \in \mathbb{R}^{d_1}. \tag{5}$$

However, it remains unclear how to complete the layer's computation short of all machines communicating their vector $\mathbf{w}_{m'}$ to a central processor which computes $\mathbf{x}^{(1)}$, and then relays the outputs back to the machines. This is a costly protocol, since it requires constant communication of all input and output parameters of the layer. Furthermore, we have not reduced the total amount of memory and computational resources (measured by the number of floating-point operations) needed to host the full weight matrix $\mathbf{W}^{(0)}$ and evaluate the product $\mathbf{W}^{(0)}\mathbf{x}^{(0)}$.

B. A New Partitioning Method

We now assume that each machine has access to a fraction $1/M$ of the inputs and is responsible for computing the same fraction of the outputs for this layer. As such, we will re-write Eq. (1) as

$$\begin{bmatrix} \mathbf{x}_1^{(1)} \\ \mathbf{x}_2^{(1)} \\ \vdots \\ \mathbf{x}_M^{(1)} \end{bmatrix} = \sigma\left(\begin{bmatrix} \mathbf{W}_{1,1} & \mathbf{W}_{1,2} & \dots & \mathbf{W}_{1,M} \\ \mathbf{W}_{2,1} & \mathbf{W}_{2,2} & \dots & \mathbf{W}_{2,M} \\ \vdots & \vdots & \ddots & \dots \\ \mathbf{W}_{M,1} & \mathbf{W}_{M,2} & \dots & \mathbf{W}_{M,M} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x}_1^{(0)} \\ \mathbf{x}_2^{(0)} \\ \vdots \\ \mathbf{x}_M^{(0)} \end{bmatrix} + \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_M \end{bmatrix}\right), \tag{6}$$

where for each $m, m' \in [M] := \{1, ..., M\}$, we have $\mathbf{x}_m^{(0)} \in \mathbb{R}^{d_0/M}$, $\mathbf{b}_m \in \mathbb{R}^{d_1/M}$, $\mathbf{x}_m^{(1)} \in \mathbb{R}^{d_1/M}$, and $\mathbf{W}_{m,m'} \in \mathbb{R}^{(d_0/M)\times(d_1/M)}$. In this representation, machine m begins with inputs $\mathbf{x}_m^{(0)}$, and must compute $\mathbf{x}_m^{(1)}$ with limited communication from the other machines. Likewise, we would like to minimize the number of parameters which must be stored on each machine.

If we write

$$\mathbf{x}_m^{(1)} = \sigma\left(\sum_{m'} \mathbf{W}_{m,m'} \cdot \mathbf{x}_{m'}^{(0)} + \mathbf{b}_m\right) = \sigma\left(\mathbf{W}_{m,m} \cdot \mathbf{x}_m^{(0)} + \sum_{m' \neq m} \mathbf{W}_{m,m'} \cdot \mathbf{x}_{m'}^{(0)} + \mathbf{b}_m\right), \tag{7}$$

we see that each machine $m' \neq m$ must communicate information about the vector $\mathbf{W}_{m,m'} \cdot \mathbf{x}_{m'}^{(0)}$ to machine m. One way to do this would be for machine m' to broadcast its input vector $\mathbf{x}_{m'}^{(0)}$, but this could incur a large amount of communication overhead. We will discuss a more efficient way to do this in the upcoming sections, but it will be helpful to establish some notation.

Let us define

$$\mathbf{W}_{m'} := \Big[\ \dots \ \mathbf{W}_{m,m'}^T \ \dots \ \Big]_{m\in[M]}^T \in \mathbb{R}^{d_1 \times \frac{d_0}{M}}, \tag{8}$$

$$\hat{\mathbf{W}}_{m'} := \Big[\mathbf{W}_{1,m'}^T \ \dots \ \mathbf{0}_{m',m'}^T \ \dots \ \mathbf{W}_{M,m'}^T \Big]_{m\in[M]}^T \in \mathbb{R}^{d_1 \times \frac{d_0}{M}}, \tag{9}$$

$$\tilde{\mathbf{W}}_{m'} := \Big[\ \dots \ \mathbf{W}_{m,m'}^T \ \dots \ \Big]_{m\neq m'}^T \in \mathbb{R}^{\frac{(M-1)\cdot d_1}{M} \times \frac{d_0}{M}}, \tag{10}$$

$$\tag{11}$$

so that $\mathbf{W}_{m'}$ represents the submatrix of columns of $\mathbf{W}$ allocated to machine m' (just as in our baseline method), $\hat{\mathbf{W}}_{m'}$ represents the same submatrix with the $\mathbf{W}_{m',m'}$ component zeroed-out, and $\tilde{\mathbf{W}}_{m'}$ represents the submatrix with the $\mathbf{W}_{m',m'}$ component removed entirely. This is illustrated graphically in Figure 3.

We will also define

$$\mathbf{w}_{m,m'} := \mathbf{W}_{m,m'} \cdot \mathbf{x}_{m'}^{(0)} \in \mathbb{R}^{d_1/M}, \tag{12}$$

$$\mathbf{w}_{m'} := \mathbf{W}_{m'} \cdot \mathbf{x}_{m'}^{(0)} = \Big[\ \dots \ \mathbf{w}_{m,m'}^T \ \dots \ \Big]_{m\in[M]}^T \in \mathbb{R}^{d_1}, \tag{13}$$

$$\hat{\mathbf{w}}_{m'} := \hat{\mathbf{W}}_{m'} \cdot \mathbf{x}_0^{(m')} = \Big[\mathbf{w}_{1,m'}^T \ \dots \ \mathbf{0}_{m',m'}^T \ \dots \ \mathbf{w}_{M,m'}^T \Big]_{m\in[M]}^T \in \mathbb{R}^{d_1}, \tag{14}$$

$$\tilde{\mathbf{w}}_{m'} := \tilde{\mathbf{W}}_{m'} \cdot \mathbf{x}_{m'}^{(0)} = \Big[\ \dots \ \mathbf{w}_{m,m'}^T \ \dots \ \Big]_{m\neq m'}^T \in \mathbb{R}^{\frac{(M-1)\cdot d_1}{M}}. \tag{15}$$

Intuitively, $\mathbf{w}_{m,m'}$ represents the parameters that machine m' must communicate to machine m, with $\mathbf{w}_{m',m'}$ being the parameters that m' requires for itself. $\mathbf{w}_{m'}$ concatenates all of these parameters into a single vector, while $\hat{\mathbf{w}}_{m'}$ zeroes-out the $\mathbf{w}_{m',m'}$ component, and $\tilde{\mathbf{w}}_{m'}$ removes it entirely.

For convenience, we will further define notation for "complements" of $\hat{\mathbf{W}}_{m'}$ and $\hat{\mathbf{w}}_{m'}$, which will be the portions of $\mathbf{W}_{m'}$ and $\mathbf{w}_{m'}$ with everything *except* the m' components zeroed out:

$$\hat{\mathbf{W}}_{m'}^c := \mathbf{W}_{m'} - \hat{\mathbf{W}}_{m'} = \Big[\mathbf{0}_{1,m'}^T \ \dots \ \mathbf{W}_{m',m'}^T \ \dots \ \mathbf{0}_{M,m'}^T \Big]_{m\in[M]}^T \in \mathbb{R}^{d_1 \times \frac{d_0}{M}}, \tag{16}$$

$$\hat{\mathbf{w}}_{m'}^c := \mathbf{w}_{m'} - \hat{\mathbf{w}}_{m'} = \Big[\mathbf{0}_{1,m'}^T \ \dots \ \mathbf{w}_{m',m'}^T \ \dots \ \mathbf{0}_{M,m'}^T \Big]_{m\in[M]}^T \in \mathbb{R}^{d_1}. \tag{17}$$

C. Lowering Communication Overhead, Memory Requirements, and Power Consumption Using Rank-Reduction

We will now discuss how to use rank-reduction methods on our weight matrices to reduce both the memory needed to store them on each machine, as well as the amount of information that must be communicated between them in order to jointly perform the full layer operation. We will also consider the effect on the amount of power consumed in applying a single layer of the network, as estimated by the number of floating-point operations (FLOPs) used.

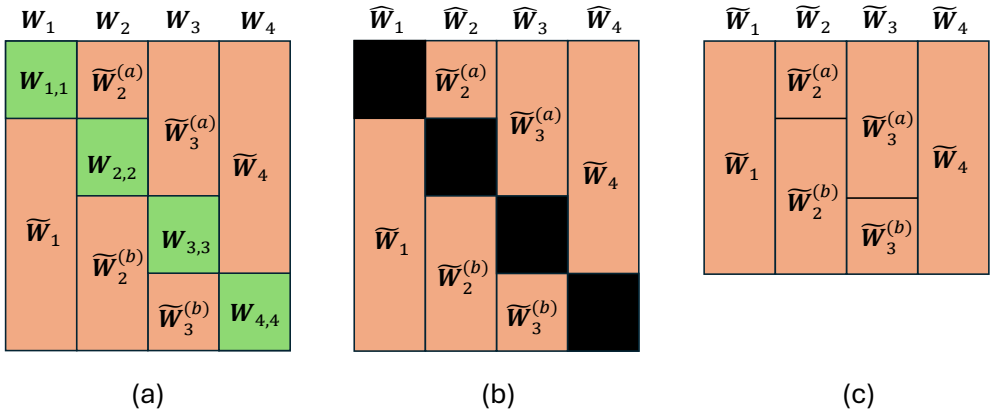


Figure 3. Partitioning a layer weight matrix amongst $M = 4$ machines. (a) Equal numbers of columns are allocated to each machine to create the matrices $\mathbf{W}_m$. (b) The $\mathbf{W}_{m,m}$ submatrices are replaced with zeros to create the matrices $\hat{\mathbf{W}}_m$. (c) The $\mathbf{W}_{m,m}$ submatrices are removed entirely to create the matrices $\tilde{\mathbf{W}}_m$.

In order to do this, we will be producing a low-rank approximation of each machine's portion of the full weight matrix. This will inevitably create some error in the full computation, which we will address in Section II.E.

1. Number of Communicated Parameters: Baseline SVD Method

A common method of compressing a weight matrix involves taking a low-rank singular value decomposition (SVD) approximation. To this end, consider taking a rank-r SVD approximation of each $\mathbf{W}_{m'}$,

$$\mathbf{W}_{m'} \approx \mathbf{W}_{m'}^{(r)} := \mathbf{U}_{m'}^{(r)} \cdot \mathbf{\Sigma}_{m'}^{(r)} \cdot \left(\mathbf{V}_{m'}^{(r)}\right)^T, \tag{18}$$

where $\mathbf{U}_{m'}^{(r)} \in \mathbb{R}^{d_1 \times r}$, $\mathbf{\Sigma}_{m'}^{(r)} \in \mathbb{R}^{r \times r}$, and $\left(\mathbf{V}_{m'}^{(r)}\right)^T \in \mathbb{R}^{r \times d_0/M}$. Then machine m' can store the product $\mathbf{\Sigma}_{m'}^{(r)} \cdot \left(\mathbf{V}_{m'}^{(r)}\right)^T \in \mathbb{R}^{r \times d_0/M}$, and communicate the r parameters

$$\mathbf{y}_{m'}^{(r)} := \mathbf{\Sigma}_{m'}^{(r)} (\mathbf{V}_{m'}^{(r)})^T \mathbf{x}_{m'}^{(0)} \in \mathbb{R}^{r \times 1} \tag{19}$$

to a central node which stores all the matrices $\mathbf{U}_{m'}^{(r)}$, $m' = 1, ..., M$, as well as the bias vector $\mathbf{b}^{(0)}$. This node may then compute the products

$$\mathbf{w}_{m'} \approx \mathbf{w}_{m'}^{(r)} := \mathbf{W}_{m'}^{(r)} \cdot \mathbf{x}_{m'}^{(0)} \tag{20}$$

$$= \mathbf{U}_{m'}^{(r)} \cdot \mathbf{\Sigma}_{m'}^{(r)} (\mathbf{V}_{m'}^{(r)})^T \mathbf{x}_{m'}^{(0)} \tag{21}$$

$$= \mathbf{U}_{m'}^{(r)} \cdot \mathbf{y}_{m'}^{(r)}, \tag{22}$$

and complete the layer computation in Eq. (4), communicating the d_1 parameters of $\mathbf{x}^{(1)}$ back to the M machines to perform the next layer's computation.

In summary, all machines except the central node must communicate a total of

$$\#\{\text{parameters transmitted per regular machine (baseline SVD method)}\} = r, \tag{23}$$

after which the central node communicates

$$\#\{\text{parameters transmitted by central machine (baseline SVD method)}\} = d_1. \quad (24)$$

Importantly, this must be done in two steps, since the central node cannot reply to the machines until it has processed all their inputs. This will incur some latency.

2. Number of Communicated Parameters: Our New Method

In our new approach, we can now consider the singular value decomposition

$$\tilde{\mathbf{W}}_{m'} = \tilde{\mathbf{U}}_{m'}\tilde{\mathbf{\Sigma}}_{m'}\tilde{\mathbf{V}}_{m'}^T, \quad (25)$$

from which we can derive the lower rank-r SVD approximation

$$\tilde{\mathbf{W}}_{m'}^{(r)} = \tilde{\mathbf{U}}_{m'}^{(r)}\tilde{\mathbf{\Sigma}}_{m'}^{(r)}(\tilde{\mathbf{V}}_{m'}^{(r)})^T \quad (26)$$

where $\tilde{\mathbf{U}}_{m'}^{(r)} \in \mathbb{R}^{((M-1)\cdot d_1/M)\times r}$ and $\tilde{\mathbf{V}}_{m'}^{(r)} \in \mathbb{R}^{(d_0/M)\times r}$ represent the leftmost r columns of $\tilde{\mathbf{U}}_{m'}$ and $\tilde{\mathbf{V}}_{m'}$ respectively, which are the r dominant left and right singular vectors. Likewise, $\tilde{\mathbf{\Sigma}}_{m'}^{(r)} \in \mathbb{R}^{r\times r}$ is the upper-left $r \times r$ submatrix of $\tilde{\mathbf{\Sigma}}_{m'}$, which is the diagonal matrix of the largest r singular values.

We can further rewrite $\tilde{\mathbf{U}}_{m'}^{(r)}$ in the form

$$\tilde{\mathbf{U}}_{m'}^{(r)} = \begin{bmatrix} \vdots \\ \tilde{\mathbf{U}}_{m,m'}^{(r)} \\ \vdots \end{bmatrix}_{m\neq m'}, \quad (27)$$

where $\tilde{\mathbf{U}}_{m,m'}^{(r)} \in \mathbb{R}^{(d_1/M)\times r}$ is the block of d_1/M rows of $\tilde{\mathbf{U}}_{m'}^{(r)}$ corresponding to the rows of $\mathbf{W}_{m,m'}$, so that

$$\mathbf{W}_{m,m'} \approx \tilde{\mathbf{W}}_{m,m'}^{(r)} := \tilde{\mathbf{U}}_{m,m'}^{(r)}\tilde{\mathbf{\Sigma}}_{m'}^{(r)}(\tilde{\mathbf{V}}_{m'}^{(r)})^T. \quad (28)$$

In this form, we see that if $\tilde{\mathbf{U}}_{m,m'}^{(r)}$ and $\mathbf{b}_m$ are stored on machine m, and the matrix product $\tilde{\mathbf{\Sigma}}_{m'}^{(r)}(\tilde{\mathbf{V}}_{m'}^{(r)})^T \in \mathbb{R}^{r\times(d_0/M)}$ is stored on machine m', then machine m' need only broadcast the r parameters given by the vector

$$\tilde{\mathbf{y}}_{m'}^{(r)} = \tilde{\mathbf{\Sigma}}_{m'}^{(r)}(\tilde{\mathbf{V}}_{m'}^{(r)})^T\mathbf{x}_{m'}^{(0)} \in \mathbb{R}^{r\times 1}, \quad (29)$$

from which machine m can compute the approximation

$$\begin{aligned} \mathbf{w}_{m,m'} \approx \tilde{\mathbf{w}}_{m,m'}^{(r)} &:= \tilde{\mathbf{W}}_{m,m'}^{(r)} \cdot \mathbf{x}_{m'}^{(0)} && (30)\\ &= \tilde{\mathbf{U}}_{m,m'}^{(r)} \cdot \left(\tilde{\mathbf{\Sigma}}_{m'}^{(r)}(\tilde{\mathbf{V}}_{m'}^{(r)})^T\mathbf{x}_{m'}^{(0)}\right) && (31)\\ &= \tilde{\mathbf{U}}_{m,m'}^{(r)} \cdot \tilde{\mathbf{y}}_{m'}^{(r)}, && (32) \end{aligned}$$

and subsequently can compute

$$\mathbf{x}_1^{(m)} \approx \sigma\left(\mathbf{w}_{m,m} + \sum_{m'\neq m} \tilde{\mathbf{w}}_{m,m'}^{(r)} + \mathbf{b}_m\right). \quad (33)$$

In summary, for a single layer of a feed-forward network, Eq. (29) shows that each machine will bear a communication overhead of

$$\#\{\text{parameters broadcast per machine (our method)}\} = r. \tag{34}$$

Importantly, we no longer incur the delays associated with needing to wait for a single central node to compute and transmit the layer outputs, as in the baseline SVD method.

3. Number of Stored Parameters: Baseline SVD Method

Each machine other than the central node must store the matrix $\mathbf{\Sigma}_{m'}^{(r)} \cdot \left(\mathbf{V}_{m'}^{(r)}\right)^T$, while the central node itself (assuming it is machine $m_c \in [M]$) stores

- $r \times \frac{d_0}{M}$ parameters for the matrix $\mathbf{\Sigma}_{m_c}^{(r)} \cdot \left(\mathbf{V}_{m_c}^{(r)}\right)^T$,
- $M \times d_1 \times r$ parameters for each of the matrices $\mathbf{U}_{m'}^{(r)}$, $m' = 1, ..., M$,
- d_1 parameters for the bias vector $\mathbf{b}^{(0)}$.

This gives

$$\#\{\text{parameters stored per regular machine (baseline SVD method)}\} = r \cdot \frac{d_0}{M}, \tag{35}$$

$$\#\{\text{parameters stored in central machine (baseline SVD method)}\} = r \cdot \frac{d_0}{M} + (Mr + 1) \cdot d_1, \tag{36}$$

for a total of

$$\#\{\text{parameters stored in all machines (baseline SVD method)}\} = r \cdot d_0 + (Mr + 1) \cdot d_1. \tag{37}$$

One important consideration in the baseline SVD method is that the central node incurs a much higher burden in both the number of parameters to communicate and the amount of data stored, which may be a limiting factor in practical architectures.

Let us define the ratio

$$\kappa := d_1/d_0, \tag{38}$$

and simplify our notation by setting $d := d_0$. If we fix $M \cdot r$ to be a fixed fraction γ of d,

$$\gamma := \frac{Mr}{d}, \tag{39}$$

then for large-enough M, we can approximate the total number of parameters as

$$\#\{\text{parameters stored in all machines (baseline SVD method)}\} = \frac{\gamma d^2}{M} + (\gamma d + 1)\kappa d \tag{40}$$

$$\approx \gamma\kappa d^2 + \kappa d. \tag{41}$$

4. Number of Stored Parameters: Our New Method

In our new method, the total storage overhead for machine m will be

- $\frac{d_0}{M} \times \frac{d_1}{M}$ parameters for the matrix $\mathbf{W}_{m,m}$,
- $r \times \frac{d_0}{M}$ parameters for the matrix $\tilde{\mathbf{\Sigma}}^{(r)}_{m'}(\tilde{\mathbf{V}}^{(r)}_{m'})^T$,
- $\frac{d_1}{M} \times r$ parameters for each matrix $\tilde{\mathbf{U}}^{(r)}_{m,m'}$, $m' \neq m$, and
- $\frac{d_1}{M}$ parameters for the vector $\mathbf{b}_m$.

This gives a total of

$$\#\{\text{parameters stored per machine (our method)}\} = \frac{d_0 \cdot d_1}{M^2} + \frac{r \cdot d_0}{M} + (M-1) \cdot \frac{r \cdot d_1}{M} + \frac{d_1}{M}, \tag{42}$$

and totaling over all M machines, we get a total of

$$\#\{\text{parameters stored in all machines (our method)}\} = \frac{d_0 \cdot d_1}{M} + r \cdot d_0 + (M-1) \cdot r \cdot d_1 + d_1. \tag{43}$$

If we again set $d := d_0$, and define κ and γ as in Eq. (38) and (39), then for large-enough M, we can approximate the total number of parameters as

$$\#\{\text{parameters stored in all machines}\} = \left(\gamma\kappa + \frac{\kappa \cdot (1-\gamma) + \gamma}{M}\right) d^2 + \kappa d \tag{44}$$

$$\approx \gamma\kappa d^2 + \kappa d. \tag{45}$$

Comparing to Eq. (41), we see that our new method has a comparable number of stored parameters. Comparing this to the original cost of storing the full matrix $\mathbf{W}$, which has κd^2 parameters, we see that both methods reduce this by roughly a fraction γ, which represents the reduction in required memory.

5. Number of FLOPs: Baseline SVD Method

We will estimate the number of FLOPs using the fact that multiplying an $N_{\text{rows}} \times N_{\text{columns}}$ matrix by a vector of dimensions $N_{\text{columns}} \times 1$ consists of $N_{\text{rows}} \cdot N_{\text{columns}}$ floating-point multiplications and approximately the same number of floating-point additions, for a total of about $2N_{\text{rows}} \cdot N_{\text{columns}}$ FLOPs. Likewise, adding two $N_{\text{rows}} \times 1$ vectors uses N_{rows} FLOPs.

In the baseline SVD method, for a regular machine (other than the central node), the majority of the FLOPs come from the matrix-vector multiplication

$\left(\mathbf{\Sigma}_{m'}^{(r)}(\mathbf{V}_{m'}^{(r)})^T\right) \cdot \mathbf{x}_{m'}^{(0)}$, where we assume the matrix $\mathbf{\Sigma}_{m'}^{(r)}(\mathbf{V}_{m'}^{(r)})^T$ has been pre-computed. This will produce a number of FLOPs equal to

$$\#\{\text{FLOPs per regular machine (baseline SVD method)}\} \approx \frac{2rd_0}{M}. \tag{46}$$

The central machine must perform

- the matrix-vector multiplication $\left(\mathbf{\Sigma}_{m'}^{(r)}(\mathbf{V}_{m'}^{(r)})^T\right) \cdot \mathbf{x}_{m'}^{(0)}$: $\sim \frac{2 \cdot r \cdot d_0}{M}$ FLOPs,
- the M matrix-vector multiplications $\mathbf{U}_{m'}^{(r)} \cdot \mathbf{y}_{m'}^{(r)}$, $m' \in [M]$: $\sim 2Mrd_1$ FLOPs,
- the vector addition $\sum_{m'=1}^{M} \mathbf{w}_{m'}^{(r)} + \mathbf{b}^{(0)}$: $\sim Md_1$ FLOPs,

yielding

$$\#\{\text{FLOPs for central machine (baseline SVD method)}\} \approx \frac{2rd_0}{M} + 2Mrd_1 + Md_1. \tag{47}$$

This gives a total over all M machines of

$$\#\{\text{FLOPs for all machines (baseline SVD method)}\} \approx 2rd_0 + 2Mrd_1 + Md_1. \tag{48}$$

Again setting d_0 to 'd' and defining κ and γ as in Eq. (38) and (39), we can approximate this for large M as

$$\#\{\text{FLOPs for all machines (baseline SVD method)}\} \approx \frac{2\gamma d^2}{M} + 2M\gamma\kappa d^2 + \frac{\gamma\kappa d^2}{r} \tag{49}$$

$$\approx \gamma \cdot \left(2 + \frac{1}{r}\right) \cdot \kappa d^2 \tag{50}$$

6. Number of FLOPs: Our New Method

In our new method, considering the number of FLOPs machine m must implement for its share of this collective computation, we note that the bulk of the operations for machine come from

- the matrix-vector multiplication $\mathbf{W}_{m,m} \cdot \mathbf{x}_m^{(0)}$: $\sim \frac{2d_0d_1}{M^2}$ FLOPs,
- the matrix-vector multiplications $\tilde{\mathbf{U}}_{m,m'}^{(r)} \cdot \tilde{\mathbf{y}}_{m'}^{(r)}$ for $m' \neq m$: $\sim \frac{2(M-1) \cdot r \cdot d_1}{M}$ FLOPs,
- the vector addition $\mathbf{w}_{m,m} + \sum_{m' \neq m} \tilde{\mathbf{w}}_{m,m'}^{(r)} + \mathbf{b}_m$: $\sim d_1$ FLOPs.

In total, we see that in our method, each machine uses a number of FLOPs equal to

$$\#\{\text{FLOPs per machine (our method)}\} \approx \frac{2d_0d_1}{M^2} + \frac{2(M-1) \cdot r \cdot d_1}{M} + d_1. \tag{51}$$

This is highly correlated with the power consumption for each machine to implement the dense layer. From Eq. (51), we see that as the number of machines M grows, we converge to roughly $(2r+1)\cdot d_1$ FLOPs per machine.

Totaling over all M machines gives us

$$\#\{\text{FLOPs for all machines (our method)}\} \approx \frac{2d_0d_1}{M} + 2(M-1)\cdot r\cdot d_1 + M\cdot d_1. \quad (52)$$

Setting d_0 to 'd,' and defining κ and γ as in Eqs. (38) and (39), we can approximate the total number of FLOPs as

$$\#\{\text{FLOPs for all machines (our method)}\} = \left(\frac{2\kappa}{M} + \frac{2(M-1)\kappa\gamma}{M} + \frac{\kappa\gamma}{r}\right)\cdot d^2 \quad (53)$$

$$\approx \gamma\cdot\left(2+\frac{1}{r}\right)\cdot\kappa d^2. \quad (54)$$

This is identical to the counterpart from the baseline SVD method from Eq. (50). Noting that the full matrix multiplication $\mathbf{W}\cdot\mathbf{x}_0$ has approximately $2\kappa d^2$ FLOPs, we see that both methods have reduced this by a factor of approximately γ, representing the corresponding reduction in required power.

D. Model Parameter Reduction Over Full SVD Approximation

Since both the baseline SVD approximation and our new method employ M rank-r matrix approximations to summarize the full weight matrix, a natural question to ask is how they might compare to taking a rank-Mr singular value approximation of the full matrix $\mathbf{W}$, which is a common approach to model parameter reduction in neural networks. To this end, we will consider only the number of parameters needed to store our approximations of $\mathbf{W}$ for each of these methods.

The traditional rank-Mr SVD will decompose $\mathbf{W}$ as

$$\mathbf{W} \approx \mathbf{U}^{(r)}\cdot\mathbf{\Sigma}^{(r)}\cdot\left(\mathbf{V}^{(r)}\right)^T, \quad (55)$$

and storing only $\mathbf{U}^{(r)}\in\mathbb{R}^{d_1\times r}$ and the product $\mathbf{\Sigma}^{(r)}\cdot\left(\mathbf{V}^{(r)}\right)^T\in\mathbb{R}^{r\times d_0}$, we summarize the full weight matrix $\mathbf{W}$ with a total number of parameters equal to

$$N_0 = M\cdot r\cdot(d_0+d_1). \quad (56)$$

The baseline SVD method, similarly, stores $\mathbf{U}^{(r)}_{m'}\in\mathbb{R}^{d_1\times r}$ and the product $\mathbf{\Sigma}^{(r)}_{m'}\cdot\left(\mathbf{V}^{(r)}_{m'}\right)^T\in\mathbb{R}^{r\times d_0/M}$ for each $m'=1,...,M$, summarizing the full weight matrix with a total number of parameters

$$N_1 = M\cdot r\cdot\left(\frac{d_0}{M}+d_1\right). \quad (57)$$

In our new proposed approach, for each $\mathbf{W}_{m'}$, we store the $\frac{d_1}{M} \times \frac{d_0}{M}$ matrix $\mathbf{W}_{m',m'}$, as well as the $\frac{(M-1)d_1}{M} \times r$ matrix $\tilde{\mathbf{U}}_{m'}^{(r)}$ and the $r \times \frac{d_0}{M}$ matrix product $\mathbf{\Sigma}_{m'}^{(r)} \left(\tilde{\mathbf{V}}_{m'}^{(r)}\right)^T$ from Eq. (26). This gives us a total parameter count of

$$N_2 = M \cdot \left[\left(\frac{d_1}{M} \cdot \frac{d_0}{M}\right) + \left(\frac{(M-1)d_1}{M} \cdot r\right) + \left(r \cdot \frac{d_0}{M}\right)\right] \tag{58}$$

$$= \frac{d_1 \cdot d_0}{M} + (M-1) \cdot d_1 \cdot r + r \cdot d_0. \tag{59}$$

Note that the expressions for N_1 and N_2 contain all the terms of Eq. (37) and (43), respectively, except the ones corresponding to the parameters of the bias vectors $\mathbf{b}^{(0)}$ and $\mathbf{b}_{m'}$, $m' \in [M]$.

Setting $d_0 = d$ and defining κ and γ as in Eq. (38) and (39) we may verify that

$$N_0 = \gamma(\kappa + 1)d^2, \tag{60}$$

$$N_1 = \gamma\left(\kappa + \frac{1}{M}\right)d^2, \tag{61}$$

$$N_2 = \left(\gamma\kappa + \frac{\kappa \cdot (1-\gamma) + \gamma}{M}\right)d^2. \tag{62}$$

This indicates that, as we distribute the network layer over a high-enough number M of partitions, we have

$$N_1 \approx N_2 \approx \gamma\kappa d^2, \tag{63}$$

so the two methods have comparable numbers of total stored parameters. But both tend to have substantially lower parameter counts than taking a traditional SVD approximation with a comparable number of singular vectors. As an example, for the case $\kappa = 1$ (so that $\mathbf{W}$ is a square matrix), then for large M the baseline SVD method and our rank-reduction method will both have about half as many parameters stored as the full SVD method (γd^2 compared to $2\gamma d^2$ parameters).

As another example, in a typical transformer network, there are often pairs of layers with $d_0 = d$, $d_1 = 4d$, and $d_2 = d$. This means that there are two weight matrices, $\mathbf{W}^{(0)} \in \mathbf{R}^{4d \times d}$ (with $\kappa^{(0)} = 4$) and $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times 4d}$ (with $\kappa^{(1)} = \frac{1}{4}$). In this case, we can see that the baseline SVD method and our new method will have roughly $4\gamma d^2 + \frac{1}{4}\gamma d^2 = \frac{17}{4}\gamma d^2$ stored parameters. The full SVD method will have about $5\gamma d^2 + \frac{5}{4}\gamma d^2 = \frac{25}{4}\gamma d^2$ stored parameters. In this case, the baseline SVD method and our new method both have less than half the total number of parameters stored than in the full SVD method.

In Figure 4, we plot N_0, N_1, and N_2 as fractions of d^2, for the case $\kappa = 1$ (a square weight matrix for simplicity) and $\gamma = 1/3$. We see that for very low values of M, our method stores more parameters than both the baseline SVD approximation and the full SVD approximation, but it quickly converges to the number of parameters of the baseline SVD approximation as M grows. Both fall to half the number of parameters of the full rank-Mr SVD approximation, which remains constant.

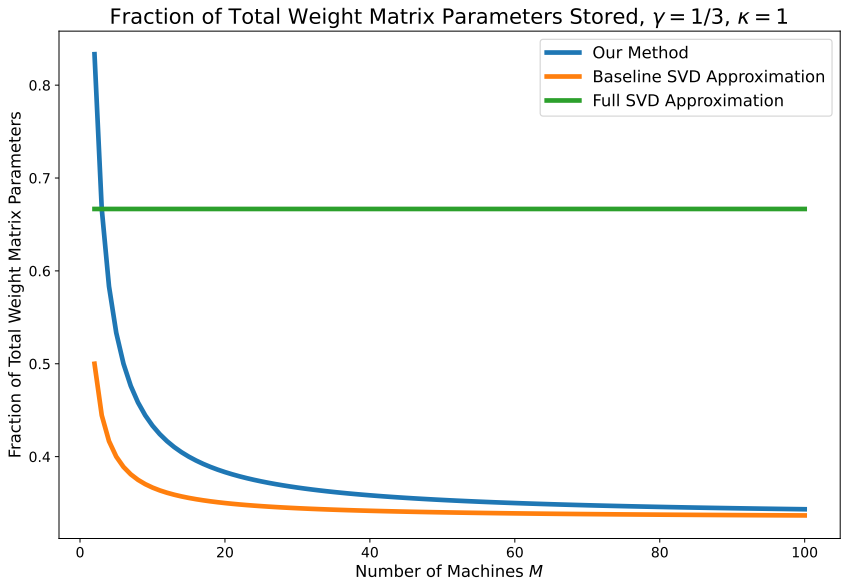


Figure 4. Amount of compression applied to a dense layer's weight matrix as a function of the number of machines M used to partition it, expressed as a fraction of the total number of parameters of the original weight matrix. We assume a square weight matrix ($\kappa = 1$), and compare our method to a SVD approximation of each matrix $\mathbf{W}_{m'}$ as well as a full SVD approximation of $\mathbf{W}$, assuming the total approximation rank Mr is one third the number of weight matrix columns d in all cases.

E. Approximation Error

We have now shown that our new method of decomposing the weight matrix $\mathbf{W}$ has approximately the same memory and power-reductions as the baseline SVD method for a fixed rank r used to approximate each of the M respective submatrices in both methods. Furthermore, we have shown that our method distributes the stored parameters and the computation more evenly than the baseline SVD approach, and has the advantage of allowing for low-latency reduced parameter communication. We will now go one step further, and argue that by selecting the rows of the submatrices $\mathbf{W}_{m,m}$ wisely, our method also achieves lower error in the network performance for the same rank approximation. This could allow us to lower the parameter count and number of FLOPs further than the baseline SVD method, while achieving comparable performance.

We begin by analyzing the approximation error in our new method. For convenience, we will define

$$\tilde{\mathbf{w}}_{m'}^{(r)} := \tilde{\mathbf{W}}_{m'}^{(r)} \cdot \mathbf{x}_0^{(m')} = \left[\dots \left(\tilde{\mathbf{w}}_{m,m'}^{(r)}\right)^T \dots \right]_{m \neq m'}^T, \tag{64}$$

$$\hat{\mathbf{w}}_{m'}^{(r)} := \left[\left(\tilde{\mathbf{w}}_{1,m'}^{(r)}\right)^T \dots \mathbf{0}_{m',m'}{}^T \dots \left(\tilde{\mathbf{w}}_{M,m'}^{(r)}\right)^T \right]^T, \tag{65}$$

which are our approximations of $\tilde{\mathbf{w}}_{m'}$ and $\hat{\mathbf{w}}_{m'}$, respectively. Then we approximate $\mathbf{w}_{m'}$ as

$$\mathbf{w}_{m'} = \hat{\mathbf{w}}^c_{m'} + \hat{\mathbf{w}}_{m'} \tag{66}$$

$$\approx \hat{\mathbf{w}}^c_{m'} + \hat{\mathbf{w}}^{(r)}_{m'}. \tag{67}$$

The linear transformation of our full dense layer can now be approximated as

$$\mathbf{W} \cdot \mathbf{x}^{(0)} = \sum_{m'=1}^{M} \mathbf{W}_{m'} \cdot \mathbf{x}^{(0)}_{m'} \tag{68}$$

$$= \sum_{m'=1}^{M} \mathbf{w}_{m'} \tag{69}$$

$$= \sum_{m'=1}^{M} \hat{\mathbf{w}}^c_{m'} + \sum_{m'=1}^{M} \hat{\mathbf{w}}_{m'} \tag{70}$$

$$\approx \sum_{m'=1}^{M} \hat{\mathbf{w}}^c_{m'} + \sum_{m'=1}^{M} \hat{\mathbf{w}}^{(r)}_{m'}. \tag{71}$$

This has approximation error

$$\mathbf{\Delta} := \sum_{m'=1}^{M} \left(\hat{\mathbf{w}}_{m'} - \hat{\mathbf{w}}^{(r)}_{m'} \right), \tag{72}$$

and we will denote each term of the summand as

$$\mathbf{\Delta}_{m'} := \hat{\mathbf{w}}_{m'} - \hat{\mathbf{w}}^{(r)}_{m'}. \tag{73}$$

Each $\mathbf{\Delta}_{m'}$ has nonzero component equal to $\tilde{\mathbf{w}}_{m'} - \tilde{\mathbf{w}}^{(r)}_{m'}$, which can be rewritten as

$$\tilde{\mathbf{w}}_{m'} - \tilde{\mathbf{w}}^{(r)}_{m'} = \tilde{\mathbf{U}}^{(>r)}_{m'} \tilde{\mathbf{\Sigma}}^{(>r)}_{m'} \left(\tilde{\mathbf{V}}^{(>r)}_{m'} \right)^T \cdot \mathbf{x}^{(0)}_{m'} \tag{74}$$

where $\tilde{\mathbf{\Sigma}}^{(>r)}_{m'}$ is the diagonal matrix consisting of the lower singular values of $\tilde{\mathbf{W}}_{m'}$ (after the r dominant singular values included in $\tilde{\mathbf{\Sigma}}^{(r)}_{m'}$), with the columns of $\tilde{\mathbf{U}}^{(>r)}_{m'}$ and $\tilde{\mathbf{V}}^{(>r)}_{m'}$ being the corresponding less dominant left and right singular vectors. If we define

$$d_* := \min \left\{ \frac{(M-1) \cdot d_1}{M},\ \frac{d_0}{M} \right\}, \tag{75}$$

then the dimensions of these matrices will be

$$\tilde{\mathbf{\Sigma}}^{(>r)}_{m'} \in \mathbb{R}^{(d_* - r) \times (d_* - r)}, \tag{76}$$

$$\tilde{\mathbf{U}}^{(>r)}_{m'} \in \mathbb{R}^{((M-1) \cdot d_1 / M) \times (d_* - r)}, \tag{77}$$

$$\tilde{\mathbf{V}}^{(>r)}_{m'} \in \mathbb{R}^{(d_0 / M) \times (d_* - r)}. \tag{78}$$

We will typically be interested in the regime where M is large enough that $\frac{d_0}{M} < \frac{(M-1) \cdot d_1}{M}$, so that $d_* = \frac{d_0}{M}$.

The norm-squared error in our approximation can be bounded as

$$\left\|\hat{\mathbf{w}}_{m'} - \hat{\mathbf{w}}_{m'}^{(r)}\right\|_2^2 = \left\|\tilde{\mathbf{w}}_{m'} - \tilde{\mathbf{w}}_{m'}^{(r)}\right\|_2^2 \tag{79}$$

$$= \left(\mathbf{x}_{m'}^{(0)}\right)^T \tilde{\mathbf{V}}_{m'}^{(>r)} \left(\tilde{\mathbf{\Sigma}}_{m'}^{(>r)}\right)^2 \left(\tilde{\mathbf{V}}_{m'}^{(>r)}\right)^T \mathbf{x}_{m'}^{(0)} \tag{80}$$

$$\leq \sigma_{m',r+1}^2 \cdot \left\|\left(\tilde{\mathbf{V}}_{m'}^{(>r)}\right)^T \mathbf{x}_{m'}^{(0)}\right\|_2^2 \tag{81}$$

$$\leq \sigma_{m',r+1}^2 \cdot \left\|\tilde{\mathbf{V}}_{m'}^T \mathbf{x}_{m'}^{(0)}\right\|_2^2 \tag{82}$$

$$= \sigma_{m',r+1}^2 \cdot \left\|\mathbf{x}_{m'}^{(0)}\right\|_2^2, \tag{83}$$

where $\sigma_{m',r+1}$ is the $(r+1)^{st}$ singular value of $\tilde{\mathbf{W}}_{m'}$.

For simplicity, we will assume that we use the same value of r for each m'. Our total error is now bounded by the triangle inequality as

$$\|\mathbf{\Delta}\|_2 \leq \sum_{m'=1}^{M} \left\|\hat{\mathbf{w}}_{m'} - \hat{\mathbf{w}}_{m'}^{(r)}\right\|_2 \tag{84}$$

$$\leq \sum_{m'=1}^{M} \sigma_{m',r+1} \cdot \left\|\mathbf{x}_{m'}^{(0)}\right\|_2. \tag{85}$$

Since each $\mathbf{x}_{m'}^{(0)}$ is a subvector of $\mathbf{x}^{(0)}$ for each m', then $\|\mathbf{x}_{m'}^{(0)}\|_2 \leq \|\mathbf{x}^{(0)}\|_2$, so we may further write

$$\|\mathbf{\Delta}\|_2 \leq \left(\sum_{m'=1}^{M} \sigma_{m',r+1}\right) \cdot \left\|\mathbf{x}^{(0)}\right\|_2. \tag{86}$$

In the case where the $\sigma_{m',r+1}$ are of similar magnitude, we can obtain a tighter bound as follows: if we set $S = \sum_{m'} \sigma_{m',r+1}$ and $\sigma_{\max} := \max_{m'}\{\sigma_{m',r+1}\}$, then Eq. (85) becomes

$$\|\mathbf{\Delta}\|_2 \leq S \cdot \sum_{m'=1}^{M} \frac{\sigma_{m',r+1}}{S} \cdot \sqrt{\left\|\mathbf{x}_{m'}^{(0)}\right\|_2^2} \tag{87}$$

$$\leq S \cdot \sqrt{\sum_{m'=1}^{M} \frac{\sigma_{m',r+1}}{S} \cdot \left\|\mathbf{x}_{m'}^{(0)}\right\|_2^2} \tag{88}$$

$$\leq \sqrt{\sigma_{\max} \cdot S} \cdot \sqrt{\sum_{m'=1}^{M} \left\|\mathbf{x}_{m'}^{(0)}\right\|_2^2} \tag{89}$$

$$= \sqrt{\sigma_{\max} \cdot \sum_{m'=1}^{M} \sigma_{m',r+1}} \cdot \left\|\mathbf{x}^{(0)}\right\|_2 \tag{90}$$

$$\leq \sqrt{M \cdot \sigma_{\max}^2} \cdot \left\|\mathbf{x}^{(0)}\right\|_2 \tag{91}$$

$$= \sqrt{M} \cdot \sigma_{\max} \cdot \left\|\mathbf{x}^{(0)}\right\|_2, \tag{92}$$

where Eq. (88) follows from Jensen's inequality and Eq. (90) follows from the fact that the vectors $\mathbf{x}_{m'}^{(0)}$ form the complete set of disjoint segments of $\mathbf{x}^{(0)}$. We see that when

the $\sigma_{m',r+1}$ are close in magnitude (so that $\sum_{m'} \sigma_{m',r+1} \approx M \cdot \sigma_{\max}$), then the bound in Eq. (92) becomes lower than that of Eq. (86) by a factor of about $\sqrt{M}$.

1. Relative Error

The final network performance will typically be a function of the relative error in the layer's output:

$$\delta := \frac{\|\mathbf{\Delta}\|_2}{\|\mathbf{W} \cdot \mathbf{x}^{(0)}\|_2}, \tag{93}$$

which we may bound using Eq. (92) as

$$\delta \leq \sqrt{M} \cdot \sigma_{\max} \cdot \frac{\left\|\mathbf{x}^{(0)}\right\|_2}{\|\mathbf{W} \cdot \mathbf{x}^{(0)}\|_2}. \tag{94}$$

Alternatively, we may consider each machine's individual relative error,

$$\delta_{m'} := \frac{\|\mathbf{\Delta}_{m'}\|_2}{\|\mathbf{W}_{m'} \cdot \mathbf{x}^{(0)}\|_2} \tag{95}$$

$$= \frac{\|\hat{\mathbf{w}}_{m'} - \hat{\mathbf{w}}_{m'}^{(r)}\|_2}{\|\mathbf{w}_{m'}\|_2}. \tag{96}$$

The square of $\delta_{m'}$ takes the form

$$\delta_{m'}^2 = \frac{\|\hat{\mathbf{w}}_{m'} - \hat{\mathbf{w}}_{m'}^{(r)}\|_2^2}{\|\mathbf{W}_{m',m'} \cdot \mathbf{x}_{m'}^{(0)}\|_2^2 + \|\hat{\mathbf{W}}_{m'} \cdot \mathbf{x}_{m'}^{(0)}\|_2^2} \tag{97}$$

$$= \frac{\|\tilde{\mathbf{w}}_{m'} - \tilde{\mathbf{w}}_{m'}^{(r)}\|_2^2}{\|\mathbf{W}_{m',m'} \cdot \mathbf{x}_{m'}^{(0)}\|_2^2 + \|\tilde{\mathbf{W}}_{m'} \cdot \mathbf{x}_{m'}^{(0)}\|_2^2} \tag{98}$$

$$= \frac{\left(\mathbf{x}_{m'}^{(0)}\right)^T \tilde{\mathbf{V}}_{m'}^{(>r)} \left(\tilde{\mathbf{\Sigma}}_{m'}^{(>r)}\right)^2 \left(\tilde{\mathbf{V}}_{m'}^{(>r)}\right)^T \mathbf{x}_{m'}^{(0)}}{\left(\mathbf{x}_{m'}^{(0)}\right)^T \mathbf{W}_{m',m'}^T \mathbf{W}_{m',m'} \cdot \mathbf{x}_{m'}^{(0)} + \left(\mathbf{x}_{m'}^{(0)}\right)^T \tilde{\mathbf{V}}_{m'} \tilde{\mathbf{\Sigma}}_{m'}^2 \tilde{\mathbf{V}}_{m'}^T \mathbf{x}_{m'}^{(0)}}. \tag{99}$$

While this quantity largely depends on the statistics of $\mathbf{x}^{(0)}$, we see that in general we can reduce the terms $\delta_{m'}$ by partitioning our weight matrix $\mathbf{W}$ such that $\text{Tr}\left(\mathbf{W}_{m',m'}^T \mathbf{W}_{m',m'}\right)$ is maximized and so that the singular values of $\tilde{\mathbf{W}}_{m'}$ (equal to the eigenvalues of $\tilde{\mathbf{V}}_{m'} \tilde{\mathbf{\Sigma}}_{m'}^2 \tilde{\mathbf{V}}_{m'}^T = \sum_{m \neq m'} \mathbf{W}_{m,m'}^T \mathbf{W}_{m,m'}$) taper quickly.

F. Selecting the $\mathbf{W}_{m,m}$ to Minimize Approximation Error

We would like to partition the indices of $\mathbf{x}^{(0)}$ and $\mathbf{x}^{(1)}$ amongst the M machines to minimize the number of parameters which need be communicated between them with negligible degradation of the network performance. In some cases, it may make sense to constrain the sets of parameters allocated to a given machine. For instance, a network may have a residual connection in which the input $\mathbf{x}_0$ to a layer is added to the output $\mathbf{x}^{(n)}$ of a later layer with equal dimension $d_n = d_0$. This is the case in transformer-based models, which sport a residual connection after two dense layers in

their feed-forward network. In this case, it is probably best to assign the n^{th} layer's output indices to the same machines which host the corresponding input indices of $\mathbf{x}^{(0)}$, so that machine m can simply add $\mathbf{x}_m^{(0)} + \mathbf{x}_m^{(n)}$, and all M machines can form the full vector sum $\mathbf{x}^{(0)} + \mathbf{x}^{(n)}$ in concert.

We will consider a greedy approach to assigning outputs of a single dense layer to each machine, assuming the layer's inputs have already been designated. Our motivation is for the case of deep networks of concatenated dense layers, where the inputs to the first layer might arise from sensor measurements taken by the machine. In our approach, given each of the M machines' d_{i-1}/M input indices to the i^{th} layer, we must select its d_i/M output indices which partition the vector $\mathbf{x}^{(i)}$. We will take a local approach by which we assign the outputs of each layer successively, so we will focus on the case $i = 1$. Note that in this setup, the matrices $\mathbf{W}_{m'}$ are pre-determined, and our problem reduces to partitioning their rows into the $\mathbf{W}_{m,m'}$ components for $m = 1, ..., M$.

As before, we will impose the simplifying constraint that we select the same number r of singular values in our approximation of $\tilde{\mathbf{W}}_{m'}$ for each m'. As noted before, each machine need only broadcast r parameters corresponding to the entries of the vector $\tilde{\mathbf{y}}_{m'}^{(r)}$ in Eq. (29). In light of our bound on approximation error in Eq. (92), for a fixed tolerance level of $\epsilon := \frac{\|\Delta\|_2}{\|\mathbf{x}_0\|_2}$, we would like to choose r to be minimal such that

$$\max_{m'} \{\sigma_{m',r+1}\} \leq \frac{\epsilon}{\sqrt{M}} \tag{100}$$

$$\iff \max_{m'} \{\sigma_{m',r+1}^2\} \leq \frac{\epsilon^2}{M}. \tag{101}$$

Intuitively, we would like to divide the d_1 output indices of $\mathbf{x}_1$ amongst the M machines such that the singular values of each $\tilde{\mathbf{W}}_{m'}$ (equal to the eigenvalues of $\sum_{m \neq m'} \mathbf{W}_{m,m'}^T \mathbf{W}_{m,m'}$) taper as quickly as possible. In light of our expression for $\delta_{m'}^2$ in Eq. (99), we would also like for the eigenvalues of $\mathbf{W}_{m',m'}^T \mathbf{W}_{m',m'}$ to be large in order to minimize the relative error of each component $\mathbf{w}_{m'}$.

It turns out that these conditions are compatible with each other. Indeed, if we write

$$\mathbf{W}_{m'}^T \mathbf{W}_{m'} = \mathbf{W}_{m',m'}^T \mathbf{W}_{m',m'} + \sum_{m \neq m'} \mathbf{W}_{m,m'}^T \mathbf{W}_{m,m'}, \tag{102}$$

we see that the sum of the eigenvalues of $\sum_{m \neq m'} \mathbf{W}_{m,m'}^T \mathbf{W}_{m,m'}$ is given by

$$\text{Tr}\left(\sum_{m \neq m'} \mathbf{W}_{m,m'}^T \mathbf{W}_{m,m'} \right) = \text{Tr}\left(\mathbf{W}_{m'}^T \mathbf{W}_{m'} \right) - \text{Tr}\left(\mathbf{W}_{m',m'}^T \mathbf{W}_{m',m'} \right), \tag{103}$$

and the individual eigenvalues of $\sum_{m \neq m'} \mathbf{W}_{m,m'}^T \mathbf{W}_{m,m'}$ fall in the range

$$\left[\lambda_{\min}\left(\mathbf{W}_{m'}^T \mathbf{W}_{m'} \right) - \lambda_{\min}\left(\mathbf{W}_{m',m'}^T \mathbf{W}_{m',m'} \right),\ \lambda_{\max}\left(\mathbf{W}_{m'}^T \mathbf{W}_{m'} \right) - \lambda_{\min}\left(\mathbf{W}_{m',m'}^T \mathbf{W}_{m',m'} \right) \right], \tag{104}$$

where $\lambda_{\min}(\cdot)$ and $\lambda_{\max}(\cdot)$ represent the minimum and maximum eigenvalues of a matrix. Note that since $\sum_{m \neq m'} \mathbf{W}_{m,m'}^T \mathbf{W}_{m,m'}$ is positive semidefinite, Equation (102)

implies that

$$\mathbf{W}_{m'}^T \mathbf{W}_{m'} \succeq \mathbf{W}_{m',m'}^T \mathbf{W}_{m',m'} \tag{105}$$

and therefore

$$\lambda_{\min}(\mathbf{W}_{m'}^T \mathbf{W}_{m'}) \geq \lambda_{\min}(\mathbf{W}_{m',m'}^T \mathbf{W}_{m',m'}), \tag{106}$$

so the interval given in Equation (104) is nonnegative.

Equation (103) suggests that if we maximize the sum of the eigenvalues of $\mathbf{W}_{m',m'}^T \mathbf{W}_{m',m'}$, then the sum of the eigenvalues of $\sum_{m \neq m'} \mathbf{W}_{m,m'}^T \mathbf{W}_{m,m'}$ will be minimized, reducing the average singular value of $\tilde{\mathbf{W}}_{m'}$. Equation (104) further shows that if the minimum eigenvalue of $\mathbf{W}_{m',m'}^T \mathbf{W}_{m',m'}$ is large, the maximum eigenvalue of $\sum_{m \neq m'} \mathbf{W}_{m,m'}^T \mathbf{W}_{m,m'}$ will be small, reducing the maximum singular value of $\tilde{\mathbf{W}}_{m'}$. Together, these imply that the singular values of $\tilde{\mathbf{W}}_{m'}$ will taper quickly, as desired, allowing it to be approximated by a lower rank r in Eq. (26).

These observations motivate the following approach: we select the submatrices $\mathbf{W}_{m',m'}$ to maximize $\sum_{m'} \mathrm{Tr}\left(\mathbf{W}_{m',m'}^T \mathbf{W}_{m',m'}\right)$. One way to approach this is to define

$$\mathbf{d}_{m'} := \mathrm{diag}\left(\mathbf{W}_{m'} \mathbf{W}_{m'}^T\right) \in \mathbb{R}^{d_1 \times 1}, \tag{107}$$

and to let $\mathbf{z}_{m'} \in \mathbb{R}^{d_1 \times 1}$ represent an indexing vector for which rows of $\mathbf{W}_{m'}$ correspond to the submatrix $\mathbf{W}_{m',m'}$. If $\mathbf{z}_{m'}$ has a '1' in each row corresponding to a row of $\mathbf{W}_{m',m'}$, and a '0' everywhere else, then we can write

$$\sum_{m'} \mathrm{Tr}\left(\mathbf{W}_{m',m'}^T \mathbf{W}_{m',m'}\right) = \sum_{m'} \mathbf{d}_{m'}^T \mathbf{z}_{m'}. \tag{108}$$

Therefore, we can re-express our optimization in the form

$$\begin{aligned} \underset{\{\mathbf{z}_{m'}\}_{m'=1,\ldots,M}}{\text{maximize}} \quad & \sum_{m'} \mathbf{d}_{m'}^T \mathbf{z}_{m'} \\ \text{subject to} \quad & \mathbf{z}_{m'} \in \{0,1\}^{d_1 \times 1}, \quad m' = 1, \ldots, M, \\ & \sum_{m'} \mathbf{z}_{m'} = \mathbf{1}, \\ & \mathbf{1}^T \mathbf{z}_{m'} = \frac{d_1}{M}, \quad m' = 1, \ldots, M, \end{aligned}$$

where $\mathbf{1}$ represents the $d_1 \times 1$ vector of all ones. The second condition, $\sum_{m'} \mathbf{z}_{m'} = \mathbf{1}$, suggests for each of the d_1 indices, only one $\mathbf{z}_{m'}$ should contain a '1'. The final condition, $\mathbf{1}^T \mathbf{z}_{m'} = \frac{d_1}{M}$, indicates that each $\mathbf{z}_{m'}$ should contain exactly d_1/M ones.

We can relax this optimization to the following linear program:

$$\begin{aligned} \underset{\{\mathbf{z}_{m'}\}_{m'=1,\ldots,M}}{\text{maximize}} \quad & \sum_{m'} \mathbf{d}_{m'}^T \mathbf{z}_{m'} \\ \text{subject to} \quad & \mathbf{0} \leq \mathbf{z}_{m'} \leq \mathbf{1}, \quad m' = 1, \ldots, M, \\ & \sum_{m'} \mathbf{z}_{m'} = \mathbf{1}, \\ & \mathbf{1}^T \mathbf{z}_{m'} = \frac{d_1}{M}, \quad m' = 1, \ldots, M. \end{aligned}$$

Here, $\mathbf{0}$ represents the $d_1 \times 1$ vector of all zeros, and the condition $\mathbf{0} \leq \mathbf{z}_{m'} \leq \mathbf{1}$ indicates elementwise inequalities for each index of the vectors $\mathbf{z}_{m'}$. This condition indicates that each entry of $\mathbf{z}_{m'}$ should be in the range $[0, 1]$. If this linear program converges to a solution where each $\mathbf{z}_{m'}$ is in $\{0, 1\}^{d_1 \times 1}$, the vectors of all '0' and '1' entries, then each $\mathbf{z}_{m'}$ will correspond to a valid indexing-vector which selects the rows of $\mathbf{W}_{m'}$ corresponding to $\mathbf{W}_{m',m'}$.

Alternatively, we may select the $\mathbf{z}_{m'}$ in a greedy fashion: We first initialize each $\mathbf{z}_{m'}$ to $\mathbf{0}$. Then for each index $i = 1, ..., d_1$, we find the m' for which $\mathbf{d}_{m'}$ has the largest entry in the i^{th} position, and set the i^{th} index of the corresponding $\mathbf{z}_{m'}$ equal to 1. If at any point $\mathbf{z}_{m'}$ has $\frac{d_1}{M}$ entries equal to 1, we remove $\mathbf{z}_{m'}$ from consideration for additional '1' entries (which we can do by replacing the entries of $\mathbf{d}_{m'}$ with $-\infty$, for instance), and continue this procedure with the remaining $\mathbf{d}_m$ and $\mathbf{z}_m$ vectors. This method is formally described in Algorithm 1.

Algorithm 1 Selecting the Matrices $\mathbf{W}_{m',m'}$

Require: Matrices $\mathbf{W}_{m'} \in \mathbb{R}^{d_1 \times \frac{d_0}{M}}$, $m' = 1, ..., M$

Ensure: Partition the indices $\{1, ..., d_1\}$ into equal-sized disjoint subsets $\mathcal{I}_1, ..., \mathcal{I}_M$ so that $\mathbf{W}_{m',m'} = \Big[[\mathbf{W}_{m'}]_{i,j}\Big]_{i \in \mathcal{I}_{m'},\ j \in \{1,...,d_0\}}$.

Initialize $\mathcal{I}_{m'} = \emptyset$, $m' = 1, \ldots, M$, and $\mathcal{D} = \Big[\mathbf{d}_1 \quad \ldots \quad \mathbf{d}_M\Big] \in \mathbb{R}^{d_1 \times M}$.
for $i_{\text{output}} = 1, \ldots, d_1$ **do**
  $(\hat{i}, \hat{m}) \leftarrow \arg\max\{\mathcal{D}_{i,m}\}$
  $\mathcal{I}_{\hat{m}} \leftarrow \mathcal{I}_{\hat{m}} \cup \{\hat{i}\}$
  $\mathcal{D}_{\hat{i},m} \leftarrow -\infty$ for $m = 1, \ldots, M$.
  if $|\mathcal{I}_{\hat{m}}| = d_1/M$ **then**
    $\mathcal{D}_{i,\hat{m}} \leftarrow -\infty$ for $i = 1, \ldots, d_1$.
  end if
end for

G. Error Reduction Compared to Baseline SVD Method

Let us look more closely at how the approximation error of our method compares to that of taking a rank-r SVD approximation of each of the submatrices $\mathbf{W}_{m'}$. In the SVD case, the approximation error will be determined by how quickly the eigenvalues of $\mathbf{W}_{m'}^T \mathbf{W}_{m'}$ taper, whereas in our proposed method, it is determined by how quickly the eigenvalues of $\sum_{m \neq m'} \mathbf{W}_{m,m'}^T \mathbf{W}_{m,m'}$ taper. For convenience, we will write

$$\mathbf{M}_0^{(m')} := \mathbf{W}_{m'}^T \mathbf{W}_{m'}, \tag{109}$$

$$\mathbf{M}_1^{(m')} := \sum_{m \neq m'} \mathbf{W}_{m,m'}^T \mathbf{W}_{m,m'}. \tag{110}$$

From Eq. (103) and Eq. (104), we have

$$\text{Tr}\left(\mathbf{M}_0^{(m')}\right) \geq \text{Tr}\left(\mathbf{M}_1^{(m')}\right), \tag{111}$$

$$\lambda_{\max}(\mathbf{M}_0^{(m')}) \geq \lambda_{\max}(\mathbf{M}_1^{(m')}). \tag{112}$$

In other words, both the sum and the maximum of the eigenvalues of $\mathbf{M}_1^{(m')}$ will be less than the corresponding sum and maximum of those of $\mathbf{M}_0^{(m')}$, and therefore our method likely achieves lower approximation error.

H. Experimental Results

Ultimately, we are concerned with the degradation of network performance resulting from using our proposed parameter reduction compared to a more classical one, like the SVD. To test this, we trained a small feed-forward network with a single hidden fully connected layer to classify images from the CIFAR-10 dataset. This consists of images from ten mutually exclusive categories (e.g., airplanes, automobiles, and birds). Each image is 32×32 pixels, with each pixel containing data values for three color channel. Our network vectorizes these values into a length-3072 vector. Our hidden layer had 3072 nodes, corresponding to a 3072×3072 weight matrix $\mathbf{W}$. We trained our network to minimize the categorical cross entropy loss function, and it attained a maximum accuracy of about 46% after about 15 epochs.

We then simulated a situation in which we partitioned the hidden layer amongst $M = 32$ machines, each of which had access to a fixed fraction of the data values (thereby partitioning $\mathbf{W}$ into $\mathbf{W}_1$, $\mathbf{W}_2$, ..., $\mathbf{W}_{32}$, each with 96 columns). In Figure 5, we examine the eigenvalues of the matrices $\mathbf{M}_0^{(m')}$ and $\mathbf{M}_1^{(m')}$ from Equations (109) and (110), for all indices $m' = 1, ..., 32$. We see that, as expected, $\text{Tr}\left(\mathbf{M}_1^{(m')}\right)$ shows a significant reduction over $\text{Tr}\left(\mathbf{M}_0^{(m')}\right)$, indicating that a fixed rank-r approximation will likely have lower error for our method than for the SVD approximation approach.

We then performed parameter reduction on $\mathbf{W}$ using both our proposed method (selecting the submatrices $\mathbf{W}_{m',m'}$ via Algorithm 1 and varying the rank r of each $\tilde{\mathbf{W}}_{m'}^{(r)}$) as well the traditional low-rank SVD approximation (taking individual SVDs of each of the $\mathbf{W}_{m'}$). In Figure 6, we examine the resulting relative error in the approximation of $\mathbf{W}\mathbf{x}^{(0)}$. In Figure 6a, we estimate the upper bound from Eq. (94), computing $\sigma_{\max}$ for each value of r and substituting the empirical mean value of $\frac{\|\mathbf{x}^{(0)}\|_2}{\|\mathbf{W}\mathbf{x}^{(0)}\|_2}$ over all entries of the CIFAR-10 test set. To get a comparable bound for the SVD approximation approach, we substitute $\sigma_{\max}$ with the maximum of the $(r+1)^{st}$ singular values of the matrices $\mathbf{W}_{m'}$.

We plot the actual empirical mean relative error in Figure 6b. As we can see, the upper bound correctly predicts a drop in relative error for our method over the SVD approximation approach, though in this case the bound is very loose. We see that the improvement in relative error shrinks as we increase the approximation rank r.

As we altered the number of parameters in each method, we also tracked the prediction accuracy of the network, which we plot in Figure 7 as a function of r. Our method has a slightly better accuracy than the SVD approach for low ranks r, corresponding to the smaller relative error, and both approaches converge to the maximum accuracy achieved by the full matrix $\mathbf{W}$ when r reaches about 10% of its maximum value of 96.

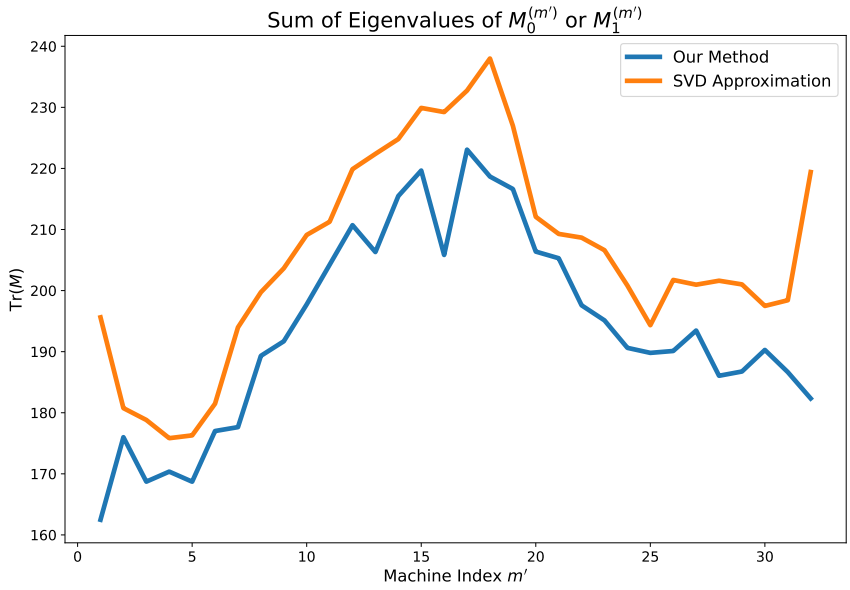


Figure 5. Reduction in the sum of the eigenvalues of $\mathbf{M}_1 := \sum_{m \neq m'} \mathbf{W}_{m,m'}^T \mathbf{W}_{m,m'}$ compared to those of $\mathbf{M}_0 := \mathbf{W}_{m'}^T \mathbf{W}_{m'}$, for each machine index m' in our CIFAR-10 experiment. This indicates a drop in the expected error of a low-rank approximation of the corresponding weight matrix $\mathbf{W}_{m'}$.

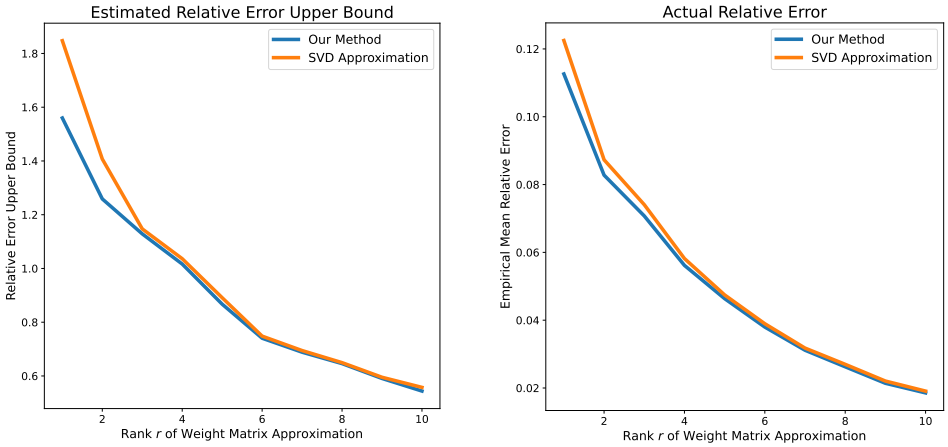
III. Partitioning Attention Mechanisms

We now consider the attention mechanism of a transformer model in which each token is mapped to a set of three vectors: a query vector $\mathbf{q} \in \mathbb{R}^{d_k \times 1}$, a key vector $\mathbf{k} \in \mathbb{R}^{d_k \times 1}$, and a value vector $\mathbf{v} \in \mathbb{R}^{d_v \times 1}$. For a set of N_T tokens, these vectors are concatenated into matrices

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}_1^T \\ \vdots \\ \mathbf{q}_{N_T}^T \end{bmatrix} \in \mathbb{R}^{N_T \times d_k}, \quad \mathbf{K} = \begin{bmatrix} \mathbf{k}_1^T \\ \vdots \\ \mathbf{k}_{N_T}^T \end{bmatrix} \in \mathbb{R}^{N_T \times d_k}, \quad \mathbf{V} = \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_{N_T}^T \end{bmatrix} \in \mathbb{R}^{N_T \times d_v}. \tag{113}$$

We will consider the dot-product attention, given by

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right) \cdot \mathbf{V} \in \mathbb{R}^{N_T \times d_v}. \tag{114}$$



(a) **(b)**

Figure 6. A single dense layer of a pre-trained CIFAR-10 classifier network partitioned over 32 machines. Fig. 6a shows the estimated mean relative error upper bound as a function of the number of parameters r broadcast by each machine. Fig. 6b shows the actual empirical mean relative error.

Explicitly, $\mathbf{Q}\mathbf{K}^T \in \mathbb{R}^{N_T \times N_T}$ is the matrix of dot products between all the query and key vectors, and the entries of the i^{th} row of softmax $\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)$ are given by

$$\left[\text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\right]_{i,j} = \frac{\exp(\mathbf{q}_i^T \mathbf{k}_j/\sqrt{d_k})}{\sum_{j'=1}^{N_T} \exp(\mathbf{q}_i^T \mathbf{k}_{j'}/\sqrt{d_k})}. \tag{115}$$

Each row of the attention matrix then represents a weighted sum of the value vectors for each token:

$$[\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V})]_{i,:} = \frac{1}{\sum_{j'=1}^{N_T} \exp(\mathbf{q}_i^T \mathbf{k}_{j'}/\sqrt{d_k})} \cdot \sum_{j=1}^{N_T} \exp\left(\frac{\mathbf{q}_i^T \mathbf{k}_j}{\sqrt{d_k}}\right) \cdot \mathbf{v}_j^T. \tag{116}$$

Toward partitioning the attention calculation over M machines, we consider the scenario where each party is responsible for $\frac{N_T}{M}$ of the tokens, and must compute the associated rows of the attention matrix. This situation may arise, for instance, when the machines are simultaneously imaging a common target from multiple angles and positions, and wish to implement a vision transformer model whereby the images are tokenized. In this case, the query, key, and value matrices are each implicitly partitioned into M submatrices:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_1 \\ \vdots \\ \mathbf{Q}_M \end{bmatrix}, \quad \mathbf{K} = \begin{bmatrix} \mathbf{K}_1 \\ \vdots \\ \mathbf{K}_M \end{bmatrix}, \quad \mathbf{V} = \begin{bmatrix} \mathbf{V}_1 \\ \vdots \\ \mathbf{V}_M \end{bmatrix}, \tag{117}$$

where $\mathbf{Q}_m \in \mathbb{R}^{\frac{N_T}{M} \times d_k}$, $\mathbf{K}_m \in \mathbb{R}^{\frac{N_T}{M} \times d_k}$, and $\mathbf{V}_m \in \mathbb{R}^{\frac{N_T}{M} \times d_v}$ are the submatrices possessed by machine $m = 1, ..., M$. In order for machine m to compute its portion of

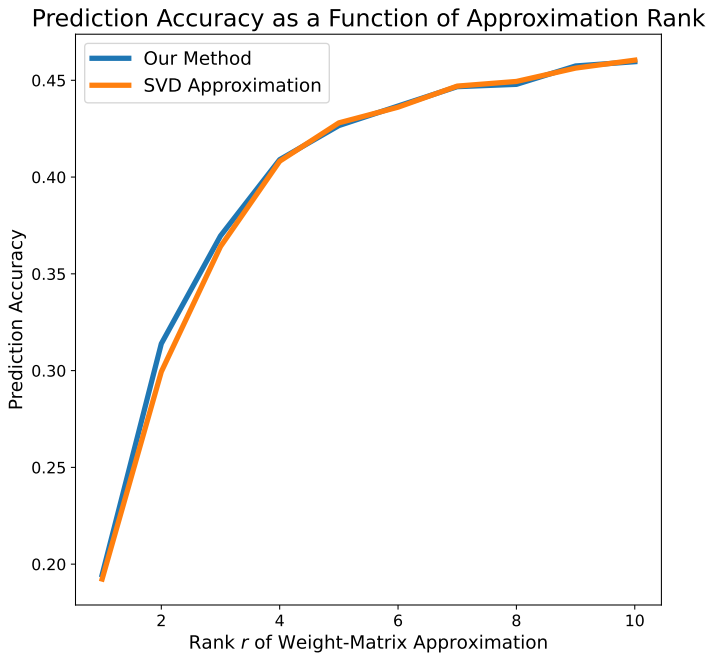


Figure 7. A single dense layer of a pre-trained CIFAR-10 classifier network partitioned over 32 machines. Here we show the network accuracy as a function of the number of parameters r broadcast by each machine.

the attention matrix, it suffices to calculate

$$\mathbf{A}_m := \exp\left(\frac{\mathbf{Q}_m \mathbf{K}_m^T}{\sqrt{d_k}}\right) \cdot \mathbf{V}_m + \sum_{m' \neq m} \exp\left(\frac{\mathbf{Q}_m \mathbf{K}_{m'}^T}{\sqrt{d_k}}\right) \cdot \mathbf{V}_{m'}, \tag{118}$$

$$\mathbf{s}_m := \left(\exp\left(\frac{\mathbf{Q}_m \mathbf{K}_m^T}{\sqrt{d_k}}\right) + \sum_{m' \neq m} \exp\left(\frac{\mathbf{Q}_m \mathbf{K}_{m'}^T}{\sqrt{d_k}}\right)\right) \cdot \mathbf{1}_{N_T \times 1}, \tag{119}$$

and produce the product

$$\mathrm{diag}(\mathbf{s}_m)^{-1} \cdot \mathbf{A}_m. \tag{120}$$

Thus, machine m requires information about $\mathbf{K}_{m'}$ and $\mathbf{V}_{m'}$ for all $m' \neq m$.

A. Compressing the Key Data to Preserve Query-Key Inner Products

We will focus on how effectively the machines can estimate their share of the scaled dot-product matrix $\mathbf{Q}\mathbf{K}^T$ with limited intercommunication. To this end, we focus on a scenario in which machine m' communicates a small amount of information about a key $\mathbf{k}$ to machine m, which holds query $\mathbf{q}$. We will call the communicated message $f(\mathbf{k})$. Machine m will use this information to construct an estimate $g(\mathbf{q}, f(\mathbf{k}))$ for the

inner product $\mathbf{q}^T\mathbf{k}$. The machines have the goal of constructing a scheme to minimize the expected error in the inner product,

$$\text{minimize}_{f,g}\ \mathbb{E}\left(\left|\mathbf{q}^T\mathbf{k} - g(\mathbf{q}, f(\mathbf{k}))\right|\right). \tag{121}$$

If we have an isometry $\mathbf{U} \in \mathbb{R}^{d_k \times d_k}$, satisfying $\mathbf{U}^T\mathbf{U} = \mathbf{U}\mathbf{U}^T = \mathbf{I}_{d_k \times d_k}$, then we can decompose our dot product as

$$\mathbf{q}^T\mathbf{k} = \mathbf{q}^T\mathbf{U}\mathbf{U}^T\mathbf{k} = \sum_{i=1}^{d_k} \mathbf{q}^T\mathbf{u}_i\mathbf{u}_i^T\mathbf{k}, \tag{122}$$

where $\mathbf{u}_i \in \mathbb{R}^{d_k \times 1}$ is the i^{th} column of $\mathbf{U}$. The $\{\mathbf{u}_i\}$ form an orthonormal basis for $\mathbb{R}^{d_k \times 1}$. Suppose machine m' communicates the first K coefficients $\mathbf{u}_i^T\mathbf{k}$, and machine m estimates the inner product $\mathbf{q}^T\mathbf{k}$ by adding the sum of the corresponding terms $\mathbf{q}^T\mathbf{u}_i\mathbf{u}_i^T\mathbf{k}$ to the expected value of the remaining terms:

$$f_{\mathbf{U}}(\mathbf{k}) := (\mathbf{u}_i^T\mathbf{k})_{i=1,\ldots,K}, \tag{123}$$

$$g_{\mathbf{U}}(\mathbf{q}, f_{\mathbf{U}}(\mathbf{k})) := \sum_{i=1}^{K} \mathbf{q}^T\mathbf{u}_i\mathbf{u}_i^T\mathbf{k} + \mathbb{E}\left(\sum_{i=K+1}^{d_k} \mathbf{q}^T\mathbf{u}_i\mathbf{u}_i^T\mathbf{k}\right). \tag{124}$$

Our expected error will be given by the uncertainty in $\sum_{i=K+1}^{d_k} \mathbf{q}^T\mathbf{u}_i\mathbf{u}_i^T\mathbf{k}$:

$$\mathbb{E}\left(\left|\mathbf{q}^T\mathbf{k} - g_{\mathbf{U}}(\mathbf{q}, f_{\mathbf{U}}(\mathbf{k}))\right|\right) = \mathbb{E}\left(\left|\sum_{i=K+1}^{d_k} \mathbf{q}^T\mathbf{u}_i\mathbf{u}_i^T\mathbf{k} - \mathbb{E}\left(\sum_{i=K+1}^{d_k} \mathbf{q}^T\mathbf{u}_i\mathbf{u}_i^T\mathbf{k}\right)\right|\right) \tag{125}$$

$$= \mathbb{E}\left(\left|\sum_{i=K+1}^{d_k} \left(\mathbf{q}^T\mathbf{u}_i\mathbf{u}_i^T\mathbf{k} - \mathbb{E}\left(\mathbf{q}^T\mathbf{u}_i\mathbf{u}_i^T\mathbf{k}\right)\right)\right|\right) \tag{126}$$

$$= \mathbb{E}\left(\left|\sum_{i=K+1}^{d_k} \mathbf{u}_i^T\left(\mathbf{k}\mathbf{q}^T - \mathbb{E}\left(\mathbf{k}\mathbf{q}^T\right)\right)\mathbf{u}_i\right|\right), \tag{127}$$

which is upper bounded using the triangle inequality as

$$\mathbb{E}\left(\left|\mathbf{q}^T\mathbf{k} - g_{\mathbf{U}}(\mathbf{q}, f_{\mathbf{U}}(\mathbf{k}))\right|\right) \leq \mathbb{E}\left(\sum_{i=K+1}^{d_k} \left|\mathbf{u}_i^T\left(\mathbf{k}\mathbf{q}^T - \mathbb{E}\left(\mathbf{k}\mathbf{q}^T\right)\right)\mathbf{u}_i\right|\right). \tag{128}$$

From the Cauchy-Schwartz inequality, we further have

$$\left|\mathbf{u}_i^T\left(\mathbf{k}\mathbf{q}^T - \mathbb{E}\left(\mathbf{k}\mathbf{q}^T\right)\right)\mathbf{u}_i\right|^2 \leq \|\mathbf{u}_i\|_2^2 \cdot \|\left(\mathbf{k}\mathbf{q}^T - \mathbb{E}\left(\mathbf{k}\mathbf{q}^T\right)\right)\mathbf{u}_i\|_2^2 \tag{129}$$

$$= \|\mathbf{u}_i\|_2^2 \cdot \mathbf{u}_i^T\left(\mathbf{q}\mathbf{k}^T - \mathbb{E}\left(\mathbf{q}\mathbf{k}^T\right)\right)\left(\mathbf{k}\mathbf{q}^T - \mathbb{E}\left(\mathbf{k}\mathbf{q}^T\right)\right)\mathbf{u}_i \tag{130}$$

$$= \mathbf{u}_i^T\left(\mathbf{q}\mathbf{k}^T - \mathbb{E}\left(\mathbf{q}\mathbf{k}^T\right)\right)\left(\mathbf{k}\mathbf{q}^T - \mathbb{E}\left(\mathbf{k}\mathbf{q}^T\right)\right)\mathbf{u}_i. \tag{131}$$

Combining Eq. (131) with (128) gives us

$$\mathbb{E}\left(\left|\mathbf{q}^T\mathbf{k} - g_{\mathbf{U}}(\mathbf{q}, f_{\mathbf{U}}(\mathbf{k}))\right|\right) \leq \sum_{i=K+1}^{d_k} \mathbb{E}\left(\sqrt{\mathbf{u}_i^T\left(\mathbf{q}\mathbf{k}^T - \mathbb{E}\left(\mathbf{q}\mathbf{k}^T\right)\right)\left(\mathbf{k}\mathbf{q}^T - \mathbb{E}\left(\mathbf{k}\mathbf{q}^T\right)\right)\mathbf{u}_i}\right) \tag{132}$$

$$\leq \sum_{i=K+1}^{d_k} \sqrt{\mathbb{E}\left(\mathbf{u}_i^T\left(\mathbf{q}\mathbf{k}^T - \mathbb{E}\left(\mathbf{q}\mathbf{k}^T\right)\right)\left(\mathbf{k}\mathbf{q}^T - \mathbb{E}\left(\mathbf{k}\mathbf{q}^T\right)\right)\mathbf{u}_i\right)} \tag{133}$$

$$= \sum_{i=K+1}^{d_k} \sqrt{\mathbf{u}_i^T\mathbb{E}\left[\left(\mathbf{q}\mathbf{k}^T - \mathbb{E}\left(\mathbf{q}\mathbf{k}^T\right)\right)\left(\mathbf{k}\mathbf{q}^T - \mathbb{E}\left(\mathbf{k}\mathbf{q}^T\right)\right)\right]\mathbf{u}_i}, \tag{134}$$

where Eq. (133) follows from Jensen's inequality, since the square root function is concave. For orthonormal vectors $\{\mathbf{u}_i\}$, this bound is minimal when the vectors $\{\mathbf{u}_i\}_{i=K+1,\ldots,d_k}$ are aligned with the $d_k - K$ singular vectors corresponding to the smallest singular values of the matrix

$$\mathbf{C}_{\mathbf{q},\mathbf{k}} := \mathbb{E}\left[\left(\mathbf{q}\mathbf{k}^T - \mathbb{E}\left(\mathbf{q}\mathbf{k}^T\right)\right)\left(\mathbf{k}\mathbf{q}^T - \mathbb{E}\left(\mathbf{k}\mathbf{q}^T\right)\right)\right]. \tag{135}$$

This suggests choosing the $\{\mathbf{u}_i\}_{i=1,\ldots,K}$ to be aligned with the first K singular vectors of $\mathbf{C}_{\mathbf{q},\mathbf{k}}$.

We may compare this approach with a more traditional one motivated by a principal component analysis (PCA), wherein we choose the $\mathbf{u}_i$ to be aligned with the first K singular vectors of the covariance matrix

$$\mathbf{C}_{\mathbf{k}} := \mathbb{E}\left[\left(\mathbf{k} - \mathbb{E}\left(\mathbf{k}\right)\right)\left(\mathbf{k}^T - \mathbb{E}\left(\mathbf{k}^T\right)\right)\right]. \tag{136}$$

While the parameters $\mathbf{u}_i^T\mathbf{k}$ will have lower variance in this case, we find that the relative error in estimating $\mathbf{u}^T\mathbf{k}$ does not converge as quickly with respect to the number of parameters used. We performed a test using a small pre-trained BERT network [11] from Google's BERT Miniatures collection from TensorFlow Hub. We selected a model with two transformer block layers, a hidden size of 768, and 12 self-attention heads. The model is available at

`https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-768_A-12/1`,

with a corresponding preprocessor model at

`https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3`.

We randomly sampled token embeddings and associated query, key, and value vectors to created empirical estimates of $\mathbf{C}_{\mathbf{q},\mathbf{k}}$, $\mathbf{C}_{\mathbf{k}}$, and the corresponding inner products $\mathbf{q}^T\mathbf{k}$ and their approximations. For each choice of the vectors $\{\mathbf{u}_i\}_{i=1}^K$, we evaluated the empirical mean relative error, $\frac{\left|\mathbf{q}^T\mathbf{k} - g_{\mathbf{U}}(\mathbf{q}, f_{\mathbf{U}}(\mathbf{k}))\right|}{|\mathbf{q}^T\mathbf{k}|}$, which we plot in Figure 8. As we can see, selecting the $\mathbf{u}_i$ to be aligned with the singular vectors of $\mathbf{C}_{\mathbf{q},\mathbf{k}}$ more efficiently reduces the relative error of the $\mathbf{q}^T\mathbf{k}$ estimates than the traditional PCA-based approach using the singular vectors of $\mathbf{C}_{\mathbf{k}}$.

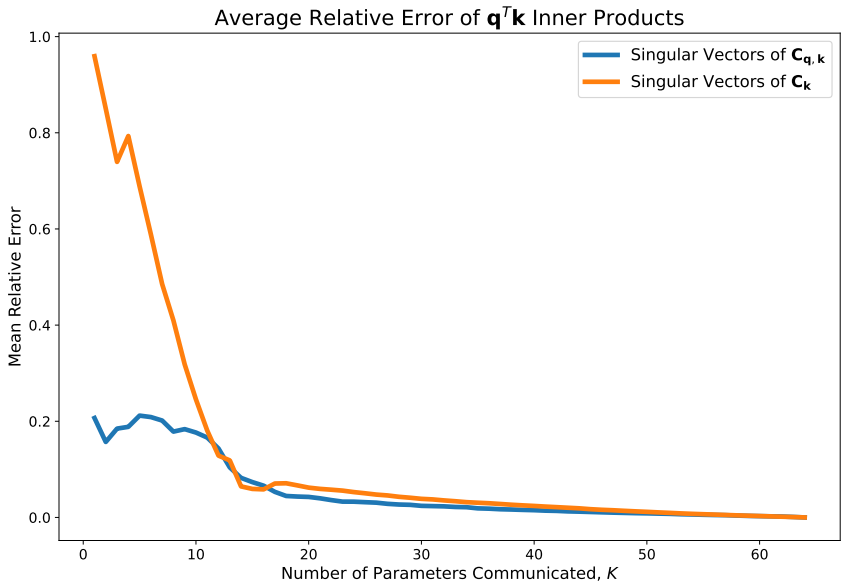


Figure 8. The mean relative error in the $\mathbf{q}^T\mathbf{k}$ inner products as a function of the number of parameters communicated. Parameters were chosen to be $\mathbf{u}_i^T\mathbf{k}$ for the dominant K singular vectors of either $\mathbf{C}_{\mathbf{q},\mathbf{k}}$ or $\mathbf{C}_{\mathbf{k}}$. Here, the $\mathbf{q}$ and $\mathbf{k}$ vectors were generated from randomly-sampled BERT model tokens.

IV. Conclusions

In this work, we have described a method to partition components of a large transformer-based AI model to ease the overwhelming size, memory, and power burdens that may impede its use in a space environment. While this study has addressed some of the main parts of the transformer network, there remains work to be done to completely formulate the partitioning of a large network that would be practical for space exploration (possibly a vision transformer using input camera imagery from multiple Mars terrestrial rovers and helicopters, for instance). Furthermore, while in this work we discussed the intercommunication overhead in terms of the number of parameters broadcast by each machine, we have yet to define the exact coding-for-communication scheme the machines would employ to guarantee a low-enough degradation in network performance. This would affect the speed with which the machines could orchestrate the network functionality, and would be a function of the positioning of the machines (affecting the capacity of the communication channels between them) as well as the power available for communication. All of these are interesting questions for future study. Still, this work represents an important step in adapting powerful state-of-the-art AI tools to space-based systems.

Acknowledgments

The authors are grateful to JPL's Blue Sky program, which funded this study. The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).

References

[1] DeepSeek-AI, A. Liu *et al.*, "Deepseek-v3 technical report," 2025. https://arxiv.org/abs/2412.19437

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, p. 6000–6010.

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020. https://arxiv.org/abs/2010.11929

[4] N. George and S. R. J, "Reduced complexity hyperspectral image compression using low rank factorization," in *2025 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*, 2025, pp. 43–48.

[5] W. Choe, Y. Ji, and F. X. Lin, "Rwkv-edge: Deeply compressed rwkv for resource-constrained devices," 2025. https://arxiv.org/abs/2412.10856

[6] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," 2020. https://arxiv.org/abs/2006.04768

[7] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *CoRR*, vol. abs/2106.09685, 2021. https://arxiv.org/abs/2106.09685

[8] Y.-C. Hsu, T. Hua, S. Chang, Q. Lou, Y. Shen, and H. Jin, "Language model compression with weighted low-rank factorization," 2022. https://arxiv.org/abs/2207.00112

[9] Y. Ji, H. Saratchandran, C. Gordon, Z. Zhang, and S. Lucey, "Efficient learning with sine-activated low-rank matrices," 2025. https://arxiv.org/abs/2403.19243

[10] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," 2019. https://arxiv.org/abs/1904.10509

[11] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. http://arxiv.org/abs/1810.04805