Activation Sparsity Opportunities for Compressing General Large Language Models

Nobel Dhar, Bobin Deng, Md Romyull Islam, Kazi Fahim Ahmad Nasif, Liang Zhao, Kun Suo
College of Computing and Software Engineering, Kennesaw State University
Email: ndhar@students.kennesaw.edu, bdeng2@kennesaw.edu, {mislam22, knasif}@students.kennesaw.edu,
{lzhao10, ksuo}@kennesaw.edu

***Abstract*—Deploying local AI models, such as Large Language Models (LLMs), to edge devices can substantially enhance devices' independent capabilities, alleviate the server's burden, and lower the response time. Owing to these tremendous potentials, many big tech companies have been actively promoting edge LLM evolution and released several lightweight Small Language Models (SLMs) to bridge this gap. However, SLMs currently only work well on limited real-world applications. We still have huge motivations to deploy more powerful (larger-scale) AI models on edge devices and enhance their smartness level. Unlike the conventional approaches for AI model compression, we investigate from activation sparsity. The activation sparsity method is orthogonal and combinable with existing techniques to maximize compression rate while maintaining great accuracy. According to statistics of open-source LLMs, their Feed-Forward Network (FFN) components typically comprise a large proportion of parameters (around $\frac{2}{3}$). This internal feature ensures that our FFN optimizations would have a better chance of achieving effective compression. Moreover, our findings are beneficial to general LLMs and are not restricted to ReLU-based models.**

This work systematically investigates the tradeoff between enforcing activation sparsity and perplexity (accuracy) on state-of-the-art LLMs. Our empirical analysis demonstrates that we can obtain around 50% of main memory and computing reductions for critical FFN components with negligible accuracy degradation. This extra 50% sparsity does not naturally exist in the current LLMs, which require tuning LLMs' activation outputs by injecting zero-enforcing thresholds. To obtain the benefits of activation sparsity, we provide a guideline for the system architect for LLM prediction and prefetching. Moreover, we further verified the predictability of activation patterns in recent LLMs. The success prediction allows the system to prefetch the necessary weights while omitting the inactive ones and their successors (compress models from the memory's perspective), therefore lowering cache/memory pollution and reducing LLM execution time on resource-constraint edge devices.

***Index Terms*—Large Language Models (LLMs), AI Compression, Activation Sparsity, Edge LLM**

I. Introduction

The rapid evolution of transformed-based AI models, such as large language models (LLMs) or Large Vision Models (LVM), has significantly accelerated the recent AI expansion. State-of-the-art models typically contain several hundreds of billions or even trillion parameters (e.g., *GPT-4* has *1.76* trillion parameters [1]) and deploy on HPC/cloud servers. If end users require services from these large-scale AI models, they must send requests from edge to the HPC via a network connection, and the HPC then allocates tens of high-end GPUs to execute the model inference and return results. This client-server architecture pushes all computing burdens to the centralized HPC, which not only requires stable network connections from the edge but also limits the system's scalability. Therefore, there is a growing interest in directly deploying LLMs on edge devices [2], [3] to eliminate the above challenges. The benefits of local edge LLMs include lowering the HPC's computing and memory burdens, reducing traffic, decreasing response time, and enhancing data privacy [4]. However, effectively deploying LLMs on edge is challenging because of their high computing and memory demands, often exceeding the capabilities of general edge devices. Big tech companies, such as *Google*, *Microsoft*, and *Meta*, proposed smaller versions of LLMs (aka. Small Language Models (SLMs)) to deliver lightweight local AI to edge. But the abilities of SLMs can only satisfy the criteria of limited applications, and we still have great motivation to offload more powerful AI models to the edge. Moreover, the supply-demand gap between edge devices and LLMs is growing exponentially. Therefore, we must explore new compression techniques to support LLM deployment on resource-constraint devices.

Pruning [5], quantization [6], and knowledge distillation [7] are three conventional methodologies to compress AI models. The compression rates theoretically may increase by utilizing one or more of these methods while maintaining appropriate accuracies. The compression rates of these approaches have theoretical up-bounds even in their ideal situations. To further attain better compression rates, we should explore a new approach that can seamlessly incorporate with existing schemes. This paper aims to investigate extra LLM compression possibilities via sparsity. AI model sparsity could be classified as *weight sparsity* and *activation sparsity*. Weight sparsity is more common in regular DNN models [8] but typically has much lower levels in state-of-the-art LLMs (Section III-A). Activation sparsity [9], [10] is associated with the output of FFN neurons, where a certain percentage of neurons will not be activated and no outputs pass to the subsequent layers. Therefore, from model compression's perspective, the weights of these inactive neurons are not required to be fetched into memory, which may enhance memory utilization, lower network traffic, and reduce execution latency.

The compression opportunity from activation sparsity is based on the observation that the majority of LLM's weights are from feedforward network (FFN) layers (around $\frac{2}{3}$ [11]), and not all FFN weights substantially impact the LLM output. We found that the natural activation sparsity (from default pre-trained models) of ReLU-based LLMs is normally much higher and can reach up to around *95%*. In contrast, state-of-the-art LLMs have switched to non-ReLU activations, such as *SwiGLU*, demonstrating negligible natural activation sparsity. Therefore, the recent evolution of LLMs' activation functions causes new challenges for compression via activation sparsity. Similar to weight sparsity, the impact of activation values is directly associated with their magnitudes. Our investigation revealed that a significant portion of recent LLMs' activation values are extremely close to zero, and approximately 50% of activation values in FFN layers can be safely withdrawn without significant accuracy loss. This feature allows us to predict activation patterns according to the user's input tokens. Or predict the deeper layers' patterns using the first few layers' patterns. The predicted inactive neurons, which have very small activation values, do not need to be fetched from the disk or SD card into memory. We can prefetch the active neurons to memory to lower the inference latency. Importantly, this approach does not require re-training or fine-tuning the LLMs, as the pattern prediction mechanism works on top of existing pre-trained LLMs. Our exploration also demonstrates that the activation patterns are very close (even 100% matched) in the same LLM with similar input tokens (e.g., 70% - 95% similarity). This highly coincident feature indicates a huge potential to predict activation patterns and, therefore, obtain an extra 50% LLM compression rate. Our finding can motivate system architects to design predictors and prefetch activated weights during the execution. This optimization should significantly reduce LLM execution latency on resource-constraint edge devices because of (1) less waiting time for fetching data from disk, (2) higher memory hit rate (inactivated neurons will not be loaded into memory to avoid cache/memory pollution), (3) less computing requirements (Only compute the activated neurons).

The main contributions of this paper are outlined as follows:

- We explore the weight and activation sparsity of state-of-the-art LLMs. For LLMs with limited natural activation sparsity, we also examine activation magnitude distributions and learn the importance levels of FFN neurons.
- Based on activation magnitude distributions, we enforced activation sparsity to the state-of-the-art LLMs. To the best of our knowledge, this is the first paper to explore the enforcing activation sparsity of state-of-the-art LLMs systematically. Our empirical analysis reveals that we can secure an extra 50% activation sparsity in FFN layers while maintaining acceptable accuracy.
- To convert the extra activation sparsity to compression benefits, we further investigate the predictability of activation patterns. According to our experiments, the activation patterns are highly predictable, and LLMs can be further compressed from memory's perspective. By following the activation pattern from the predictor, only the activated weights should be prefetch to memory while the inactivated ones remain in the lower-level disk. This optimization is orthogonal to existing compression techniques and may significantly reduce LLM execution time by lessening data transfer from the slow disk.

The remaining sections of this paper are organized as follows: Section II introduces our sparsity exploration methodologies. Section III investigates the LLMs' natural sparsity features. Section IV explores the tradeoffs between LLM enforcing activation sparsity and perplexity. Section V demonstrates the similarity and predictability of activation patterns. Related work is discussed in Section VI, and finally, we conclude in Section VII.

II. Sparsity Exploration Approaches

A. LLM Selection and Evaluation Environment

This work targets state-of-the-art LLMs and considers the diversity of internal activation functions. Table I summarizes the pre-trained LLMs we explore, which are collected from the *Hugging Face* repository. These models were carefully chosen for their broad representation of contemporary LLM architectures, ensuring that our findings are applicable across a diverse set of current models and are not limited by the choice of architectures. For input benchmarks, we utilize the *Wikitext-2* [12] dataset due to its widespread adoption in LLM research, allowing easier comparison with existing studies.

TABLE I: Selected Pre-trained LLMs

Model Name	Version	Activation Function	Number of Parameters
Llama	3	SwiGLU	8B
Mistral	0.1	SwiGLU	7B
Phi	2	NewGELU	2.7B
Phi	3-mini-128k	SwiGLU	3.8B
OPT	1	ReLU	6.7B

The LLM deployment, execution, and behavior analysis are performed on our local *Lambda* server, which is equipped with four *NVIDIA GeForce RTX 2080 Ti* GPUs. We also developed a custom PyTorch script to log activation values and enforce sparsity during LLM inferences. This allowed us to systematically track activation features and evaluate the impact of sparsity under diversified input conditions.

B. Targeting Components of LLMs for Compression

Popular LLMs today are based on the transformer architecture. Within this architecture, we target the MLP (Multi-Layer Perceptron) layers, a.k.a Feed-Forward Network(FFN) layers, for model compression. As highlighted in Figure 1, the components within FFN typically include *Gate Projection*, *Up Projection*, *Down Projection*, and *Activation_Function*. The order of FFN layers may be varied for different LLMs. The analysis and optimizations of the FFN layers are critical for model compression and enhance the inference performance because the FFN layers contribute about $\frac{2}{3}$ of total parameters on average [11]. Therefore, FFN layers are essentially the storage and computing bottleneck, and compressing/optimizing these components would be beneficial.

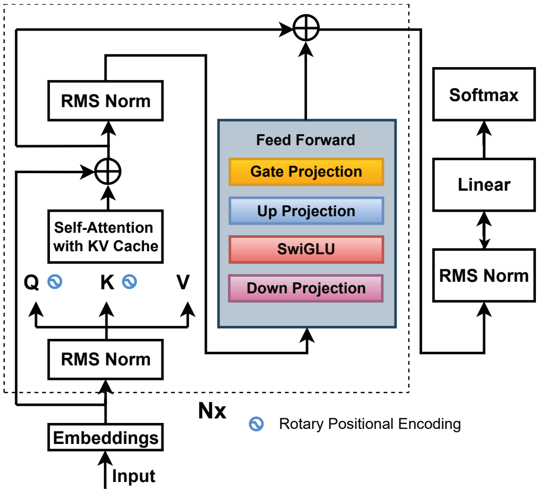


Fig. 1: Architecture of state-of-the-art decoder-only LLMs

C. Enforcing Sparsity in FFN Layers

Our primary motivation for exploring LLMs' activation sparsity is to minimize the neuron loading from the lower-level memory hierarchy (e.r., SD card or disk) and reduce computing requirements. The potential benefits of activation sparsity are similar to those from Sparse Matrix Multiplication. The state-of-the-art LLMs with non-ReLU activation functions provide very low or no activation sparsity. Because FFN layers store most of the LLM's parameters, enforcing sparsity in FFN layers can substantially reduce the number of active neurons, leading to less computing and memory usage. However, the critical premise of enforcing activation sparsity is that the LLM accuracy should not be obviously reduced. Moreover, as another critical component of transformer-based models, attention layers are typically susceptible to being modified because they aim to capture the dependencies of all elements in a sequence. Introducing sparsity in attention layers makes it more possible to degrade model accuracy [13]. Therefore, we focus on enforcing sparsity on FFN layers to alleviate the storage and computing bottleneck.

D. Auxiliary Program for Sparsity Analysis

We developed an auxiliary program for the LLM sparsity analysis, which includes the following main functionalities: (1) *activation behavior collection*, (2) *Important Neuron Determination*, and (3) *evaluations with specific thresholds*.

1) Activation Behavior Collection:

We built an activation collection model consisting of the following four subcomponents.

Input Sequence: Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ represent the LLM input tokens where N is the number of tokens.

Forward Pass Inference: For each input token $\mathbf{x}_i$, perform a forward pass through the model to obtain the activation values $\mathbf{A}_{i,l}$ for each layer l:

$$\mathbf{A}_{i,l} = f_l(\mathbf{x}_i) \tag{1}$$

where f_l is the activation function of layer l and $i \in [1, N]$.

Register Hooks: For each layer l, register hooks to capture the activation values during the inference. We define $\mathbf{A}_l$ as the activations for layer l across all input tokens:

$$\mathbf{A}_l = \{\mathbf{A}_{i,l} \mid i \in \{1, 2, \ldots, N\}\} \tag{2}$$

Activation Storage: Store the collected activation values $\mathbf{A}_l$ in an HDF5 file for next-step processing.

2) Important Neuron Determination:

Following the *Activation Behavior Collection* step discussed in Section II-D1, we must define the approach to select a specific percentage of neurons with smaller output magnitudes that are less important. The magnitude is directly associated with the importance of the neuron. Therefore, enforcing sparsity to the smaller magnitude neurons should have minimum difference compared to original output. The *Important Neuron Determination* contains the following substeps.

Flatten Activation Matrix to Single-Dimension Vector: For each layer l, flatten the activation matrix values $\mathbf{A}_l$ into a single dimension vector $\mathbf{A}_l^{\text{flat}}$:

$$\mathbf{A}_l^{\text{flat}} = \text{flatten}(\mathbf{A}_l) \tag{3}$$

Converting to Absolute Values: Compute the absolute values of the flattened activation vector. The importance of activation values are directly associated with magnitudes instead of signs:

$$\mathbf{A}_l^{\text{abs}} = \left|\mathbf{A}_l^{\text{flat}}\right| \tag{4}$$

Sorting: Sort the vector with absolute values in ascending order to facilitate the following percentile computation:

$$\mathbf{A}_l^{\text{sorted}} = \text{sort}(\mathbf{A}_l^{\text{abs}}) \tag{5}$$

Percentile Calculation: For a given sparsity level α (e.g., 30%), calculate the threshold $T_{l,\alpha}$ as the lowest α-th percentile of $\mathbf{A}_l^{\text{abs}}$. P_α denotes as the percentile function at sparsity α.

$$T_{l,\alpha} = P_\alpha(\mathbf{A}_l^{\text{sorted}}) \tag{6}$$

3) Evaluations with Specific Sparsity Thresholds:

After applying a pre-defined activation threshold, we obtain a new LLM with a different sparsity. Specifically, we implemented a mechanism to deactivate neurons in *gate projection*, *up projection*, and *down projection* components whose values fell below the threshold. Activation values are set to zero if they are less than $T_{l,\alpha}$ in magnitude:

$$\mathbf{A}_{i,l}^{\text{thresholded}} = \begin{cases} 0 & \text{if } |\mathbf{A}_{i,l}| < T_{l,\alpha} \\ \mathbf{A}_{i,l} & \text{otherwise} \end{cases} \tag{7}$$

III. Sparsity Findings of State-of-the-art LLMs

A. Weight Magnitude Distributions

Before systematically studying the LLMs' sparsity, exploring their weight magnitude distributions is a prerequisite to understanding internal features. Figure 2 illustrates the weight distributions of four recent LLMs: *Mistral-7B*, *LLama-3-8B*, *OPT-6.7B*, and *Phi-2-2.7B* The x-axis represents sequential

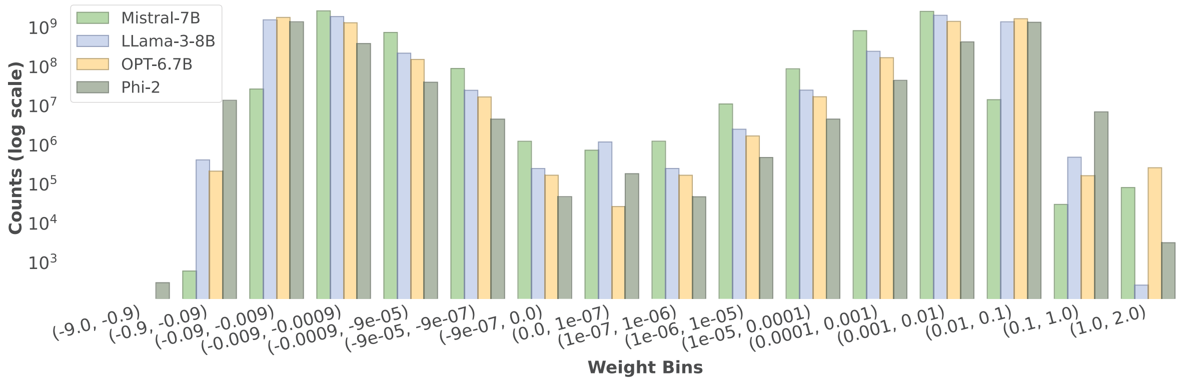


Fig. 2: Weight magnitude distributions of state-of-the-art LLMs. The notation *(a,b)* in the x-axis indicates a specific value range; E.g., *(-9.0,-0.9)* indicates the total number of weights whose values are larger than *-9.0* and smaller than *-0.9*.

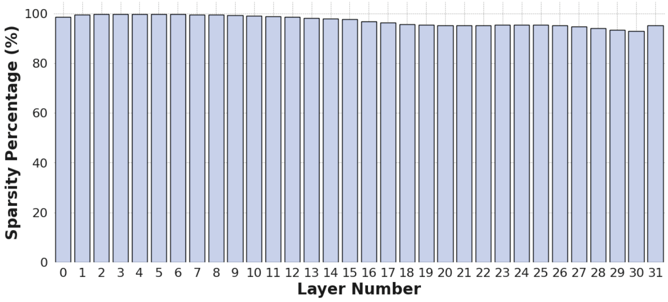


Fig. 3: Natural activation sparsity of all layers in *OPT-6.7B*

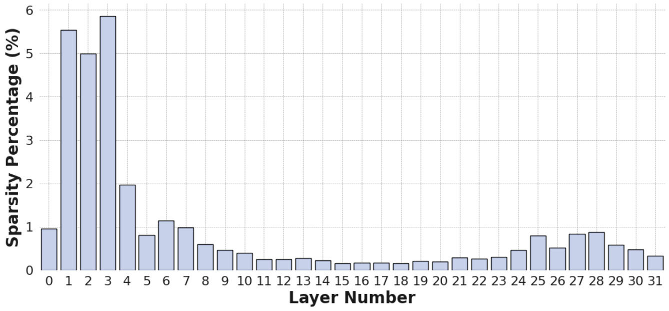


Fig. 4: Natural activation sparsity of all layers in *Phi-2-2.7B*

weight bins; each bin shows the weight count in a specific range. The bin ranges are denoted in value pairs, indicating the start and end of each range. Bin ranges in Figure 2 span from *(-9.0, -0.9)* to *(1.0, 2.0)*, all weights of our evaluated models fall into these ranges. The y-axis displays the weight counts on a logarithmic scale.

Observation: Upon analyzing the weight magnitude distributions of LLMs in Figure 2, we can obtain an essential guideline for LLM pruning or enforcing activation sparsity. The distribution of weights across different bins reveals that a large proportion of the weights fall within the small magnitude ranges, and a very small number of weights are absolute zeros. This trend is observed in all four evaluated LLMs, including *Mistral-7B*, *LLama-3-8B*, *OPT-6.7B*, and *Phi-2-2.7B*. In other words, the state-of-the-art LLMs do not directly provide significant weight sparsity, even though the magnitudes of a considerable percentage of weights are very close to zero.

Insight: Unlike some conventional DNNs [8], we do not obverse substantial weight sparsity in four recent LLMs. Therefore, to compress LLM via the sparsity feature, we must explore another sparsity type: activation sparsity.

B. Natural Activation Sparsity Analysis

Activation sparsity could be another essential feature and opportunity to compress the LLMs, especially when deploying these large-scale AI models in resource-constrained environments. The sparsity feature will not only provide opportunities to reduce memory storage but also lower the computing resource requirements. This improvement eventually optimizes the edge LLM execution latency, which provides quick input responses and better user experiences. We show the results of *OPT-6.7B* and *Phi-2-2.7B* in this section, as they are the only two models that exhibit natural sparsity in their Feed-Forward Network (FFN) activation values in our study. We do not observe any natural sparsity in other LLMs (*Mistral-7B*, *LLama-3-8B* and *Phi-3-3.8B*) we evaluated based on WikiText. The term '*natural activation sparsity*' indicates the activation sparsity directly comes from the pre-trained LLM instead of from enforcing sparsity.

Observation 1 (*OPT-6.7B*): The activation sparsity histogram of FFN layers for the *OPT-6.7B* is demonstrated in Figure 3. The sparsity percentages from *Layer 0* to *Layer 31* are all high, even though a slight sparsity degradation is observed in deeper layers. *Layer 30* has *92.83%* natural sparsity, which is the minimal sparsity among all layers. According to Table I, the activation function of *OPT-6.7B* is ReLU. The ReLU activation function's outputs are set to zero if the neuron inputs are less than zero. This attribute of the ReLU function is the main reason for obtaining high natural sparsity. Apple's recent work [10] for LLM compression is based on a similar observation and targeting to models with the ReLU activation.

Observation 2 (*Phi-2-2.7B*): Figure 4 illustrates the activation sparsity of FFN layers for the *Phi-2-2.7B* model. Even though a certain degree of sparsity exists, the maximum sparsity level (*Layer 3*) is still less than 6%. The sparsity percentages of the majority of layers are less than 1%. Compared to the *OPT-6.7B*, *Phi-2-2.7B* shows much lower activation sparsity. The main reason for this discrepancy is their internal activation functions, where *OPT-6.7B* utilizes ReLU while *Phi-2-2.7B* uses NewGELU. By comparing their equation curves, ReLU has a significantly higher possibility of obtaining a '0' activation output.

Insight: The benefits of natural activation sparsity only exist in ReLU-based LLMs (e.g., *OPT-6.7B*). Even though we can also observe natural activation sparsity in NewGELU-based LLMs (e.g., *OPT-6.7B*), the sparsity level is too low to make effective model compression. The state-of-the-art LLMs mainly use SwiGLU function. Therefore, we should explore other new and general methods to obtain extra activation sparsity for LLM compression.

C. Activation Magnitude Distributions

From the discussion in Section III-B, most state-of-the-art LLMs do not exhibit inherent natural activation sparsity due to the old function ReLU or NewGELU has been replaced. Significant portions of activation values in recent LLMs do not reach absolute zero. Therefore, this section examines activation magnitude distributions for other potential sparsity opportunities. For example, according to the results of activation magnitude distributions, can we adjust the sparsity level by modifying a certain percentage of small magnitude values to zero and increasing the activation sparsity (Section IV)? Exploring activation magnitude distributions is a crucial prerequisite step to answer this question. So, the distributions of (*Phi-3-3.8B*, *LLaMA-3-8B* and *Mistral-7B*) are shown and analyzed below. These three recent LLMs are non-ReLU-based and do not exhibit natural sparsity.

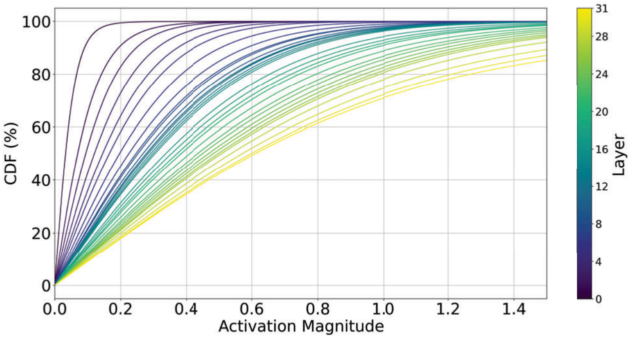


Fig. 5: Activation magnitude distribution of Phi-3-3.8B for all 32 layers (Gate-Up Projection)

Observation 1 (*Phi-3-3.8B*): Figure 5 displays the activation cumulative distribution function (CDF) of the *Gate-Up Projection* activation components for the *Phi-3-3.8B* across all its 32 layers. A significant concentration of activation values falls below *0.05*, particularly in the first layer (*Layer 0*), where approximately 60% of activation values are under *0.05*. In the subsequent three layers (*Layer 1 - 3*), about 70% of activation values range from 0.0 to 0.2, indicating a slight dispersion. As moving beyond these first several layers, activation values are more evenly spread over relatively wider ranges. The CDF further suggests that setting a cutoff threshold at *0.6* for *Phi-3-3.8B* could result in around 50% activation reduction, providing the potential for extra activation sparsity.

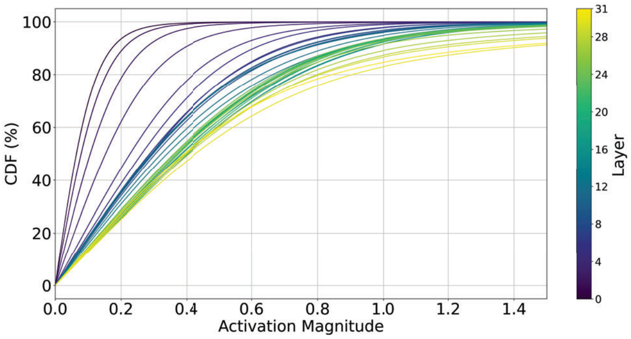


Fig. 6: Activation magnitude distribution of LLaMA-3-8B for all 32 layers (Gate Projection)

Observation 2 (*LLaMA-3-8B*): Figure 6 presents the activation CDF of *Gate Projection* component of *LLaMA-3-8B* in all its 32 layers. For *Layer 0*, 65% of activation values fall below *0.1*. For the subsequent three layers (*Layer 1 - 3*), more than 60% of activation values are below *0.2*. When examining the remainder of layers, the graph suggests that setting a threshold at *0.5* would allow us to discard 50% of neurons in FFN, implying a potential for 50% activation sparsity.

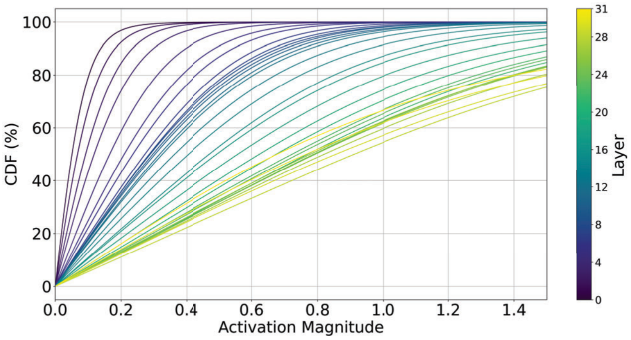


Fig. 7: Activation magnitude distribution of Mistral-7B for all 32 layers (Gate Projection)

Observation 3 (*Mistral-7B*): Figure 7 reveals the activation CDF of the *Gate Projection* component of *Mistral-7B* in all its 32 layers. In *Layer 0*, approximately 80% of the activation values are below *0.1*. From *Layer 1* to *Layer 3*, around 80% of the activation values lie below *0.25*. For the deeper layers, the CDF indicates that applying a cutoff threshold at *1.1* would potentially omit 60% of the FFN neurons to obtain extra sparsity. The trends of activation magnitude distributions for the three evaluated LLMs are similar. The first few layers consist of smaller magnitude activation outputs. For the deeper

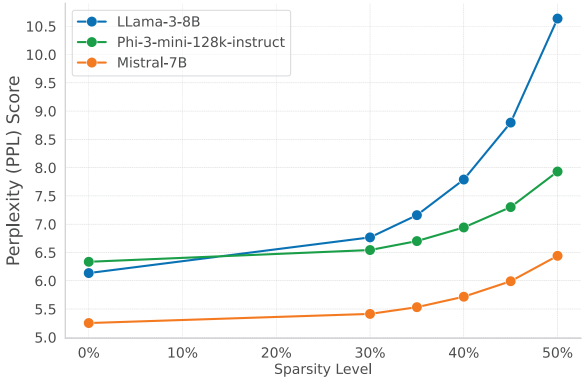


Fig. 8: Tradeoffs between Activation Sparsity and Perplexity (PPL) Scores

layers, the magnitude distribution gradually becomes more even, but they are still limited within relatively small ranges.

Insight: The magnitudes of majority activation values fall into very small data ranges. This observation allows us to set small thresholds to omit fewer contribution weights and easily obtain high sparsity levels.

IV. Tradeoffs between Activation Sparsity and Perplexity

Section III-C concludes that we can obtain a high sparsity level by enforcing relatively small threadholds. However, the new question for enforcing aviation sparsity is how much accuracy degradation will be observed if we get this extra sparsity? This section investigates how much sparsity we can tune while maintaining an acceptable accuracy degradation. The sparsity tuning allows us to make more effective compression and support more powerful LLMs on edge systems. The LLM accuracies are evaluated by the Perplexity (PPL) scores. Similarly to Section III-C, we shows the results of *Phi-3-3.8B*, *LLaMA-3-8B* and *Mistral-7B* for consistency. The *x-axis* represents the sparsity level, ranging from 0% to 50%, indicating the percentage of activations enforced to become zero. The *y-axis* shows the Perplexity (PPL) score, where lower scores indicate better accuracy when executing the *Wikitext* benchmark.

Observation: Figure 8 illustrates tradeoffs between activation sparsity and perplexity (PPL) scores. As the sparsity level increases, the PPL generally rises for all models, indicating that increasing LLM activation sparsity reduces inference accuracy. At 30% sparsity level, all models almost keep the same PPL with 0% sparsity. In other words, all three models can obtain 30% sparsity with only negligible accuracy loss. *LLama-3-8B* shows the steepest increase in PPL, indicating the most significant accuracy drop in performance as sparsity increases. However, even for 50% sparsity, the maximum PPL from *LLama-3-8B* is still less than *11*, which is considered acceptable in many application scenarios. In contrast, *Mistral-7B*'s accuracy is most stable when changing activation sparsity.

Insight: We can obtain an extra 50% activation sparsity for the state-of-the-art LLMs by enforcing the threshold setting while maintaining acceptable PPL/accuracy.

V. Predictability Analysis for LLM Activation Patterns

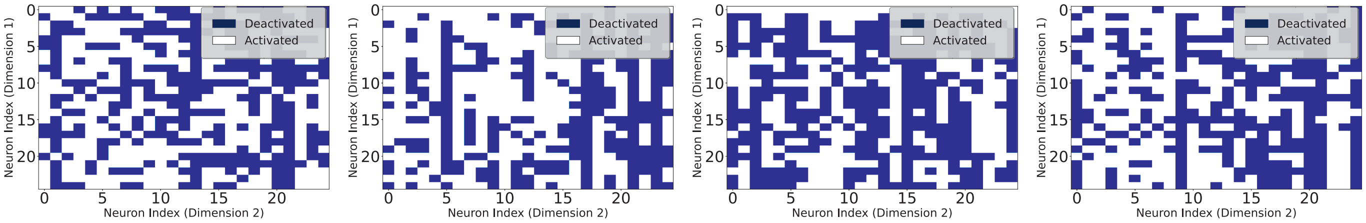
According to our experimental data and analysis in Section IV, we conclude that we can enforce around 50% activation sparsity to state-of-the-art LLMs while maintaining almost the same accuracy/perplexity. A straightforward way to utilize activation sparsity to compress LLMs is via activation pattern prediction. We can predict the future activation pattern and only fetch the possible active weights from the disk or SD card to the main memory. A reasonably predicted success rate can provide benefits, including lower LLM response latency, fewer computing resource requirements, and better main memory utilization. Dhar et al. [2] indicate that memory is the essential bottleneck for LLM edge deployment, significantly increasing the LLM execution latency on resource-constraint devices. Success predictions have great potential to alleviate memory bottlenecks. The incorrect prediction has an extra latency penalty because the system must fetch the mispredicted weights from the disk to the main memory. However, the misprediction in this problem is not a correctness issue. This section will systematically explore LLM activation patterns and analyze their predictability. We aim to provide a guideline for system architects to design an effective prefetcher to assist LLM deployment on edge.

TABLE II: Activation Pattern Match Rates for Similar Inputs

Sample	Similarity Percentages of Input Variants					
	95%	**90%**	**85%**	**80%**	**75%**	**70%**
1	100%	100%	100%	100%	100%	100%
2	57.73%	57.73%	57.73%	57.73%	57.73%	57.73%
3	100%	100%	100%	100%	100%	100%
4	100%	53.83%	53.83%	53.83%	53.93%	53.83%
5	100%	100%	100%	100%	100%	100%
6	100%	100%	100%	100%	100%	100%
7	100%	100%	100%	100%	100%	100%
8	57.73%	57.73%	57.73%	57.73%	57.73%	57.73%
9	100%	100%	100%	100%	100%	100%
10	100%	100%	100%	100%	100%	100%
11	100%	100%	100%	100%	100%	100%
12	100%	100%	100%	100%	100%	100%

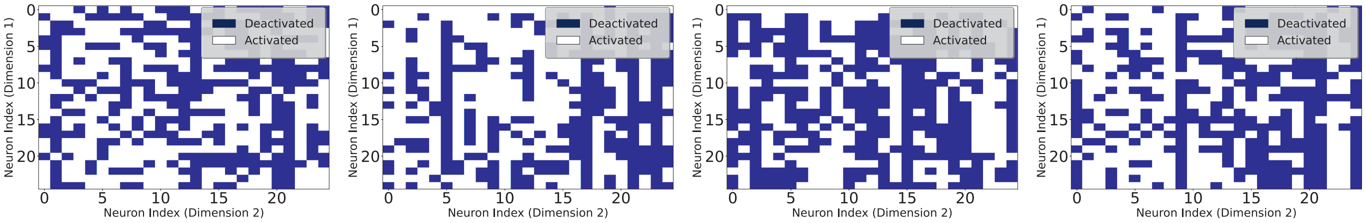
A. *Potential Predict Approach and Evaluation Methodology*

LLM typically consists of multiple deep layers. E.g., *LLaMA-3-8B* has 32 layers. During the LLM inference process, the earlier layer executes first, and its outputs are forwarded to the following layer for further processing. Therefore, we can use the earlier layer's activation patterns to predict future layers' activation patterns. E.g., we may use the pattern in *Layer 1* to predict the patterns of *Layer 16-32*. As shown in ShadowLLM's [14] approach, a lightweight predictor, with only one hidden layer, is capable of predicting activation patterns in future layers. This method introduces minimal computational overhead compared to the benefits it provides. Selectively



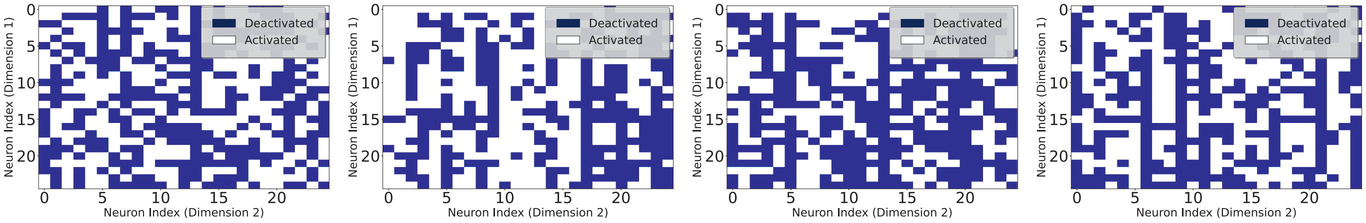
(a) Layer 1 (b) Layer 10 (c) Layer 20 (d) Layer 32

Fig. 9: Activation heatmap pattern of LLaMa-3-8b with default Sample 1 input



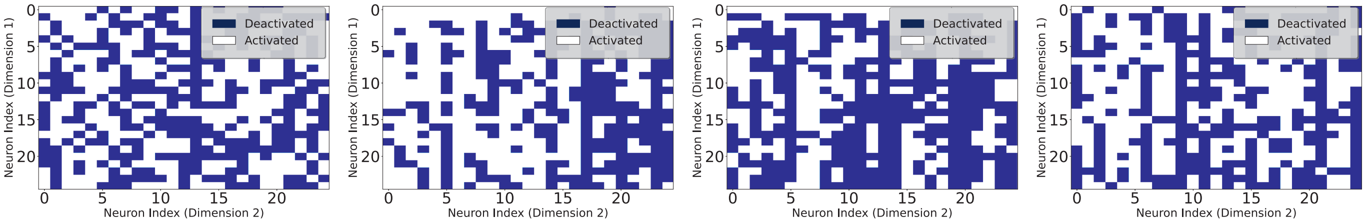
(a) Layer 1 (b) Layer 10 (c) Layer 20 (d) Layer 32

Fig. 10: Activation heatmap pattern of LLaMa-3-8b with 70% similarity Sample 1 input; Comparing with default Sample 1's patterns in Figure 9, matching rates are 100% for all evaluated layers



(a) Layer 1 (b) Layer 10 (c) Layer 20 (d) Layer 32

Fig. 11: Activation heatmap pattern of LLaMa-3-8b with default Sample 8 input



(a) Layer 1 (b) Layer 10 (c) Layer 20 (d) Layer 32

Fig. 12: Activation heatmap pattern of LLaMa-3-8b with 70% similarity Sample 8 input; Comparing with default Sample 8's patterns in Figure 11, matching rates are 57.73%, 71.51%, 71.51%, and 64.06% for Layer 1, 10, 20, and 32 , respectively.

prefetching the predicted weights significantly reduces memory and CPU usage during inference. The system has time to prefetch the predicted weights in the future layers and reduce the disk data loading time. From the main memory's perspective, the executing LLM is effectively compressed if we can achieve a reasonable success rate for pattern prediction.

To evaluate the predictability of the LLM activation patterns, we randomly select 12 samples as the input prompts, where each sample consists of 2048 tokens. User input prompts are always difficult to match 100%, even when they want to ask the same question. Therefore, we also develop variants for these 12 samples with 70% to 95% similarity to tolerate the inconsistency of user inputs. The '70% to 95% similarity' indicates the '30% to 5%' modifications on the original sample. Considering many recent LLMs using the same SwiGLU activation function, for simplicity, we evaluate *LLaMa-3-8b* as a representative in this section.

B. Pattern Similarity and Predictability Analysis

Table II summarizes the activation pattern match rates of *LLaMa-3-8B Layer 1* with different but similar input tokens. For each default input sample (total 12 samples), we developed 6 variants with 95% to 70% similarity. Even with 30% input content change (70% similarity), we still observe that the majority of evaluated samples (9/12) get 100% match rates. For the worst similarity, we obtain 53.83% activation match rate with 70% input similarity. From Table II, we conclude that the activation patterns are not as diverse as user inputs. Similar user inputs are highly possible to share the same activation pattern. This many-to-one mapping feature limits the total number of possible activation outputs, which is the essential prerequisite of effective pattern prediction. Also, Table II illustrates the similarity results of *LLaMa-3-8B Layer 1*. However, we should further confirm if the high matching rate of activation patterns would also be applied in the deeper LLM layers. If matching rates of all layers are high, then theoretically, we can design a predictor to prefetch associate activate neurons for all layers according to user inputs.

Figure 9 and Figure 10 are the comparisons for *Sample 1* where their input similarity is only 70%. For easier reading, we just show the tiny piece of the activation heatmap (25×25 neurons) in four layers. For *Sample 1*, even using varying input prompts (70% similarity), the matching rates of activation patterns of all layers are 100%. Figure 11 and Figure 12 are the pattern comparisons for *Sample 8* and its 70% similarity variant. The matching percentages of *Layer 1*, *Layer 10*, *Layer 20*, and *Layer 32* are 57.73%, 71.51%, 71.51%, and 64.06%, respectively. An effective predictor with prefetching mechanisms may still help in this worst sample type because we can prefetch the majority of neurons in FFN components and, therefore, cut down data loading time from disk.

Insight 1: The results from Table II indicate that LLM activation patterns are possible to share with many similar user inputs. This many-to-one mapping feature narrows the possible output categories and is an essential prerequisite for effective activation pattern prediction.

Insight 2: Most evaluated input samples have 100% same activation patterns for all layers with their 70% similarity variants. The activation patterns of all LLM layers can be considered as an entire predicted output and selected based on *Layer 1*'s activation pattern. In other words, the activation patterns and state-of-the-art LLMs are predictable and can effectively compress new large language models from the main memory's perspective.

Insight 3: Combining the tradeoff conclusion between sparsity and perplexity from Section IV, the effective activation pattern prediction and prefetching can compress 50% of LLMs from system resources' perspective. Therefore, it allows better LLM execution on resource-constraint edge devices.

VI. Related Works

LLM Compression for Edge Systems. Despite the impressive growing capabilities of LLMs, the quick escalation in model size has led to an exponential increase in the memory and computational demands, presenting significant deployment challenges (Kaplan *et al.* [15]; Liu *et al.* [9]) in resource-constraint edge devices. Various compression techniques have been proposed to harvest the benefits of local LLM intelligence in edge devices, which includes quantization [16], [17], pruning [16], [18] and model distillation [19]–[21]. Additionally, efficient sampling methods [22], [23] have been proposed to expedite inference decoding. Our work investigates the potential of forced activation sparsity, which is orthogonal to the above-mentioned approaches and may combine with them to enhance the LLM compression further.

Activation Sparsity Utilization. Activation sparsity occurs when a portion of activation neuron outputs are zero. The computation of these inactive neurons can be ignored, which indicates that they are unnecessary to be fetched into the main memory and execute the computation. Li *et al.* [24] and Liu *et al.* [9] have recognized the intrinsic activation sparsity found in several LLMs and have utilized this characteristic to speed up inference. These techniques can be combined to enhance performance effectively. Despite these insights, no prior work has provided concrete model results after employing activation sparsity. However, our study aims to bridge this gap by not only implementing activation sparsity but also presenting detailed performance metrics, thus validating the practical benefits of this approach.

ReLUfication. The activation sparsity benefits from previous research are limited to ReLU-based Transformer architectures, including LLMs such as T5 (Raffel *et al.* [25]) and OPT (Zhang *et al.* [26]), as well as in some vision models like ViT (Dosovitskiy *et al.* [27]). However, recent LLMs such as Falcon (Almazrouei *et al.* [28]) and LLaMa (Touvron *et al.* [29]) predominantly use non-ReLU activation functions like GELU (Hendrycks and Gimpel *et al.* [30]) and Swish (Elfwing *et al.* [31]), which do not directly provide activation sparsity. To utilize the benefits of activation sparsity without developing a ReLU-activated LLM from scratch, many researchers have adopted a technique known as ReLUfication. This method introduces sparse ReLU-based activations into non-ReLU LLMs. For instance, Zhang *et al.* [32] successfully converted a GELU-activated BERT (Devlin *et al.* [33]) into a ReLU-activated version by directly swapping the activation function. Similarly, Zhang *et al.* [34] applied this approach to Falcon and LLaMa, which originally used GELU and Swish, respectively. However, simply replacing the activation function often fails to achieve satisfactory sparsity, primarily due to the unaddressed inherent limitations of the original non-ReLU activation distributions. Instead of utilizing ReLUfication, we propose to enforce sparsity without altering the activation function. This method retains the original activation function's properties while leveraging sparsity benefits. Our approach directly omits the lower magnitude values to induce sparsity, offering a novel and effective method for LLM compression.

VII. Conclusion

Deploying LLM on edge devices offers numerous compelling benefits. Recently, major tech companies have started to invest in and seek edge LLM solutions. However, the insufficient computing and memory resources on edge devices restricted effective LLM executions. Most regular edge devices are still unable to meet the requirements of LLM execution, even when combined with existing compression techniques. Therefore, we explore new schemes that could seamlessly combine with regular LLM compression approaches. The new activation functions of state-of-the-art LLMs eliminate the feasibility of Apple's compression approach, which performs well for ReLU-based LLMs. In this work, we search for new compression opportunities from activation sparsity, which can benefit the majority of general LLMs and not be bound by specific activation functions. Our systematic explorations indicate that we can safely secure 50% extra sparsity in FFN layers with a negligible accuracy loss for the state-of-the-art LLMs. This finding allows us to compress the LLMs from memory's perspective via prediction and prefetching. Moreover, to verify the predictability of LLM activation patterns, we also analyze matching rates of LLM activation patterns with multiple user input variants. We finally conclude that the LLM activation patterns are highly predictable and have a huge potential to get extra sparsity. The paper provides a guideline for design pattern predictors with prefetching capabilities, leading to less LLM execution latency, lower power cost, and improved user experience. Moreover, besides general LLMs, our new compression approach can be simply extended to other large-scale transformer-based AI models.

Acknowledgement

We are grateful to the anonymous reviewers for their comments and suggestions on this paper. This work was supported in part by U.S. National Science Foundation (NSF) grants CPS-2103459, SHF-2210744.

References

[1] M. Lubbad, "Gpt-4 parameters: Unlimited guide nlp's game-changer," 2023.

[2] N. Dhar, B. Deng, D. Lo, X. Wu, L. Zhao, and K. Suo, "An empirical analysis and resource footprint study of deploying large language models on edge devices," 2024.

[3] A. Ullah, G. Qi, S. Hussain, I. Ullah, and Z. Ali, "The role of llms in sustainable smart cities: Applications, challenges, and future directions," 2024.

[4] W. Yin, M. Xu, Y. Li, and X. Liu, "Llm as a system service on mobile devices," 2024.

[5] J. Yu, A. Lukefahr, and Palframan..., "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," 2017.

[6] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X.-s. Hua, "Quantization networks," 2019.

[7] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, 2021.

[8] J. Kepner, S. Alford, V. Gadepally, M. Jones, L. Milechin, A. Reuther, R. Robinett, and S. Samsi, "Graphchallenge. org sparse deep neural network performance," *arXiv preprint arXiv:2004.01181*, 2020.

[9] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, A. Shrivastava, C. Zhang, Y. Tian, C. Ré, and B. Chen, "Deja vu: contextual sparsity for efficient llms at inference time." JMLR.org, 2023.

[10] K. Alizadeh, I. Mirzadeh, D. Belenko, K. Khatamifard, M. Cho, C. C. D. Mundo, M. Rastegari, and M. Farajtabar, "Llm in a flash: Efficient large language model inference with limited memory," 2024.

[11] J. Liu, Y. Song, K. Xue, H. Sun, C. Wang, L. Chen, H. Jiang, J. Liang, and T. Ruan, "Fl-tuning: Layer tuning for feed-forward network in transformer," *arXiv preprint arXiv:2206.15312*, 2022.

[12] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," 2016.

[13] T. P. Pires, A. V. Lopes, Y. Assogba, and H. Setiawan, "One wide feedforward is all you need," 2023.

[14] Y. Akhauri, A. F. AbouElhamayed, J. Dotzel, Z. Zhang, A. M. Rush, S. Huda, and M. S. Abdelfattah, "Shadowllm: Predictor-based contextual sparsity for large language models," 2024. [Online]. Available: https://arxiv.org/abs/2406.16635

[15] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," 2020.

[16] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2016.

[17] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," 2017.

[18] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2017.

[19] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[20] R. Tang, Y. Lu, L. Liu, L. Mou, O. Vechtomova, and J. Lin, "Distilling task-specific knowledge from bert into simple neural networks," 2019.

[21] Y. Gu, L. Dong, F. Wei, and M. Huang, "Minillm: Knowledge distillation of large language models," 2024.

[22] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding." PMLR, 2023.

[23] Y. Wang, K. Chen, H. Tan, and K. Guo, "Tabi: An efficient multi-level inference system for large language models." Association for Computing Machinery, 2023.

[24] Z. Li, C. You, S. Bhojanapalli, D. Li, A. S. Rawat, S. J. Reddi, K. Ye, F. Chern, F. Yu, R. Guo, and S. Kumar, "The lazy neuron phenomenon: On emergence of activation sparsity in transformers," 2023.

[25] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," 2023.

[26] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "Opt: Open pre-trained transformer language models," 2022.

[27] 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," 2021.

[28] E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocaru, M. Debbah, Étienne Goffinet, D. Hesslow, J. Launay, Q. Malartic, D. Mazzotta, B. Noune, B. Pannier, and G. Penedo, "The falcon series of open language models," 2023.

[29] H. Touvron, L. Martin, K. Stone, and e. a. Peter Albert, "Llama 2: Open foundation and fine-tuned chat models," 2023.

[30] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," 2023.

[31] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," 2017.

[32] Z. Zhang, Y. Lin, Z. Liu, P. Li, M. Sun, and J. Zhou, "MoEfication: Transformer feed-forward layers are mixtures of experts." Association for Computational Linguistics, 2022.

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[34] Z. Zhang, Y. Song, G. Yu, X. Han, Y. Lin, C. Xiao, C. Song, Z. Liu, Z. Mi, and M. Sun, "Relu^2 wins: Discovering efficient activation functions for sparse llms," *arXiv preprint arXiv:2402.03804*, 2024.