ReLU Strikes Back: Exploiting Activation Sparsity in Large Language Models

Iman Mirzadeh, Keivan Alizadeh, Sachin Mehta, Carlo C Del Mundo,
Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, Mehrdad Farajtabar
Apple

Abstract

Large Language Models (LLMs) with billions of parameters have drastically transformed AI applications. However, their demanding computation during inference has raised significant challenges for deployment on resource-constrained devices. Despite recent trends favoring alternative activation functions such as GELU or SiLU, known for increased computation, this study strongly advocates for reinstating ReLU activation in LLMs. We demonstrate that using the ReLU activation function has a negligible impact on convergence and performance while significantly reducing computation and weight transfer. This reduction is particularly valuable during the memory-bound inference step, where efficiency is paramount. Exploring sparsity patterns in ReLU-based LLMs, we unveil the reutilization of activated neurons for generating new tokens and leveraging these insights, we propose practical strategies to substantially reduce LLM inference computation up to three times, using ReLU activations with minimal performance trade-offs.

1 Introduction

The enthusiasm for Large Language Models (LLMs) has spurred interest in AI across various domains [5, 9, 6]. However, the efficient utilization of LLMs faces challenges due to substantial computational and memory demands during inference [58, 39, 3]. To improve inference efficiency, techniques like quantization, speculative decoding, pruning, and weight sparsification have been explored [11, 48, 40, 51, 69, 19, 14]. Activation sparsity is particularly promising, offering a favorable trade-off between accuracy and speedup, especially on modern GPUs [49].

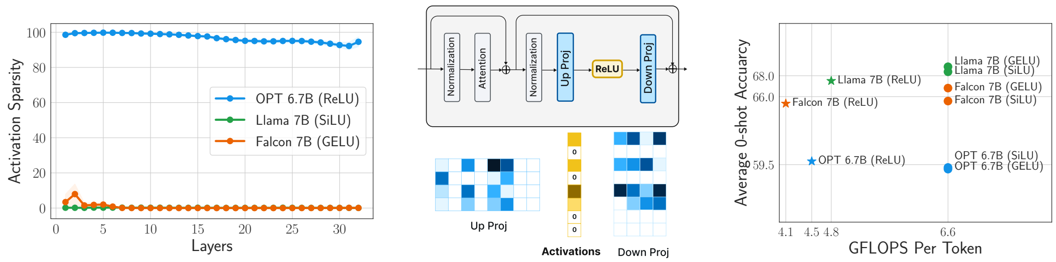
Notably, using Rectified Linear Unit (ReLU) activation induces sparse activations and has been widely adopted in prior works [26, 42, 46, 67]. We affirm this property using the OPT model [78] and measure the sparsity of the activations in the Fed Forward Network (FFN) layers (Fig. 1a). This sparsity substantially reduces computation (Fig. 1b), resulting in a 32% savings in FLOPS[1] (Fig. 1c).

However, a recent trend has emerged, favoring variations of ReLU that are smoother but more complex [27, 62]. These alternatives have gained popularity due to their slightly faster convergence and improved final accuracy [64]. For example, PaLM [9] and Llama models [71] adopt SiLU[2] [27, 16, 62], while MPT [54] and Falcon models [2] use GELU [27]. Nonetheless, as demonstrated in Fig. 1c, when we finetune several pretrained LLMs with different activation functions, their performance does not change significantly (within a specific model), while ReLU models require much less computation.

[1]In this work, we use FLOPS as a proxy for inference efficiency. In Appendix C, we demonstrate that for LLMs with activation sparsity, FLOPS can serve as a good approximation of real-world efficiency due to the structure inherent in activation sparsity (e.g., skipping the entire row corresponding to zero activations).

[2]To be more precise, the mentioned models use SwiGLU activation function, but in this work, we focus on the gating module that uses SiLU (Swish) function.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).



(a) Sparsity of different models (b) Sparsity for Efficiency (c) Accuracy vs. Computation

Figure 1: **(a)** Activation Sparsity of different pretrained models: ReLU-based OPTs show significantly higher sparsity. **(b)** Zeroed out entries after ReLU save compute in large semi-structured chunks (e.g., rows). **(c)** Comparison of inference efficiency and performance of the different models with different activation functions after fine-tuning: The choice of activation function does not significantly impact the accuracy, as any of GELU, SiLU, or ReLU can be used on all three models and achieve the same level of accuracy as the original activation function. However, using ReLU can provide an additional benefit of leading to activation sparsity and faster inference.

In this paper, we re-evaluate using ReLU for LLMs, emphasizing computational efficiency during inference. We demonstrate comparable performance with other activation functions when trained from scratch (Sec. 2) and highlight the benefits of fine-tuning existing pretrained LLMs with ReLU activations (Sec. 3). Moreover, built on the insights from our study, we propose several promising directions that can improve inference efficiency of LLMs (Sec. 4).

2 Does the Activation Function Matter?

This section first overviews our experimental setup, then, by training various models from scratch with different activation functions, we demonstrate that changing activation functions minimally impacts performance. However, the impact on inference efficiency is substantial.

Experimental Setup. We use open source pretrained models such as OPT [78], Llama (v1) [71], and Falcon [2] as they use different architectures and pretraining setup (e.g., attention/FFN structure/normalization, activation functions), allowing our study covers a wider range of models. We use the RefinedWeb dataset [57], for our pretraining in Sec. 2.1 and finetuning pretrained models in Sec. 3. We chose RefinedWeb because it is a high-quality subset of Common Crawl, which is often used in the pretraining phase of LLMs, including Llama, Falcon, and OPT. We also use the validation split of WikiText [52] for measuring the sparsity and recording preactivation distributions of various pretrained models. However, our conclusions hold for other datasets we have tested.

2.1 Training from scratch: performance and sparsity

While the previous literature suggests that non-ReLU variants can improve the performance of transformers [64, 55], we argue the impact is marginal at best. To support our claim, we train the OPT 1.3B model from scratch on a hundred billion tokens of the RefinedWeb datasets with different activation functions, including ReLU, SiLU, and GELU. All these activation functions can be viewed as f(x) $= x \cdot \sigma(\beta x)$, where β controls the gating part (smoothed cutoff threshold) of the activation function (see Fig. 2a). For $\beta = 1$, we will have SiLU($x \cdot \sigma(x)$), and $\beta = 1.7$ is a good approximation of GELU. Finally, as $\beta \to \infty$, the activation function becomes closer to ReLU. To further explore the spectrum of ReLU to SiLUwe add another one with $\beta = 8$.

As shown in the bottom row of Fig. 2, the performance of the models is very similar when using different activation functions. This is consistent with the scaling laws literature ([36, 30]), which suggests that the performance of sufficiently large models trained on sufficiently large data depends heavily on compute and data, not architectural details.

While the performance levels of the different activations are similar, their activation sparsity levels differ. Here, we define sparsity as the average sparsity level across all layers for each model. Shown in Fig. 2c, as we transition from SiLU to ReLU (increasing β), the sparsity also increases. This results

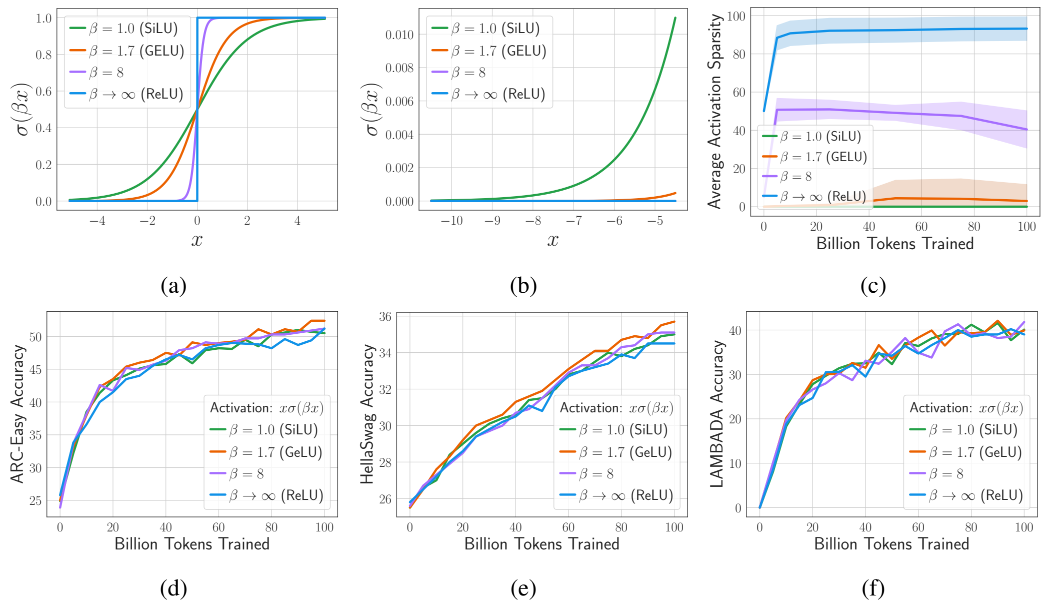


Figure 2: **(top)** (a) Shapes of different gating functions over [-5, 5]; (b) Continuation of (a) where SiLU is comparably larger compared to others; (c) Sparsity of the FFN with different activations: increasing β increases sparsity. **(bottom)** when trained from scratch, OPT 1.3 B models using different activation functions achieve similar performance.

from the different gating thresholds, as ReLU drops significantly more values compared to GELU and SiLU (see Fig. 2b). In Appendix F.1, we illustrate the evolution of the pre-activation distribution throughout training.

Overall, the results support our initial claim: non-ReLU activations result in a negligible performance gain (if any) but a substantial loss in sparsity and efficiency. While, at times, the performance of GeLU or SiLU might be slightly higher, ReLU can match it with slightly longer training. We acknowledge that to compensate for the small gap in performance, we need to pay the one-time cost of longer training. However, in return, we get a significantly more sparsity.

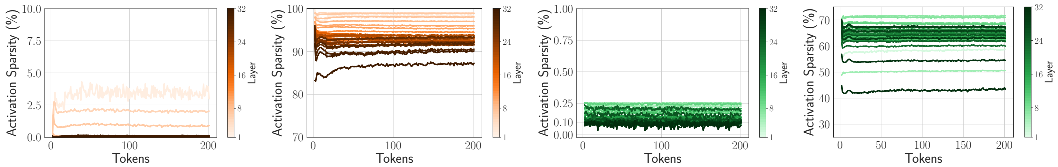
3 Relufication: Improving Activation Sparsity of Pretrained LLMs

While in the previous section, we have seen that the performance does not depend on the activation function, we note that most of the available pretrained LLMs are trained with activation functions other than ReLU. Hence, to incorporate the computational benefits of ReLU activations at inference time, we perform various architectural surgeries and study the consequences of such changes.

We present our findings about incorporating ReLU activations into the pretrained LLMs, a process we refer to as *relufication*. More specifically, we show that replacing the activation functions of pretrained LLMs with ReLU is possible, and the performance can be recovered very rapidly during finetuning. We show these modifications, which are easy to implement, lead to more efficient models at inference time while maintaining comparable performance to the original pretrained models.

The process of *relufication* for different pretrained architectures is shown in Fig. 6. This process can be done in multiple stages, but here we focus on the first stage and postpone the second stage to Appendix D. The first and more intuitive stage replaces non-ReLU activations with ReLU in the FFN layer. For the Falcon and Llama models, this means replacing GELU and SiLU, respectively. We note that since OPT models already use ReLU activations, we keep those unchanged. After finetuning on 30 billion tokens of the RefinedWeb, Fig. 3 shows that the modified models have significantly more sparsity in their activations.

In addition to the drastic improvement in activation sparsity, we can make several notable observations. First, while the shape of preactivation depends on the pretraining dynamics and architecture, in Fig. 9,



(a) Falcon 7B (GELU) (b) Falcon 7B (ReLU) (c) Llama 7B (SiLU) (d) Llama 7B (ReLU)

Figure 3: Activation sparsity of Falcon and Llama models improves significantly after *relufication*.

Table 1: Comparing performance across several tasks: After the first stage of *relufication*, the activation sparsity of models increases significantly, hence increased efficiency measured by FLOPS. Within each group, the performance levels are comparable.

Model	Sparsity (%)	FLOPS (G)	0-shot Accuracy (%) Avg	Arc-E	Arc-C	Hellaswag	BoolQ	PIQA	LAMBADA	TriviaQA	WinoGrande	SciQ	MMLU (5-shot)
Falcon 7B (ReLU)	**94**	**4.1**	**65.2**	72.2	39.1	55.4	70.6	78.4	69.2	40.5	67.5	93.1	**27.9**
Falcon 7B (GELU)	1	6.6	**66.8**	74.6	40.2	57.7	73.5	79.4	74.5	40.4	67.2	94.0	**27.7**
Llama 7B (ReLU)	**62**	**4.8**	**67.1**	75.2	40.1	55.2	73.4	77.7	71.5	49.6	67.1	94.2	**34.7**
Llama 7B (SiLU)	0	6.6	**68.4**	75.5	42.1	69.9	74.8	78.7	73.1	49.9	69.8	95.4	**35.1**

we show that it does not change significantly during the relatively short finetuning stage. As a result, we can predict the activation sparsity before finetuning, knowing it will not change significantly. Later in Sec. E.3 we build on this observation and propose shifting the preactivation values before applying ReLU and further increasing the activation sparsity. The stability of the preactivation distribution may suggest that the behavior of the network does not change while creating sparse representations. Indeed, we show that after replacing the activation function with ReLU, finetuned models quickly recover their performance in Fig. 4. We believe optimizing this process even further (e.g., using better finetuning data) is an exciting follow-up direction.

4 Conclusion & Future Directions

In this study, we conducted a large-scale investigation of the activation functions, and we have shown that the choice of activation functions during pretraining and finetuning does not have a significant impact on performance while using ReLU can provide an additional benefit of leading to activation sparsity and more efficient inference.

To bridge the gap between existing pre-trained models and our work, we have *relufied* several models to incorporate the ReLU activation function into the architecture of these already pre-trained models. We have shown that across several zero-shot and few-shot tasks, the ReLU-based LLMs perform similarly to their non-ReLU models at a significantly reduced computation.

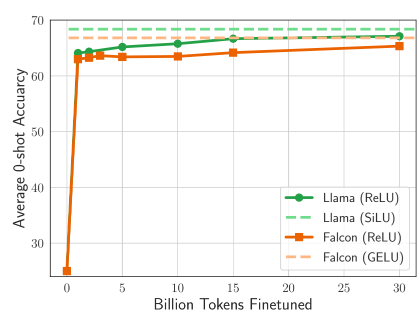


Figure 4: Evolution of zero-shot accuracy during finetuning: The model quickly recovers most of its lost performance due to the architecture surgery.

As a stepping stone for future works, in Appendix D, we continue the *relufication* process and we show that we can improve the efficiency of several LLMs even more. In addition, in Appendix E, we investigate two promising applications of activation sparsity in LLMs, and we show that we can leverage these sparse networks for faster inference via activation reuse and speculative decoding [40], in addition to exploring unconventional but promising activation functions for LLMs.

We believe our work is among the few studies that investigate changes in the architectural components of LLMs on a large scale. We hope our findings motivate the community to further investigate the advantages of well-structured activation sparsity, ultimately enhancing the efficiency of these models.

References

[1] Rishabh Agarwal, Nino Vieillard, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. Gkd: Generalized knowledge distillation for auto-regressive sequence models. *CoRR*, 2023.

[2] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Maitha Alhammadi, Mazzotta Daniele, Daniel Heslow, Julien Launay, Quentin Malartic, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. The falcon series of language models: Towards open frontier models. 2023.

[3] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2022.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[6] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[7] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. Quip: 2-bit quantization of large language models with guarantees. *CoRR*, abs/2307.13304, 2023. doi: 10.48550/arXiv.2307.13304.

[8] Tianyu Chen, Shaohan Huang, Yuan Xie, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. Task-specific expert pruning for sparse mixture-of-experts. *ArXiv*, abs/2206.00277, 2022.

[9] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[10] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 933–941. JMLR.org, 2017.

[11] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

[12] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *CoRR*, 2023.

[13] Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. Spqr: A sparse-quantized representation for near-lossless LLM weight compression. *CoRR*, abs/2306.03078, 2023. doi: 10.48550/arXiv.2306.03078.

[14] Gaochen Dong and Wei Chen. Blockwise compression of transformer-based models without retraining. *arXiv preprint arXiv:2304.01483*, 2023.

[15] Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten P. Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen S. Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V. Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. Glam: Efficient scaling of language models with mixture-of-experts. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022,*

17-23 July 2022, Baltimore, Maryland, USA, volume 162 of *Proceedings of Machine Learning Research*, pages 5547–5569. PMLR, 2022.

[16] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018.

[17] William Fedus, Jeff Dean, and Barret Zoph. A review of sparse expert models in deep learning. *ArXiv*, abs/2209.01667, 2022.

[18] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23:120:1–120:39, 2022.

[19] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 10323–10337. PMLR, 2023. URL `https://proceedings.mlr.press/v202/frantar23a.html`.

[20] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: accurate post-training quantization for generative pre-trained transformers. *CoRR*, abs/2210.17323, 2022. doi: 10.48550/arXiv.2210.17323.

[21] Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Trans. Syst. Sci. Cybern.*, 5:322–333, 1969.

[22] Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, September 2021. URL `https://doi.org/10.5281/zenodo.5371628`.

[23] Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Knowledge distillation of large language models. *CoRR*, 2023.

[24] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. Retrospective: Eie: Efficient inference engine on sparse and compressed neural network, 2023.

[25] Hussein Hazimeh, Zhe Zhao, Aakanksha Chowdhery, Maheswaran Sathiamoorthy, Yihua Chen, Rahul Mazumder, Lichan Hong, and Ed H. Chi. Dselect-k: Differentiable selection in the mixture of experts with applications to multi-task learning. In *Neural Information Processing Systems*, 2021.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[27] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[28] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=d7KBjmI3GmQ`.

[29] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *CoRR*, 2015.

[30] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. *CoRR*, abs/2203.15556, 2022. doi: 10.48550/arXiv.2203.15556.

[31] Jiri Hron, Yasaman Bahri, Jascha Sohl-Dickstein, and Roman Novak. Infinite attention: NNGP and NTK for deep attention networks. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 4376–4386. PMLR, 2020.

[32] Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 8003–8017. Association for Computational Linguistics, 2023. URL `https://doi.org/10.18653/v1/2023.findings-acl.507`.

[33] Ranggi Hwang, Jianyu Wei, Shijie Cao, Changho Hwang, Xiaohu Tang, Ting Cao, Mao Yang, and Minsoo Rhu. Pre-gated moe: An algorithm-system co-design for fast and scalable mixture-of-expert inference. *arXiv preprint arXiv:2308.12066*, 2023.

[34] Ajay Jaiswal, Shiwei Liu, Tianlong Chen, and Zhangyang Wang. The emergence of essential sparsity in large pre-trained models: The weights that matter. *CoRR*, 2023.

[35] Sebastian Jaszczur, Aakanksha Chowdhery, Afroz Mohiuddin, Lukasz Kaiser, Wojciech Gajewski, Henryk Michalewski, and Jonni Kanerva. Sparse is enough in scaling transformers. In *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=-b5OSCydOMe`.

[36] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL `https://arxiv.org/abs/2001.08361`.

[37] Jeonghoon Kim, Jung Hyun Lee, Sungdong Kim, Joonsuk Park, Kang Min Yoo, Se Jung Kwon, and Dongsoo Lee. Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization. *CoRR*, abs/2305.14152, 2023. doi: 10.48550/arXiv.2305.14152. URL `https://doi.org/10.48550/arXiv.2305.14152`.

[38] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. *CoRR*, abs/2306.07629, 2023. doi: 10.48550/arXiv.2306.07629.

[39] Sehoon Kim, Coleman Hooper, Thanakul Wattanawong, Minwoo Kang, Ruohan Yan, Hasan Genc, Grace Dinh, Qijing Huang, Kurt Keutzer, Michael W Mahoney, et al. Full stack optimization of transformer inference. In *Architecture and System Support for Transformer Models (ASSYST@ ISCA 2023)*, 2023.

[40] Sehoon Kim, Karttikeya Mangalam, Suhong Moon, John Canny, Jitendra Malik, Michael W. Mahoney, Amir Gholami, and Kurt Keutzer. Speculative decoding with big little decoder, 2023.

[41] Rui Kong, Yuanchun Li, Qingtian Feng, Weijun Wang, Linghe Kong, and Yunxin Liu. Serving moe models on resource-constrained edge devices via dynamic expert swapping, 2023.

[42] Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Nir Shavit, and Dan Alistarh. Inducing and exploiting activation sparsity for fast inference on deep neural networks. In *International Conference on Machine Learning*, pages 5533–5543. PMLR, 2020.

[43] Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. OWQ: lessons learned from activation outliers for weight quantization in large language models. *CoRR*, abs/2306.02272, 2023. doi: 10.48550/arXiv.2306.02272.

[44] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR, 2023.

[45] Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, et al. Large models are parsimonious learners: Activation sparsity in trained transformers. *arXiv preprint arXiv:2210.06313*, 2022.

[46] Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J. Reddi, Ke Ye, Felix Chern, Felix X. Yu, Ruiqi Guo, and Sanjiv Kumar. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[47] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. AWQ: activation-aware weight quantization for LLM compression and acceleration. *CoRR*, abs/2306.00978, 2023. doi: 10.48550/arXiv.2306.00978.

[48] Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. Llm-qat: Data-free quantization aware training for large language models. *CoRR*, 2023.

[49] Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR, 2023.

[50] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL `https://openreview.net/forum?id=Bkg6RiCqY7`.

[51] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*, 2023.

[52] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[53] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 5191–5198, 2020.

[54] NLP Team MosaicML. Introducing mpt-7b: A new standard for open-source, commercially usable llms, 2023. URL `www.mosaicml.com/blog/mpt-7b`.

[55] Sharan Narang, Hyung Won Chung, Yi Tay, Liam Fedus, Thibault Févry, Michael Matena, Karishma Malkan, Noah Fiedel, Noam Shazeer, Zhenzhong Lan, Yanqi Zhou, Wei Li, Nan Ding, Jake Marcus, Adam Roberts, and Colin Raffel. Do transformer modifications transfer across implementations and applications? In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 5758–5773. Association for Computational Linguistics, 2021.

[56] Gunho Park, Baeseong Park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models. *CoRR*, 2023.

[57] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The refinedweb dataset for falcon LLM: outperforming curated corpora with web data, and web data only. *CoRR*, abs/2306.01116, 2023. doi: 10.48550/arXiv.2306.01116.

[58] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5, 2023.

[59] Joan Puigcerver, Carlos Riquelme, Basil Mustafa, and Neil Houlsby. From sparse to soft mixtures of experts. *arXiv preprint arXiv:2308.00951*, 2023.

[60] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: memory optimizations toward training trillion parameter models. In Christine Cuicchi, Irene Qualters, and William T. Kramer, editors, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, page 20. IEEE/ACM, 2020.

[61] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation AI scale. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 18332–18346. PMLR, 2022.

[62] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[63] Michael Santacroce, Zixin Wen, Yelong Shen, and Yuanzhi Li. What matters in the structured pruning of generative language models? *CoRR*, 2023.

[64] Noam Shazeer. Glu variants improve transformer, 2020.

[65] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[66] Kai Shen, Junliang Guo, Xu Tan, Siliang Tang, Rui Wang, and Jiang Bian. A study on relu and softmax in transformer. *CoRR*, 2023.

[67] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single GPU. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 31094–31116. PMLR, 2023.

[68] Zhao Song, Lichen Zhang, and Ruizhe Zhang. Training multi-layer over-parametrized neural network in subquadratic time, 2021.

[69] Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. *CoRR*, 2023.

[70] NLLB Team, Marta R. Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, Anna Sun, Skyler Wang, Guillaume Wenzek, Al Youngblood, Bapi Akula, Loic Barrault, Gabriel Mejia Gonzalez, Prangthip Hansanti, John Hoffman, Semarley Jarrett, Kaushik Ram Sadagopan, Dirk Rowe, Shannon Spruit, Chau Tran, Pierre Andrews, Necip Fazil Ayan, Shruti Bhosale, Sergey Edunov, Angela Fan, Cynthia Gao, Vedanuj Goswami, Francisco Guzmán, Philipp Koehn, Alexandre Mourachko, Christophe Ropers, Safiyyah Saleem, Holger Schwenk, and Jeff Wang. No language left behind: Scaling human-centered machine translation, 2022.

[71] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023.

[72] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference*

on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 5998–6008, 2017.

[73] Mitchell Wortsman, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Replacing softmax with relu in vision transformers, 2023.

[74] Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 38087–38099. PMLR, 2023.

[75] Rongjie Yi, Liwei Guo, Shiyun Wei, Ao Zhou, Shangguang Wang, and Mengwei Xu. Edgemoe: Fast on-device inference of moe-based large language models. *arXiv preprint arXiv:2308.14352*, 2023.

[76] Biao Zhang and Rico Sennrich. Root mean square layer normalization. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12360–12371, 2019.

[77] Mingyang Zhang, Hao Chen, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, and Bohan Zhuang. Pruning meets low-rank parameter-efficient fine-tuning. *CoRR*, 2023.

[78] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. OPT: open pre-trained transformer language models. *CoRR*, abs/2205.01068, 2022.

[79] Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Moefication: Transformer feed-forward layers are mixtures of experts. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 877–890. Association for Computational Linguistics, 2022.

[80] Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A survey on model compression for large language models. *CoRR*, 2023.

[81] Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. St-moe: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022.

Appendix

The appendix is structured as follows:

- Appendix A expands the brief discussion of our experimental setup for this work.
- Appendix B provides a detailed discussion of the existing literature related to our work.
- Appendix C provides an important discussion on our *efficiency model*. We show that unlike the case for unstructured sparsity, activation sparsity is structured (e.g., skipping rows and columns in the calculation). Hence, in this case, FLOPS can be a reasonable proxy for efficiency.
- Appendix D continues the *relufication* process we studied in Sec. 3. We introduce the second stage of the relufication process to increase the sparsity, and we show that this has a negligible impact on performance.
- Appendix E discusses two promising directions based on the activation sparsity. We study the relationship between activation sparsity and relufication (Sec. E.1, and E.2). Moreover, we show that by understanding the preactivation distribution of the pretrained model, we can explore the idea of using unconventional activation functions such as *shifted relu* to improve the activation sparsity significantly (Sec. E.3).
- Finally, in Appendix F, we provide additional results on the preactivation distribution of OPT models trained from scratch and the viewing of the relufication process as a compression technique.

A Extended Experimental Setup

Models. We use open source pretrained models such as OPT [78], Llama (v1) [71], and Falcon [2] as they use different architectures and pretraining setup (e.g., attention/FFN structure/normalization, activation functions), allowing our study to cover a wider range of models.

Datasets. We use the RefinedWeb dataset [57], for our pretraining in Sec. 2.1 and finetuning pretrained models in Sec. 3. We chose RefinedWeb because it is a high-quality subset of Common Crawl, which is often used in the pretraining phase of LLMs, including Llama, Falcon, and OPT. We also use the validation split of WikiText [52] for measuring the sparsity and recording preactivation distributions of various pretrained models. However, our conclusions hold for other datasets we have tested.

Training and Finetuning. For finetuning the pretrained models, we follow the original pretraining recipe, except we use a fixed learning rate of 1.5e-5 for Llama 7B, Falcon 7B, and OPT 6.7B models. In addition, we use the AdamW optimizer [50] for our finetuning with ZeRO stage 1 [60], where we shard the optimizer states across different GPUs. For pretraining OPT 1.3B models from scratch in Sec. 2.1, we follow the OPT training recipe.

Evaluation. For our *performance* evaluation, we use the few-shot tasks from Language Model Evaluation Harness [22]. We select these tasks such that they can measure various abilities of the models (e.g., reading comprehension, reasoning, etc.), and we aim to be consistent with other works in the literature to make the comparison easier. Consistent with the other sections, we compare activation sparsity as a measure of *efficiency*. Further details regarding the relationship between activation sparsity, FLOPS, and inference efficiency are discussed in Appendix C.

B Related Works

Activation Functions in Transformers. The original Transformer architecture [72] was proposed with the ReLU activation function [21], following the popularity of ReLU at the time. Later, several studies aimed to improve the ReLU activation function by increasing its smoothness [27] and/or including parameterized gating mechanisms, such as GELU, SiLU, GLU, and SwiGLU [10, 62]. Earlier studies demonstrated the benefits of these alternatives to ReLU for transformers [64, 55], but on a small scale (e.g., they trained models up to a couple of 100M parameters with at most 35B tokens, while in this work, we train 1B parameter models on more than 100B tokens). However, we believe the impact of activation functions on performance is marginal, following scaling laws [36, 30], which state that architectural changes do not significantly impact performance.

Activation Sparsity. A body of prior research [42, 24, 68] has demonstrated that increasing sparsity can lead to reductions in both inference and training times. Dejavu [49] and [45] observed pronounced sparsity in activations when using the ReLU function in feedforward layers. These studies propose that predicting this sparsity can further boost inference speeds. Similarly, [35] employed ReLU activations and introduced a controller to actively promote sparsity. Notably, these studies predominantly focus on networks employing ReLU activations, leaving out those with alternative activation functions. In contrast, our approach modifies networks by substituting other activation functions with ReLU. We then fine-tune these networks to achieve activation sparsity in the MLP layer post-ReLUfication. We further illustrate that inserting ReLU prior to the QKV and Feedforward layers can substantially reduce FLOPS, albeit at a minor cost to accuracy. Unlike the aforementioned studies, we do not utilize a sparsity predictor to minimize FLOPS.

ReLU in Attention Mechanisms. Beyond the activation function in the MLPs of large language models, a softmax activation is often employed within the attention module. Prior studies have indicated that it's feasible to replace this softmax with ReLU without compromising accuracy [73, 66, 31]. This avenue of research is distinct from our approach of Relufication, which specifically focuses on activations preceding weight multiplications.

Model compression for efficient inference Quantization, pruning and distillation are the main three techniques for compressing neural networks [80]. Quantization has been used to reduce model size and faster inference [12, 48, 56, 11, 47, 43, 13, 38, 7, 74]. The quantized model occupies less space reducing the memory latency [20, 37]. Reluification is orthogonal to quantization and reduces the amount of memory required to be loaded and can further decrease the memory latency. Distillation [32, 29, 23, 53, 1] is another technique to train smaller models. This is orthogonal to using ReLU activations as any activation can be used in distillation methods. Sparsifying or pruning weights of neural networks [19, 34, 77, 69, 63, 51] can reduce computation and inference time. Weight sparsity is usually unstructured and hard to implement for hardware, but the sparsity induced by ReLU can easily be implemented as a matrix multiplication of non zero rows. Weight sparse models can be combined with our relufication for further decrease in compute.

Mixture of Experts. Mixture of Experts (MoE) LLMs usually subdivide the feed-forward layer into multiple experts. A router is then employed to selectively and sparsely activate these experts [65, 18, 70]. Similar to our work, MoE is a form of activation sparsity but in a group form and can be seen as a subset of sparse activation. Subsequent studies have further refined the inference and training methodologies for MoE models [59, 33, 75, 15, 41, 61, 81, 8, 25]. MoE can be also combined with Relufication, having sparsity inside FFN of each expert.

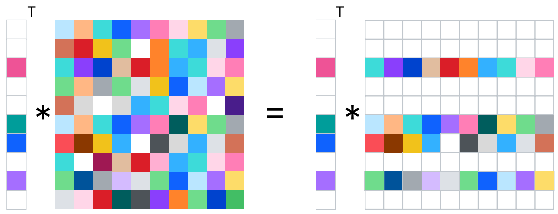
Another line of work is MoEfication of networks that have sparse activations by subdividing neurons [79]. Relufication can also help MoEfication be applicable to a wider range of networks by increasing sparsity of FFNs. For a more in depth review of mixture of expert models we refer the reader to [17].

Speculative Decoding and Sparsity. Speculative decoding is a method that aims to improve model latency when faced with memory bandwidth constraints [44, 40]. It involves using a smaller model to predict the next tokens, with a larger model subsequently verifying multiple tokens in a single operation. In this work, we examine the direct effects of incorporating sparsity into speculative decoding. We show that adding sparsity can lead to performance improvements in speculative decoding. Additionally, we provide guidelines on selecting parameters for speculative decoding when sparsity is introduced.

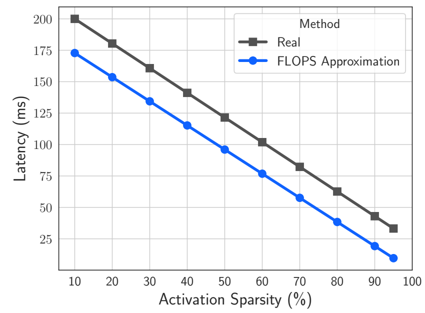
C Discussion on Activation Sparsity and Inference Efficiency

The primary motivation behind our work is to enhance *efficiency*, and we believe it is essential to provide a precise definition of this term. Throughout the main text, we predominantly use FLOPS as our efficiency metric. In this section, we argue why, in the presence of *activation sparsity*, FLOPS can serve as a suitable proxy for measuring various efficiency metrics.

Firstly, it is important to be reminded of the two major factors contributing to the efficiency of Large Language Models (LLMs): (1) the total amount of computation and (2) input/output (IO) transfer—i.e., transferring parameters from RAM to CPU/GPU for calculations. Notably, for today's large models, factor (2) acts as the major bottleneck during the inference phase. We refer the reader to the detailed analysis by Liu et al. [49]. Ultimately, for a specific target device and assuming an efficient implementation, we believe that the most practical measure of efficiency is *latency* (e.g., the average time to generate a token). However, each device possesses its unique properties, necessitating a more ubiquitous proxy metric.



(a) sparse vector, dense matrix multiplication: by skipping rows, we reduce both the weight transfer (i.e., loading these rows for computation) and computation (i.e., the result will be zero).



(b) Comparing FLOPS versus real latency for the OPT model (FFN).

Figure 5: For LLMs that have sparse activations, FLOPS is a good approximation of the real latency.

We argue that the way we calculated *FLOPS* in our paper and is greatly influenced by *activation sparsity* can reasonably approximate *efficiency*. Here are our reasons:

- **Reduced Computation**: As shown in Fig. 5a, with activation sparsity, we have a sparse vector-dense matrix multiplication at inference, while this will be a sparse-matrix-dense matrix multiplication during training. It is important to note that this is a semi-structured sparsity (unlike magnitude weight pruning), and we can leverage the fact that we are transferring weights in large chunks (i.e., *rows*). Modern accelerators already support sparse operations[3] and we can build on these existing tools.
- **Reduced IO Transfer**: During inference, weights need to be transferred to the device cache for computation (e.g., from RAM to CPU cache or GPU VRAM to GPU cache). This step constitutes the main bottleneck during token generation. For instance, approximately 99.3% of the total latency is attributed to IO, as indicated by Liu et al. [49]. However, by storing matrices in a row-major order, we can *skip loading* unnecessary rows as the output will be zero.

Overall, as depicted in Figure 5b based on the calculations by Liu et al. [49], we demonstrate that for the OPT model on an NVIDIA A100 GPU node, counting FLOPS provides a reasonable approximation to and is highly correlated with the time needs to generate tokens, especially, for LLMs with activation sparsity.

D Extended Relufication: Pushing for More Sparsity

In Sec. 3, we replaced non-ReLU activations to gain more sparsity. This leads to the input of *down projection* layer being sparse, roughly 30% of the total computation. However, there are other matrix-vector multiplications in the decoder layer of transformers besides the down projection. For instance, before the *up projection* and *gate projections* of FFN layer, and *QKV projections* in the attention layer (see Fig. 6). Together, the mentioned matrix-vector multiplications consume about 55% of the total computation.

To this end, we utilize the fact that in modern transformer layers, the input to both the attention and FFN layers come from a normalization layer, e.g., LayerNorm [4] or RMSNorm [76]. These layers can be viewed as a specific form of MLP, where, instead of applying arbitrary learnable parameters, they learn to scale inputs. Therefore, we apply ReLU to obtain sparse activations after normalization layers which we call the *second stage* of relufication in Fig. 6, continuing the *first stage* of relufication we performed in Sec. 3.

Tab. 2 shows that different stages of the relufication process do not significantly reduce zero-shot accuracy while using significantly less compute. The sparsity is broken down into three categories:

[3]For example, both cuSPARSE on NVIDIA CUDA® and Accelerate on Apple devices.

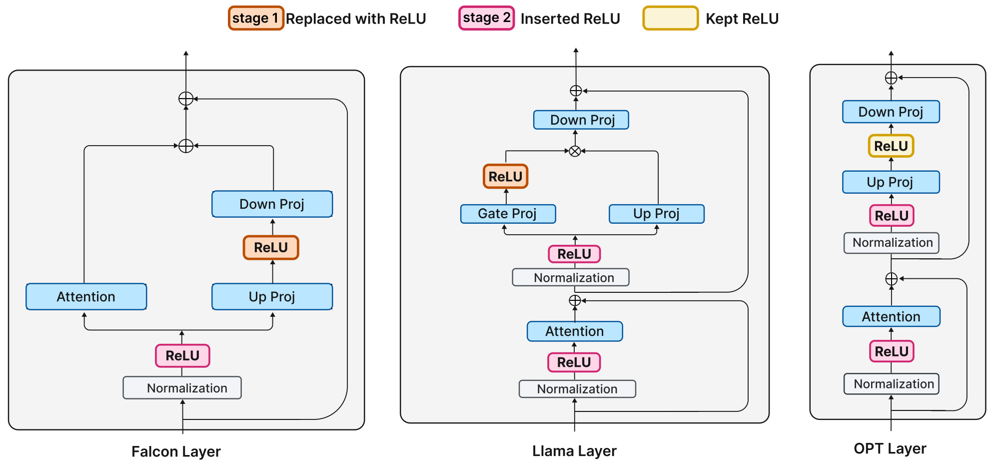


Figure 6: Architectural surgeries for *relufication*. In stage 1 (c.f., Sec. 3), we keep the existing ReLUs (in the case of OPT) or replace the activation function between up projection and down projections from GELU (Falcon) and SiLU (Llama) to ReLU. In stage 2, we insert new ReLUs after normalization layers.

Table 2: Comparing zero-shot performance across several tasks: After *relufication*, the activation sparsity of models increases significantly, hence increased efficiency measured by FLOPS. Within each group, the performance levels are comparable.

Model (stage)	Input Sparsity (%)			FLOPS (G)	Zero-Shot Accuracy (%)										
	QKV	DownProj	UpProj		Avg	Arc-E	Arc-C	Hellaswag	BoolQ	PIQA	LAMBADA	TriviaQA	WinoGrande	SciQ
OPT 1.3B	0	96	0	1.3	50.7	57.3	22.9	41.3	57.0	71.8	56.0	6.1	58.9	84.6
OPT 2.7B (s2)	50	96	35	1.1	53.1	60.3	26.8	44.9	55.4	73.9	57.6	12.4	59.6	86.7
OPT 2.7B	0	96	0	1.8	54.5	63.3	29.2	45.8	57.6	74.2	61.4	12.3	60.8	85.9
OPT 6.7B (s2)	50	97	40	2.8	58.6	66.5	32.2	49.1	63.0	76.4	63.3	23.8	63.1	90.3
OPT 6.7B	0	97	0	4.5	59.8	68.0	32.4	50.2	68.4	75.8	67.2	20.9	65.3	90.2
Falcon 7B (s2)	56	95	56	2.2	64.8	73.6	38.6	55.3	68.4	78.9	67.6	40.4	67.1	93.4
Falcon 7B (s1)	0	94	0	4.1	65.2	72.2	39.1	55.4	70.6	78.4	69.2	40.5	67.5	93.1
Falcon 7B	0	1	0	6.6	66.8	74.6	40.2	57.7	73.5	79.4	74.5	40.4	67.2	94.0
Llama 7B (s2)	51	65	67	2.9	66.4	73.8	39.6	54.8	69.9	77.9	70.7	48.5	68.6	93.8
Llama 7B (s1)	0	62	0	4.8	67.1	75.2	40.1	55.2	73.4	77.7	71.5	49.6	67.1	94.2
Llama 7B	0	0	0	6.6	68.4	75.5	42.1	69.9	74.8	78.7	73.1	49.9	69.8	95.4

up, down, and QKV projections. Notably, the input to QKV is less sparse than FFN projections, which opens an interesting avenue for future research. We note that the small gap in performance between the original vs. relufied models may be partially due to the finetuning process and not necessarily the activation function. Our finetuning is applied only for 30B and 50B tokens for stages 1 and 2, respectively. Putting into prospect and comparing it with 1T tokens of Llama, for example, this is equivalent to 3-5% of the original training duration. As discussed in Sec. 2.1, according to the scaling properties of LLMs, the gap will be further bridged by additional finetuning steps.

We also assess the in-context learning ability of the relufied models with the Massive Multitask Language Understanding (MMLU) [28] benchmark in Tab. 3. Our results show that when we augment the original LLMs with different activations and finetune, the few-shot performance does not change significantly either. Moreover, in Sec. F.2 we show that a larger but relufied model outperforms the non-relufied smaller model of the same FLOPS. Overall, the results affirm that the proposed relufication procedure can decrease the inference FLOPS at various stages and rates while maintaining on-par performance on various tasks.

E Applications

In this section, we discuss promising directions motivated by our investigation in Sec. 3. First, we introduce *aggregated sparsity*, showing that ReLU networks reuse previously activated neurons when generating tokens. Hence, we can leverage this to increase the generation speed. Next, we relate aggregated sparsity with speculative decoding to further improve speculative decoding's inference

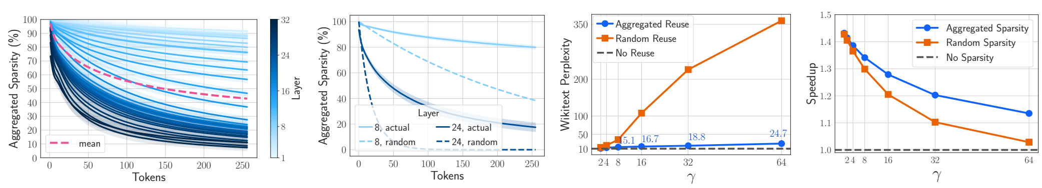
Table 3: MMLU five-shot accuracy. Models finetuned with different activation functions have similar performance.* Denotes we replace the SiLU function in Llama's SwiGLU activation function with ReLU.

Model	Activation	FLOPS(%)	Avg	Humanities	STEM	Social Sciences	Other
Falcon 7B	SiLU	100	26.4	24.8	27.4	27.2	26.2
Falcon 7B	GELU	100	27.7	28.1	26.0	28.0	29.4
Falcon 7B	ReLU	62	27.9	26.0	26.5	31.8	27.9
Llama 7B	SiLU*	100	35.1	37.9	30.2	37	37.1
Llama 7B	GELU	100	35.9	38.4	29.4	37.6	39.5
Llama 7B	ReLU	72	34.7	34.8	31.2	36.3	37.8

time. Finally, we briefly discuss a promising direction of using the *shifted ReLU* activation function to improve the sparsity further.

E.1 Aggregated Sparsity: Reusing Previously Activated Neurons

A consequence of using only a small subset of neurons for each token is that if these neurons are shared to some degree, the model still does not use all of the neurons until many tokens are processed. We refer to this as *aggregated sparsity*, which we defined as the ratio of neurons that have not been used up to processing the first t token. Note that this metric will always be non-increasing. Intuitively, it measures the unused capacity of feed-forward neurons for processing a specific prompt.



(a) (b) (c) (d)

Figure 7: **(a)** Aggregated sparsity of different layers and their mean. **(b)** Aggregated sparsity during token generation and comparison with a random sparsity. **(c)** Perplexity, based on the number of tokens for which loaded weights from previous tokens are reused. The dashed line represents no reuse, the solid blue line shows the case with activation reuse according to aggregated sparsity, and the orange line depicts the perplexity when activations are reused according to a random sparsity. **(d)** The inference speedup of speculative decoding with aggregated sparsity and with random sparsity. Speedup equal to 1.0 is the standard version of speculative decoding.

Here in Fig. 7a we show that for the OPT-6.7B model, on average, about 50% of all the neurons will be unused across the first 150 tokens of prompts coming from the WikiText dataset. Our empirical results hold for other ReLU models and other datasets. Additionally, in Fig. 7b, we show that this pattern is far from random activation of neurons during the token generation with a rate equal to the average rate of activation usage per token. Let s_i be the activation sparsity of layer i averaged over all tokens. Then, the probability of an activation not used in generating the first t tokens in uniformly random selection is s_i^t . Fig. 7b shows this quantity for two layers $i = 8, 24$ for the first 256 tokens in dashed line. It also shows the real (observed) number of activations being used in the solid line. The fact that the random aggregated sparsity (referred to as random sparsity) is lower than the observed aggregated sparsity (we refer to it as aggregated sparsity) shows a clear pattern of reusing activations.

We can benefit from the overlapping activations by utilizing previously loaded weights from the down projection layer for upcoming tokens. To test this, we initiate with reading 128 tokens. For the subsequent 128 tokens, we intermittently avoid loading new weights for every γ token. Using $\gamma = 16$ as an example, tokens 129-145 are generated conventionally. However, for tokens 146-161, we retain the existing weight without introducing any new weight. This pattern continues, with every next set of γ tokens alternating between conventional generation and weight reuse. In Fig. 7c, we observe only a slight increase in perplexity when using this approximation to address the memory and I/O-intensive nature of LLM inference. This figure contrasts the perplexity obtained from reused activations and random selections. The reuse strategy aligns well with the baseline, whereas random selection notably increases perplexity, highlighting the effectiveness of reusing the already loaded activations for subsequent tokens.

E.2 Activation Sparsity and Speculative Decoding

As highlighted in Sec. E.1, activation reuse happens for multiple consecutive tokens. When multiple consecutive tokens are processed together, we can save the I/O (i.e., transferring weights to GPU/CPU as discussed in Appendix C) associated with activations that are not used in any of them. If the reuse was not happening, and the sparsity of all tokens was purely random, the aggregated sparsity would shrink exponentially and quickly diminish. Speculative decoding [44] is a related technique that uses a smaller model M_q to propose γ tokens and a larger model M_p to verify those tokens and select matching ones. It improves the runtime of the model by avoiding running M_p sequentially.

Theoretical latency improvement. To improve speculative decoding, aggregated sparsity can trim down the portion of the model that needs to be run. Instead of running the full model, only the non-sparse parts need to be evaluated, which will reduce I/O and compute latency. Suppose the average aggregated sparsity of M_p for γ tokens is $\bar{s}_{\text{agg}}(\gamma)$, and cost of running M_q over M_p is c. As per the primary text, token acceptances are assumed to follow an independent and identically distributed (i.i.d.) behavior. Denote α as the expected probability of a token generated by M_q being approved by M_p. The following theorems hold:

Theorem 1. *The expected improvement factor in latency for speculative decoding with sparsity, over standard speculative decoding is* $\frac{c\gamma+1}{c\gamma+(1-\bar{s}_{agg}(\gamma))}$.

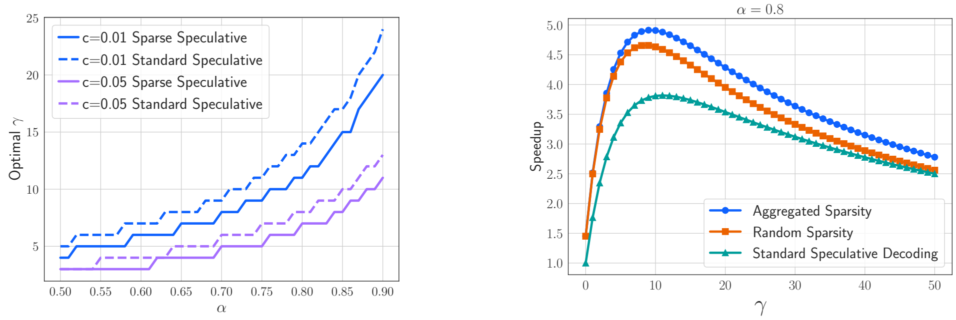
Proof. The amount of time required to run sparsified model is quantified as $Tc\gamma + (1 - \bar{s}_{\text{agg}}(\gamma))T$. It is the time of running a smaller model plus a larger model's non-sparse portion. Run time of speculative decoding without sparsity is $Tc\gamma + T$. The number of generated tokens in both is the same. Therefore the relative speedup of using sparsity is given by: $\frac{Tc\gamma+T}{Tc\gamma+(1-\bar{s}_{\text{agg}}(\gamma))T}$. □

Theorem 2. *The expected improvement factor in latency, when combining sparsity with speculative decoding against normal (autoregressive) decoding using only M_p, is* $\frac{1-\alpha^{\gamma+1}}{(c\gamma+(1-\bar{s}_{agg}(\gamma)))(1-\alpha)}$.

Proof. Similar to theorem above $Tc\gamma + (1 - \bar{s}_{\text{agg}}(\gamma))T$ gives the time required for sparse speculative decoding. According to the original paper, the standard speculative decoding yields an average of $\frac{1-\alpha^{\gamma+1}}{1-\alpha}$ tokens generated per each run [44]. Thus, the anticipated run time when generating tokens with sparsity during speculative decoding becomes $\frac{(c\gamma+(1-\bar{s}_{\text{agg}}(\gamma)))(1-\alpha)}{1-\alpha^{\gamma+1}}T$. Given the runtime for producing a single token via an autoregressive approach is T, the inverse of this fraction gives the desired results. □

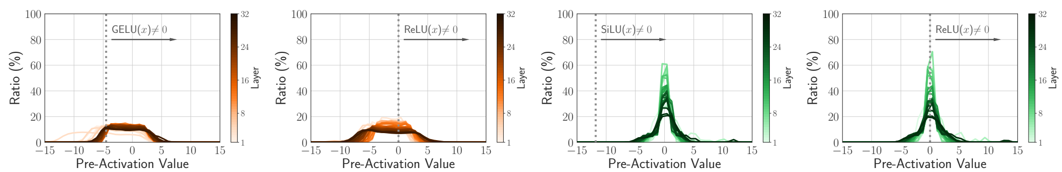
Fig. 7d compares sparse speculative decoding to the standard version for OPT 6.7B model. As a case study, for $\gamma = 16$, the sparse version has a 1.27x speedup over the standard speculative decoding. If the aggregated sparsity was random over different tokens, the speedup would have been only 1.20x. Note that even random sparsity will lead to speedup over standard speculative decoding. This further shows the value of relufication. However, the speedup due to random sparsity would diminish quickly in comparison to aggregated sparsity as we go for larger γ. For example, for $\gamma = 64$ the speedup is almost negligible, while the speedup for the aggregated sparsity is around 1.14x.

Optimal γ. The optimal γ for speculative decoding can be found by optimizing the speedup factor equation in Theorem 2. When sparsity is not present, the equation can be solved numerically, but for reluified networks, the aggregated sparsity for different γ's will affect the final results. We have found optimal γs based on $\bar{s}_{\text{agg}}(\gamma)$ for OPT 6.7B and presented the results in figure 8a. The chosen γ for sparse speculative decoding is smaller than standard speculative decoding since higher γ will result in less sparsity. The gap in γ is always less than 20%. Also, in figure 8b, it can be seen for the specific case of $\alpha = 0.8, c = 0.02$, the sparse speculative decoding has the highest speed-up factor over autoregressive at $\gamma = 10$s vs standard version's optimal point which happens for $\gamma = 12$. Sparse speculative decoding at $\gamma = 12$ is better than standard speculative decoding at $\gamma = 12$, while sparse speculative decoding at $\gamma = 10$ beats both. Another observation from 8b is for the case of purely random sparsity, the benefit of sparse speculative decoding would diminish over standard speculative decoding in higher γs. In contrast, the benefits of aggregated sparsity would last for larger values of γ.



(a) (b)

Figure 8: (a) optimal γ for sparse speculative decoding (b) speed up of sparse speculative decoding and standard speculative decoding over autoregressive decoding when $\alpha = 0.8$ and $c = 0.02$



(a) Falcon 7B (GELU) (b) Falcon 7B (ReLU) (c) Llama 7B (SiLU) (d) Llama 7B (ReLU)

Figure 9: The preactivation distribution of pretrained models for Falcon and Llama does not change significantly during the short finetuning stage of relufication. The dashed line shows the cutoff point before which the output is almost zero.

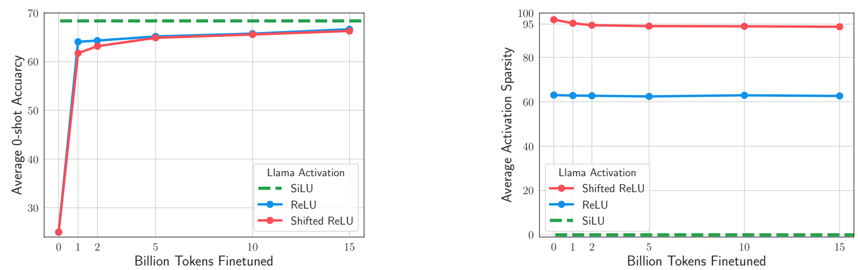
E.3 The Shifted ReLU Activation

Our work in this section is motivated by the observation from Sec. 3, where, comparing Fig. 3d with Fig. 3b revealed that the relufied Llama has much less sparsity (65%) than the relufied Falcon model (95%). In addition, we build on two of our previous findings. First, the preactivation distribution of the relufied Llama (Fig. 9c) includes a considerable mass after the cutoff value at zero. Second, the shape of the preactivation distribution does not change before and after the relufication process (Fig. 9c and Fig. 9d).

Therefore, we may be able to shift the preactivation distribution to the left to put more volume before the cutoff at 0. To this end, for preactivation input x, rather than applying $\text{ReLU}(x)$, we use $\text{ReLU}(x - b)$ where $b \in \mathbb{R}$ is a constant scalar. We propose to set the value b based on the preactivation distribution. For instance, based on the distribution in Fig. 9d, setting $b = 1$ and hence using $\text{ReLU}(x - 1)$ as our activation function will result in dropping 95% of the preactivations and make it significantly sparser. Another benefit of this approach is simplicity, as this does not require changing the loss function or the training regime.

Figure 10a shows that the shifted ReLU activation function has on-par accuracy with the ReLU activation function. Moreover, similar to our observation in Sec. 3, the shifted ReLU activation quickly recovers the lost performance due to the drastic change of activation function, while it also maintains a very high-level activation sparsity during the finetuning stage. The gap between shifted ReLU and ReLU is wider in the early stages of training, and it narrows down when more tokens are seen.

A deeper investigation of ReLU-variants that can promote sparsity without sacrificing performance is an appealing future direction. Moreover, it will be interesting to study the impact of the shifted ReLU for stage-2 of our relufication process where the sparsity level is usually not very high.



(a) (b)

Figure 10: The effect of shifted ReLU on Llama model. **(a)** The performance is almost the same as the original ReLU. **(b)** Shifted ReLU (i.e., $\text{ReLU}(x-1)$) is much sparser than the original ReLU.

F Additional Results

F.1 Pre-activation distribution of OPT models trained from scratch

The distribution of pre-activation inputs is suspected to be the primary factor determining the level of sparsity. As demonstrated in Sec. 3, the pre-activation distribution differs significantly for Llama and Falcon. One might question whether, when controlling for training data and optimization algorithms, the distribution shapes would vary. To investigate this, we trained OPT 1.3B models from scratch using our four activation function variants and illustrated the pre-activation distribution throughout training in Fig. 11. The distributions begin similarly but gradually diverge. Moving from SiLU to ReLU (increasing β), the pre-activation distribution concentrates more around 0, almost forming a unimodal shape. Further exploration of the dynamics of pre- and post-activations and their correlation with efficiency and accuracy is an intriguing avenue for future research.

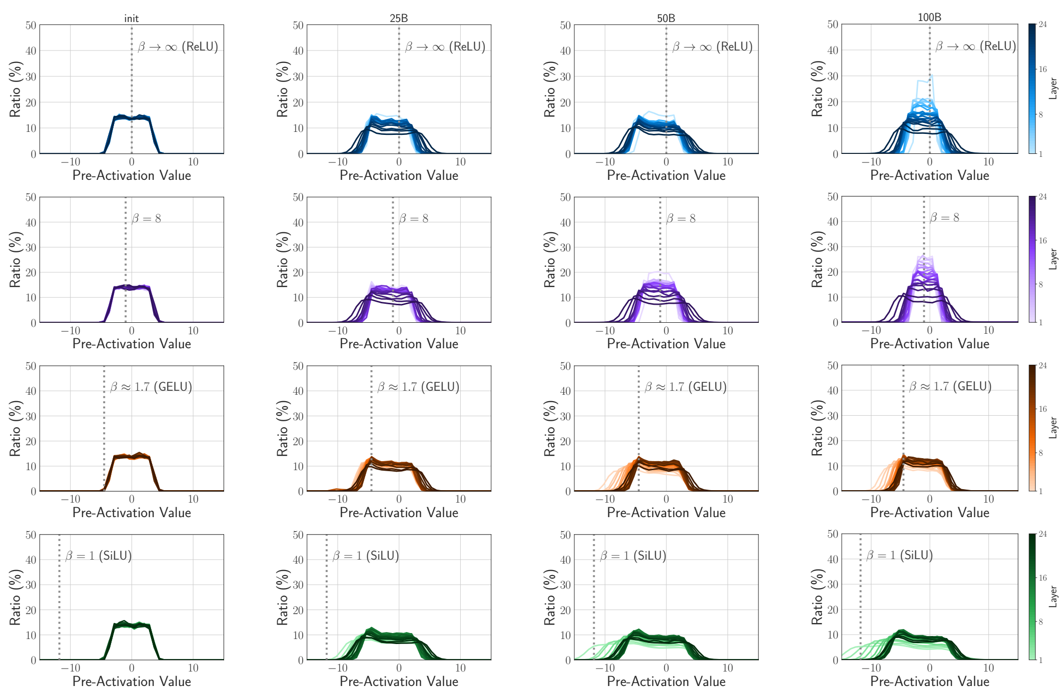


Figure 11: Pre-activation distributions of various OPT 1.3B models with all four types of activations trained from scratch at various number of seen tokens during training.

F.2 Is a large relufied model better than smaller dense ones?

When it comes to deploying efficient models, one may naturally use an original smaller-size (dense) model. The argument would be the performance of the relufied larger model might be already equal to or less than the smaller dense model. To study the above question, we plotted the performance vs. efficiency of the original and the relufied OPT models in Fig. 12. Taking the relufied OPT 6.7B model as an example, it operates at 2.8 GFLPOPs per token. Interpolating the blue line (that can be seen as a scaling plot of the OPT model), a dense model with equivalent FLOPS falls more than 2% short in zero-shot performance.

Similarly, compared to the relufied OPT 2.7B model, the equivalent (in FLOPS) dense model performs almost 2% lower. Indeed, the fact that the relufied models lie well above the scaling graph of the original OPT models, shows the effectiveness of relufication processes as a method to get better but more efficient models. As a side benefit, it makes the efficiency spectrum of the available LLMs more continuous. For example, consider a combination of hardware and use case that only allows deploying LLMs with lower than 3 GFLOPS during inference. Going with standard pretrained models, the only available option is OPT 2.7B with almost 1 GFLOPS, as the 6.7B does not satisfy the hardware constraint. In this situation, our relufied model not only falls in the limited inference budget but is also very close to the next largest available model in terms of accraucy. An exciting and timely direction for future research is finding methods, that, given an LLM (or a family of LLMs), are able to produce the best performing model matching the specified inference computation budget.

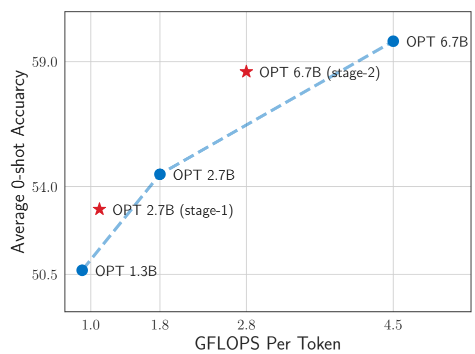


Figure 12: Performance of sparse large models vs. dense smaller models: The relufied large models (red stars) are above the scaling curve of original dense models (blue circles and dashed line).