ProSparse: Introducing and Enhancing Intrinsic Activation Sparsity within Large Language Models

Chenyang Song[1], **Xu Han**[1*], **Zhengyan Zhang**[1], **Shengding Hu**[1], **Xiyu Shi**[2]
Kuai Li[3], **Chen Chen**[3], **Zhiyuan Liu**[1*], **Guangli Li**[2], **Tao Yang**[3], **Maosong Sun**[1]
[1] Dept. of Comp. Sci. & Tech., Institute for AI, Tsinghua University, Beijing, China
[2] SKLP, Institute of Computing Technology, Chinese Academy of Sciences, China
[3] Tencent Machine Learning Platform, China
scy22@mails.tsinghua.edu.cn, {han-xu,liuzy}@tsinghua.edu.cn

Abstract

Activation sparsity refers to the existence of considerable weakly-contributed elements among activation outputs, serving as a promising paradigm for accelerating model inference. Nevertheless, most large language models (LLMs) adopt activation functions without intrinsic activation sparsity (e.g., GELU and Swish). Some recent efforts have explored introducing ReLU or its variants as the substitutive activation function to pursue activation sparsity and acceleration, but few can simultaneously obtain high activation sparsity and comparable model performance. This paper introduces a simple and effective method named "ProSparse" to sparsify LLMs while achieving both targets. Specifically, after introducing ReLU activation, ProSparse adopts progressive sparsity regularization with a factor smoothly increasing for multiple stages. This can enhance activation sparsity and mitigate performance degradation by avoiding radical shifts in activation distributions. With ProSparse, we obtain high sparsity of 89.32% for LLaMA2-7B, 88.80% for LLaMA2-13B, and 87.89% for end-size MiniCPM-1B, respectively, with comparable performance to their original Swish-activated versions. These present the most sparsely activated models among open-source LLaMA versions and competitive end-size models. Inference acceleration experiments further demonstrate the significant practical acceleration potential of LLMs with higher activation sparsity, obtaining up to 4.52× inference speedup.

1 Introduction

Recent years have witnessed significant breakthroughs made by large language models (LLMs) with commendable performance across a wide range of NLP tasks (Brown et al., 2020; Wei et al., 2021; Ouyang et al., 2022; OpenAI, 2023; Achiam et al., 2023). Nevertheless, the formidable computational costs required by the deployment and inference of LLMs pose a considerable challenge (Aminabadi et al., 2022; Pope et al., 2023). The utilization of activation sparsity is one of the most promising techniques to enhance inference efficiency (Liu et al., 2023; Song et al., 2023), by discarding the redundant computation associated with the elements among LLM activation outputs that contribute weakly to the final outputs.

The adoption of ReLU, which naturally outputs zero elements, as the activation function is a straightforward method to achieve intrinsic activation sparsity in early LLMs (Raffel et al., 2020; Zhang et al., 2022a). However, recent LLMs predominantly favor non-ReLU activation functions, such as GELU and Swish (Touvron et al., 2023a; Chowdhery et al., 2023; Almazrouei et al., 2023). Although these non-ReLU LLMs may also display activation sparsity (Zhang et al., 2024), such sparsity is manually imposed by searching adaptive activation thresholds (i.e., non-intrinsic), which can potentially lose minor neuron outputs and degrade performance. To pursue the sparsity-based inference acceleration, the task of ReLUfication is proposed, aiming to introduce ReLU-based intrinsic activation sparsity into non-ReLU LLMs. Preliminary methods (Zhang et al., 2022b, 2024) directly substitute the activation functions with ReLU. As such substitution cannot overcome the inherent limitation imposed by the original dense activation distribution, inserted and shifted ReLU functions (Mirzadeh et al., 2023) are introduced to enforce higher sparsity through radically shifting the activation distribution. However, existing efforts fail to achieve satisfactory sparsity and risk performance degradation.

In this paper, we propose a simple and effective ReLUfication method named "**ProSparse**" to help non-ReLU LLMs obtain high activation sparsity

*The corresponding authors of this paper: Xu Han (han-xu@tsinghua.edu.cn) and Zhiyuan Liu (liuzy@tsinghua.edu.cn).

Proceedings of the 31st International Conference on Computational Linguistics, pages 2626–2644
January 19–24, 2025.

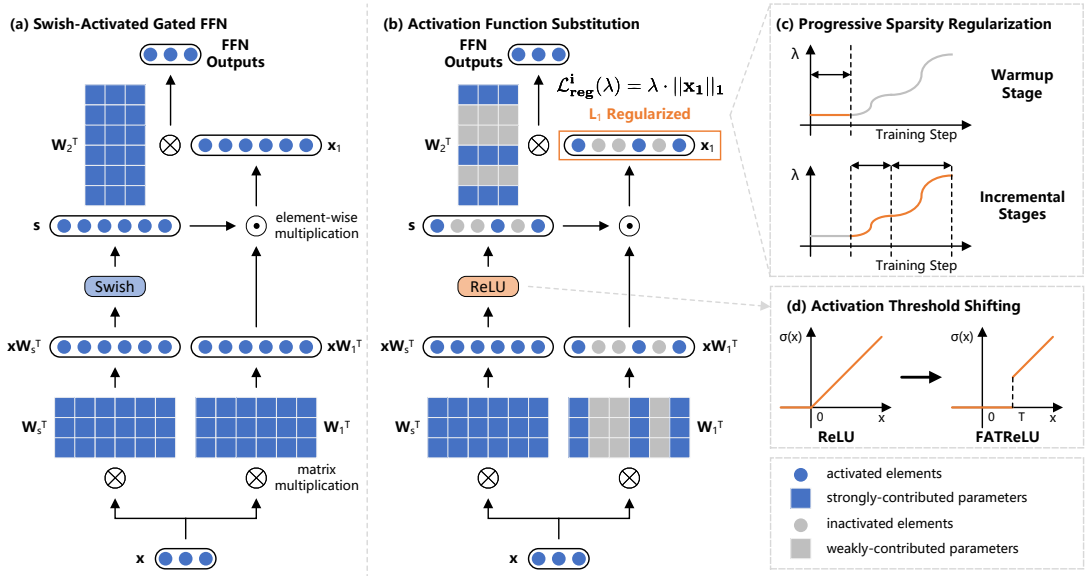

Figure 1: The overall architecture of ProSparse, which includes three steps: activation function substitution, progressive sparsity regularization, and activation threshold shifting.

without performance degradation. ProSparse includes three steps shown in Figure 1: activation function substitution, progressive sparsity regularization, and activation threshold shifting. The first step is to replace the activation function with ReLU and then apply continual training. As discussed above, this can hardly achieve satisfactory sparsity. Therefore, in the second step, we apply sparsity regularization (Ma et al., 2019) to the intermediate activation outputs of the feed-forward networks (FFNs) within LLMs to seek higher sparsity. Considering the potential performance risks of forcing the fixed regularization factor (Ma et al., 2019; Li et al., 2020), we progressively increase the factor in multiple stages, including one flat warmup stage and multiple incremental stages along gentle sine curves. Such progressive regularization can provide more time for adaption to increasing regularization and avoid a radical shift in activation distribution, thereby mitigating performance degradation. The final step adopts FATReLU (Kurtz et al., 2020), shifting the ReLU activation threshold to a positive value. This prunes less influential neurons to improve sparsity.

In experiments, we apply ProSparse to the ReLUfication of LLaMA2 (Touvron et al., 2023b) and end-size MiniCPM (Hu et al., 2024). Activation sparsity of 89.32%, 88.80%, and 87.89% are successfully achieved for LLaMA2-7B, LLaMA2-13B, and MiniCPM-1B, respectively, with performance comparable to their original Swish-activated versions on various LLM benchmarks. Furthermore, we demonstrate the practical inference acceleration of higher activation sparsity, by respectively applying an approximate algorithm and an accurate algorithm to the inference of models with different sparsity. For the approximate one, we use PowerInfer (Song et al., 2023), which achieves state-of-the-art speedup ratios tailored for sparsely activated LLMs but risks inaccurate inference due to the mistakes of activation predictors. For the accurate one, we implement and release two GPU operators that leverage the input-side and output-side sparsity during the computation of ReLU-activated FFNs*.

The experimental results demonstrate that models with higher sparsity can achieve more significant inference acceleration with both approximate and accurate algorithms (e.g., up to 4.52× speedup with PowerInfer). Moreover, comprehensive analyses are conducted to figure out the quantitative relationship between the activation sparsity and the regularization factor, making the activation sparsity obtained by ProSparse more controllable. We also discuss the rationality of progressive L_1 regularization, empirical methods of performing supervised fine-tuning (SFT) on sparsely activated models, and the sparsity distribution among distinct datasets or layers.

In summary, we make the following contributions in this paper: (1) We propose ProSparse, an effective ReLUfication method that converts non-ReLU LLMs into much sparser ReLU-

*Source codes for these two operators are available at https://github.com/Raincleared-Song/sparse_gpu_operator.

activated LLMs without performance degradation. (2) Sparsely activated versions of LLaMA2-7B, LLaMA2-13B, and MiniCPM-1B comparable to their original Swish-activated versions in performance are both obtained and available[†]. (3) We demonstrate the practical inference acceleration effect of higher activation sparsity that ProSparse can reach. Valuable observations and analyses are also conducted.

2 Preliminaries and Related Works

Here we discuss how to improve LLM inference efficiency. Refer to existing surveys (Zhao et al., 2023) for works about LLMs and Appendix A for works about L_1 regularization.

Inference Acceleration for LLMs Efficiency has long been a crucial topic in various AI applications (Chen et al., 2023b). The sustainable increase in LLM scales brings the exponential growth of inference computations, making the deployment of LLMs a formidable challenge (Kaplan et al., 2020; Liu et al., 2023). To reduce the computational costs required by LLM inference, various model compression or decoding acceleration methods have been proposed, such as quantization (Jacob et al., 2018; Nagel et al., 2019; Zhao et al., 2019; Bai et al., 2022; Xiao et al., 2023; Yao et al., 2023), pruning (Hoefler et al., 2021; Ma et al., 2023; Sun et al., 2023; Frantar and Alistarh, 2023; Xia et al., 2023), distillation (Tang et al., 2019; Touvron et al., 2021; Gu et al., 2023; Hsieh et al., 2023), and efficient sampling methods (Leviathan et al., 2023; Wang et al., 2023; Chen et al., 2023a; Miao et al., 2023). While these works have proved effective for inference acceleration and other scenarios (e.g., secure federated learning (Ding et al., 2023b)), none of these methods leverages the intrinsic mechanisms within LLMs.

Activation Sparsity Recent works (Li et al., 2022; Liu et al., 2023; Song et al., 2023) have noticed the intrinsic activation sparsity within some LLMs and its potential in inference acceleration. Activation sparsity refers to the phenomenon where considerable zero or negligible elements in activation outputs, corresponding to certain model parameters (i.e., neurons), have a weak impact on LLM outputs given a specific input. These weakly-contributed parameters are regarded as inactivated and can thus be skipped during inference to save computational resources. Notably, **the utilization of activation sparsity is orthogonal to model compression and efficient sampling, and these approaches can be readily combined**. Another fact worth attention is **the fundamental difference between activation sparsity and pruning**, see Appendix A.

ReLUfication Activation sparsity naturally exists in ReLU-activated architecture (Li et al., 2022), from LLMs (Raffel et al., 2020; Zhang et al., 2022a) to vision models (Dosovitskiy et al., 2020). However, recent LLMs such as Falcon (Almazrouei et al., 2023) and LLaMA (Touvron et al., 2023b) prevalently adopt non-ReLU activation functions such as GELU (Hendrycks and Gimpel, 2016) and Swish (Elfwing et al., 2018) without intrinsic activation sparsity. Therefore, to leverage the merits of activation sparsity without training a ReLU-activated LLM from scratch, many works conduct ReLUfication, introducing sparse ReLU-based activations into non-ReLU LLMs. Zhang et al. (2022b) converts a GELU-activated BERT (Devlin et al., 2018) into a ReLU-activated version through activation function substitution and additional training. ReluLLaMA and ReluFalcon apply a similar paradigm to Falcon and LLaMA, respectively (Zhang et al., 2024). Since activation substitution cannot reach satisfactory sparsity, mainly due to the unhandled limitation of the original dense activation distribution, the inserted and shifted ReLU activation functions are introduced (Mirzadeh et al., 2023), conducting a radical shift in activation distribution. Although these operations are claimed to achieve sparsity of nearly 95%, we cannot replicate the results in our experiments (see the 3rd paragraph of Section 4.4) and the sparsity is still limited. By contrast, ProSparse is a ReLUfication method designed to achieve high sparsity and mitigate performance degradation concurrently.

3 Methods

3.1 Definitions and Notations

For the convenience of subsequent demonstrations, here we define activation sparsity in detail. Since the activation function mainly exists in the FFNs within LLMs, we first discuss the computation process of FFNs. Given the hidden dimension d_{model}

[†]Models are respectively available at https://huggingface.co/SparseLLM/prosparse-llama-2-7b, https://huggingface.co/SparseLLM/prosparse-llama-2-13b, and https://huggingface.co/SparseLLM/ProSparse-MiniCPM-1B-sft.

and the intermediate dimension d_{ff}, the computation process of a gated FFN (i.e., the most widely adopted FFN architecture in recent LLMs (Dauphin et al., 2017; Shazeer, 2020)) can be formalized as:

$$\begin{aligned} \mathbf{s} = \sigma(\mathbf{x}\mathbf{W}_s^T), \quad \mathbf{x}_1 = \mathbf{s} \odot (\mathbf{x}\mathbf{W}_1^T), \\ \text{FFN}(\mathbf{x}) = \mathbf{x}_1\mathbf{W}_2^T, \end{aligned} \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^{d_{model}}$, $\mathbf{s}, \mathbf{x}_1 \in \mathbb{R}^{d_{ff}}$, σ, and $\odot$ denote the input hidden states, the gating scores, the intermediate outputs, the activation function, and the element-wise multiplication respectively. $\mathbf{W}_s, \mathbf{W}_1 \in \mathbb{R}^{d_{ff} \times d_{model}}$ and $\mathbf{W}_2 \in \mathbb{R}^{d_{model} \times d_{ff}}$ are learnable weights.

We define the **activation sparsity** (hereinafter abbreviated as **sparsity**) as the ratio of zero elements (i.e., inactivated elements) in $\mathbf{x}_1$ for a specific input $\mathbf{x}$. The sparsity of an LLM is evaluated using the **average sparsity**, defined as the average value of sparsity across all layers in an LLM on a sufficiently large amount of input data.

In this paper, we focus on the task of ReLUfication, namely converting an LLM using a non-ReLU activation function σ (e.g., GELU or Swish) into a ReLU-activated one, while making the average sparsity as high as possible and mitigating performance degradation.

3.2 ProSparse

We propose ProSparse to achieve the above targets. Three steps are carefully designed to introduce and enhance the intrinsic activation sparsity for a non-ReLU LLM: (1) activation function substitution; (2) progressive sparsity regularization; (3) activation threshold shifting.

Activation Function Substitution For lack of attention to activation sparsity, a majority of recent mainstream LLMs adopt non-ReLU activation functions such as GELU and Swish that output few zero elements (i.e., low activation sparsity according to the above definition). Therefore, the first step of ProSparse is to introduce intrinsic sparsity through activation function substitution, which replaces the FFN activation function σ with ReLU, namely $\sigma(x) = \max(x, 0)$, followed by continual training. This can make the ratio of zero activation elements significantly larger and preliminarily adapt the LLM to new ReLU activation.

Progressive Sparsity Regularization Nevertheless, activation function substitution by nature does not change the activation distribution, which will potentially limit the sparsity to relatively low values. To push for higher sparsity, a typical method is L_1 sparsity regularization (Li et al., 2022), which introduces the L_1 regularization loss as an extra training target. Given the intermediate output $\mathbf{x}_1$ of the i-th FFN layer in an LLM, the regularization loss is defined as:

$$\mathcal{L}_{reg}^i(\lambda) = \lambda \cdot ||\mathbf{x}_1||_1, \tag{2}$$

where $||\cdot||_1$ is the L_1 norm operator and λ is the regularization factor. For an LLM with K FFN layers, the total regularization loss is summed from the losses of all layers, namely $\mathcal{L}_{reg}(\lambda) = \sum_{i=1}^{K} \mathcal{L}_{reg}^i(\lambda)$. The overall optimization target is $\mathcal{L}_{lm} + \mathcal{L}_{reg}(\lambda)$, where $\mathcal{L}_{lm}$ is the vanilla language modeling loss.

Considering the potential performance degradation due to fixed regularization factors (Georgiadis, 2019; Kurtz et al., 2020; Li et al., 2022), we propose the progressive sparsity regularization, where the factor λ is carefully scheduled to gently increase in multiple stages. Refer to Appendix B for more details.

Concretely, for the warmup stage, we set λ to a constant value, which is relatively small to prevent radical activation distribution shifts and introduce higher preliminary sparsity. Next, for each of the remaining stages (called incremental stages), λ is scheduled to increase along a smooth sine curve from a trough value to a peak value. Inspired by the cosine annealing scheduler for learning rates (Loshchilov and Hutter, 2016), we choose the sine function owing to its special trend, as the small derivatives near its troughs and peaks can make λ not increase radically around these two points. This provides the LLMs with more time to adapt the activation distributions to the newly increased L_1 regularization. Notably, each stage is accompanied by certain steps of training. The step numbers and peak λ values are chosen according to the target sparsity and stability.

Activation Threshold Shifting As demonstrated by recent works, there exist considerable amounts of non-zero low elements in the activation outputs, which have little influence on final results and thus can be pruned for higher sparsity (Zhang et al., 2024). Therefore, we convert the ReLU into FATReLU (Kurtz et al., 2020) by shifting the activation threshold, i.e.,

$$\sigma(x) = \begin{cases} x & \text{when } x \geq t, \\ 0 & \text{otherwise}, \end{cases} \tag{3}$$

where $t > 0$ is a positive threshold. As long as t is properly chosen (see Appendix O), FATReLU can increase sparsity with negligible losses (Zhang et al., 2024).

3.3 Practical Inference Acceleration

To go beyond theoretical analyses based on FLOPS (Floating Point Operations Per Second) (Mirzadeh et al., 2023) and establish the practical value of ProSparse, we discuss how to realize inference acceleration with sparsely activated LLMs on real hardware and how to evaluate the practical acceleration effects. We consider two categories of acceleration algorithms based on activation sparsity: approximate algorithms and accurate algorithms.

Approximate Acceleration Algorithms Utilizing activation sparsity, recent approximate acceleration algorithms predominantly rely on activation predictors, typically small neural networks, to forecast the activation distributions indicated by the sparse intermediate outputs $\mathbf{x}_1$ given a specific input $\mathbf{x}$ (Liu et al., 2023; Song et al., 2023). In this way, they can make wiser hardware allocation or computation policies to avoid resource waste on weakly-contributed parameters. However, their efficiency and accuracy largely depend on the predictors' performance, and invalid predictions can cause suboptimal hardware allocation or even inference inaccuracy. Therefore, to gain more practical acceleration effects from approximate algorithms, both high activation sparsity and predictability are indispensable.

To this end, we focus on three metrics for acceleration analysis: the activation recall, the predicted sparsity, and the inference speed. The former two metrics evaluate the performance of activation predictors as well as the activation predictability of a sparse LLM (Zhang et al., 2024). For inference speed, we adopt PowerInfer (Song et al., 2023), a state-of-the-art approximate algorithm to measure practical speedup ratios. Refer to Appendix C for more related introductions and the detailed approach to calculating these metrics.

Accurate Acceleration Algorithms Targeting acceleration without potential inference inaccuracies, we implement two hardware-efficient sparse GPU operators with system-level optimizations, such as operator fusion, coalesced memory access, and vectorization, thereby exploiting input-side and output-side sparsity in Equation 1.

Concretely, we reorganize a ReLU-activated gated FFN into three major steps and our two operators are responsible for the step (2) and (3) respectively: (1) A dense matrix-vector multiplication operator $\mathbf{x}\mathbf{W}_s^T$ directly supported by vendor libraries such as cuBLAS; (2) A fused operator of ReLU and $\mathbf{s} \odot (\mathbf{x}\mathbf{W}_1^T)$ with output-side sparsity; (3) A sparse matrix-vector multiplication operator $\mathbf{x}_1\mathbf{W}_2^T$ with input-side sparsity. We adopt the single-step speedup ratios of steps (2) and (3) with these two operators respectively to reflect the practical accurate acceleration potential of sparse LLMs. Refer to Appendix D for implementation details.

4 Experiments

4.1 Experimental Settings

Our training data consists of both language modeling datasets and instruction tuning datasets. For evaluation, we adopt comprehensive benchmarks covering code generation, commonsense reasoning, reading comprehension, and 4 other popular tasks. Refer to Appendix E for more details.

4.2 Overall Results

We apply ProSparse to Swish-activated LLaMA2-7B, LLaMA2-13B (Touvron et al., 2023b), and MiniCPM-1B (Hu et al., 2024). The obtained sparsely activated models are then compared with their original Swish-activated versions. For comprehensiveness, we also consider ReluLLaMA[‡], the only open-source ReLU-based LLMs fine-tuned from LLaMA2. All the average sparsity values are computed on the same mixed dataset sampled from training datasets. For more hyper-parameters, see Appendix L and O.

Results are shown in Table 1 (see Appendix G for performance on each independent benchmark). We can draw three conclusions:

(1) *Effectiveness*: ProSparse simultaneously achieves high sparsity and comparable downstream performance for all the three Swish-activated models considered. The activation sparsity obtained by ProSparse is significantly higher than ReluLLaMA, reaching the state-of-the-art level among all the open-source LLaMA versions and competitive end-size models.

(2) *Scale Generalizability*: The effectiveness of ProSparse consistently holds under three model scales. The promising results on the end-size

[‡] https://huggingface.co/SparseLLM/ReluLLaMA-7B

Setting	Code Generation	Commonsense Reasoning	Reading Comprehension	GSM8K	MMLU	BBH	AGI Eval	Average Performance	Average Sparsity
LLaMA2-7B	16.37	69.59	61.87	12.96	44.45	32.96	27.53	37.96	-
ReluLLaMA-7B	15.85	69.64	70.54	5.84	38.64	35.07	27.73	37.62	66.98
ProSparse-7B*	19.47	66.29	63.33	12.74	45.21	33.59	27.55	38.31	88.11
ProSparse-7B	19.42	66.27	63.50	12.13	45.48	34.99	27.46	**38.46**	**89.32**
LLaMA2-13B	20.19	72.58	71.55	22.21	54.69	37.89	29.33	44.06	-
ReluLLaMA-13B	20.19	70.44	73.29	18.50	50.58	37.97	28.22	42.74	71.56
ProSparse-13B*	29.03	69.75	67.54	25.40	54.78	40.20	28.76	**45.07**	87.97
ProSparse-13B	28.42	69.76	66.91	26.31	54.35	39.90	28.67	44.90	**88.80**
MiniCPM-1B	36.85	63.67	60.90	35.48	50.44	35.03	28.71	44.44	-
ProSparse-1B*	41.38	64.55	60.69	34.72	49.36	34.04	28.27	**44.72**	86.25
ProSparse-1B	42.04	64.37	60.73	34.57	49.51	34.08	27.77	**44.72**	**87.89**

Table 1: The overall experimental results with the comparison of activation sparsity (%) and downstream performance (%). "LLaMA2" and "MiniCPM" refer to the original Swish-activated LLaMA2 (Touvron et al., 2023b) and MiniCPM (Hu et al., 2024) versions respectively. "ProSparse-7B*", "ProSparse-13B*", and "ProSparse-1B*" denote the ProSparse versions **without activation threshold shifting**.

model (i.e., MiniCPM-1B) reveal the potential of ProSparse as well as activation sparsity on end-user devices, where the inference efficiency of LLMs is significantly emphasized.

(3) *Effect of Activation Threshold Shifting*: Based on the results without activation threshold shifting (i.e., those with the "*" marker), we can demonstrate the effectiveness of this technique in improving the sparsity without compromising performance. Notably, the threshold t must be carefully chosen, see Appendix O.

4.3 Acceleration Effect of Sparsity

Approximate Acceleration Algorithm In this section, we train the activation predictors for each sparse LLM and compute the recalls, predicted sparsity, and actual inference speeds on PowerInfer (Song et al., 2023). As the FFN in each Transformer layer has different activation distributions as well as different predictors, the former two metrics are averaged from the results of all layers. Note that MiniCPM-1B has not been tested since PowerInfer does not support its architecture at present. Refer to Appendix F for training details of predictors.

As demonstrated by the results shown in Table 2, compared with llama.cpp[§], an acceleration framework without sparsity utilization, PowerInfer achieves up to 4.52× speedup, revealing the significant potential of sparsity-based acceleration. Moreover, an increased activation sparsity can considerably improve the activation recall, the predicted sparsity, and the inference speed of PowerInfer. This proves the considerable practical values of even more sparsely activated LLMs in improving the inference speed with predictor-based approximate acceleration algorithms and mitigating the inaccurate inference problem. ProSparse, which reaches a high sparsity without performance degradation, can thus gain the most acceleration effects with PowerInfer.

Accurate Acceleration Algorithm Furthermore, with LLMs of different sparsity, we measure the average single-step wall-clock time spent by our two sparse GPU operators, which are responsible for step (2) and step (3) in Section 3.3 respectively. As demonstrated in Table 2, higher activation sparsity can make accurate algorithms based on GPU operators more efficient. Besides, our two sparse GPU operators also display satisfactory speedup ratios up to 2.44× and 1.70× respectively with better acceleration effects for larger models. Note that despite the less significant acceleration effects than PowerInfer, our GPU operators are highly pluggable, predictor-free, and immune to potential inference accuracies.

4.4 Analysis and Discussion

***Q1:* What is the effect of L_1 regularization and its trend of increase?** For this question, we consider two regularization-free ReLUfication baselines: vanilla ReLU (Zhang et al., 2024) and shifted ReLU (Mirzadeh et al., 2023). Both only include the substitution of activation functions with ReLU (i.e., $\max(x, 0)$) and shifted ReLU (i.e., $\max(x - b, 0)$, where b is a tunable bias) respectively.

First, we consider the training dynamics of the above two baselines and ProSparse, as shown in Figure 2. The setting "Fixed L_1" is a reference setting with a constant regularization factor. Clearly, the training stages with increasing sparsity only include those with regularization applied, namely the whole "Fixed L_1", the warmup stage, and the in-

[§] https://github.com/ggerganov/llama.cpp

Setting	Average Sparsity	Approximate Acceleration: Activation Recall	Approximate Acceleration: Predicted Sparsity	Approximate Acceleration: Inference Speed	Approximate Acceleration: Speedup to Dense	Accurate Acceleration: Step (2) Time (↓)	Accurate Acceleration: Speedup to Dense	Accurate Acceleration: Step (3) Time (↓)	Accurate Acceleration: Speedup to Dense
Dense-7B	-	-	-	3.67	1.00	90.55	1.00	82.92	1.00
ReluLLaMA-7B	66.98	90.89	58.95	11.37	3.10	67.12	1.35	63.00	1.32
ProSparse-7B*	88.11	**93.46**	75.24	**16.30**	**4.44**	46.66	1.94	55.56	1.49
ProSparse-7B	**89.32**	92.34	**78.75**	-	-	**45.38**	**2.00**	**55.05**	**1.51**
Dense-13B	-	-	-	1.92	1.00	131.36	1.00	113.68	1.00
ReluLLaMA-13B	71.56	86.41	71.93	6.59	3.43	69.92	1.88	75.47	1.51
ProSparse-13B*	87.97	91.02	77.93	**8.67**	**4.52**	55.29	2.38	67.50	1.68
ProSparse-13B	**88.80**	**91.11**	**78.28**	-	-	**53.78**	**2.44**	**66.73**	**1.70**

† For "Dense" settings, the "Inference Speed" is obtained by llama.cpp, and the time for steps (2) and (3) is measured without sparse GPU operators. For other sparse settings, the "Inference Speed" is obtained by PowerInfer, and sparse GPU operators are applied. ProSparse settings with activation threshold shifting and the MiniCPM architecture are not supported by PowerInfer at present.

Table 2: The comparison of activation recalls (%), predicted sparsity (%), inference speeds (tokens per second) by llama.cpp (Dense) or PowerInfer (others), and the average wall-clock time (us) without (Dense) or with (others) our sparse GPU operators among LLMs with different sparsity. "Step (2)" and "Step (3)" correspond to the steps in Section 3.3.

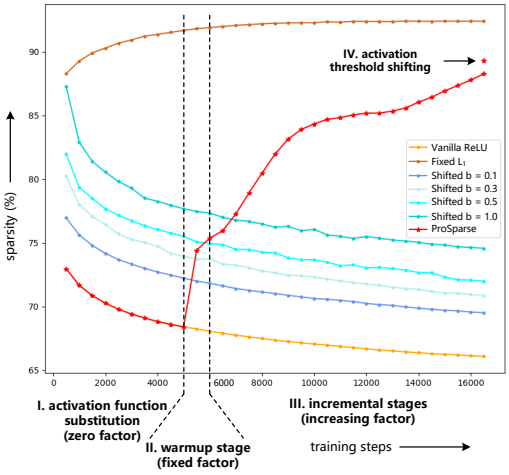

Figure 2: The trend of sparsity (7B models) along the training process. "Shifted" denotes Shited ReLU and $b = 0.1$ corresponds to the results in Table 4.

cremental stages of ProSparse. Therefore, **among the settings involved, the trend of sparsity is incremental only if non-zero L_1 regularization is applied**[¶]. **Neither vanilla ReLU nor shifted ReLU can push for higher sparsity without regulatization**.

However, concerns may naturally arise about the performance, as the additional L_1 loss term can unavoidably influence the optimization of the language modeling target. For this problem, we evaluate the above methods given different numbers of training tokens. Through experiments (see Appendix H), while a performance gap exists between ProSparse and two baselines given limited 34.60B tokens, it obtains comparable performance when sufficient 89.13B tokens are provided and thus the regularization can increase more smoothly, with a final sparsity value close to the limited-token setting. Therefore, L_1 **regularization can reach far higher activation sparsity and maintain comparable performance to regularization-free methods with a sufficiently smooth increase trend of the factor, at the cost of an acceptable rise in training tokens** (i.e., 54.53B, only **2.73%** of the 2T tokens used to pre-train LLaMA2 (Touvron et al., 2023b)).

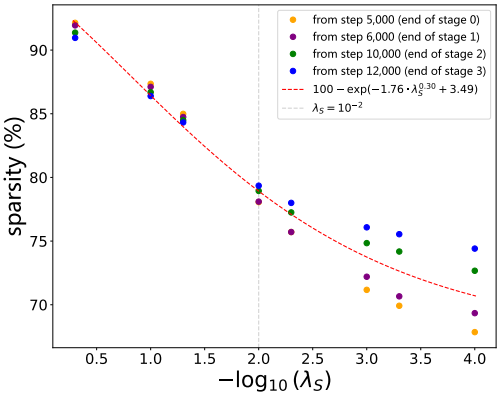

Figure 3: The activation sparsity obtained by applying different final-stage regularization factors λ_S to the checkpoints at different training stages (16,500 steps in total) of ProSparse-7B.

***Q2:* How to reach a target activation sparsity value?** Considering the additional training costs needed to reach higher sparsity, a common requirement lies in how to manipulate ProSparse to reach a desired sparsity given a limited computation budget. The key challenge is the search for suitable regularization factors. To this end, we manage to find the quantitative relationship between the final activation sparsity and the regularization factors to avoid the empirical hyper-parameter search.

Specifically, we select checkpoints at different

[¶]We did not reproduce the flat sparsity trend claimed in the paper of Shifted ReLU (Mirzadeh et al., 2023).

training stages of ProSparse-7B (see Table 6), apply a constant regularization factor λ_S, and then resume training for sufficient steps (i.e., no less than 4,000 steps) as the last regularization stage. With the same accumulated training token numbers as ProSparse-7B, we can obtain different activation sparsity by tuning the value of λ_S. The results shown in Figure 3 provide two observations: (1) **The final activation sparsity is mainly dependent on the last-stage regularization factor λ_S when λ_S is relatively large** (e.g., $\lambda_S >= 10^{-2}$ for ProSparse-7B). (2) **The activation sparsity shows a negative exponential relationship with λ_S^{α}.** For ProSparse-7B, specifically, the sparsity approximates $100 - \exp(-1.76 \cdot \lambda_S^{0.30} + 3.49)$ (i.e., the red fitted curve in Figure 3). In summary, to reach a given relatively high sparsity level (e.g., sparsity larger than 80%, satisfying $\lambda_S >= 10^{-2}$), the only thing needed is to control the regularization factor λ_S of the final progressive regularization stage. Therefore, **given the fixed model size, ProSparse is a highly controllable ReLUfication method in terms of sparsity adjustment**.

Q3: **Is progressive sparsity regularization effective?** If the activation sparsity mainly depends on the final-stage regularization factor, why should we increase the factor progressively? The answer lies in the performance concern. To substantiate the effectiveness of progressive sparsity regularization, the second step of ProSparse, we conduct ablation studies by making the regularization factor constant throughout the training process after activation function substitution. By setting the factor to 0.1, we obtain a model with activation sparsity of 88.62%, slightly lower than ProSparse-7B (89.32%). However, with the same number of training tokens, this model only has an average performance of 36.34%, considerably lower than ProSparse-7B (38.46%). Similarly, for the 13B setting, we obtain a model with the comparable sparsity of 88.96% and an average performance of 42.85%, lower than ProSparse-13B (see Appendix I). Therefore, **progressive sparsity regularization is indispensable in mitigating the performance degradation caused by ReLUfication**.

Q4: **How to SFT sparsely activated models?** It is non-trivial to apply SFT to sparsely activated models obtained by ProSparse. Our practice of training ProSparse-1B can provide some experience: **SFT can be applied to sparsely activated models obtained by ProSparse with a well-chosen regularization factor, and this factor for SFT is empirically smaller than λ_S to accommodate newly injected knowledge and avoid performance degradation**. See Appendix J for more details and observations.

Q5: **How does the sparsity distribute?** Another interesting observation is the imbalanced sparsity distributions among distinct datasets and layers. Specifically, **the activation sparsity of ProSparse models is higher on more formatted instruction tuning datasets and higher layers (i.e., layers closer to outputs)**. More detailed analyses are provided in Appendix M and N.

5 Conclusion

In this work, we propose ProSparse, an effective ReLUfication method for introducing and enhancing intrinsic activation sparsity from non-ReLU LLMs with comparable performance. Extensive experiments demonstrate the effectiveness of ProSparse and its practical values in inference acceleration with various algorithms. Deeper analyses concerned with certain ProSparse techniques, model properties, and SFT issues further substantiate the practicality of ProSparse and provide valuable insights.

Broader Impacts

This paper presents a simple and effective method, ProSparse, for introducing and enhancing ReLU-based intrinsic activation sparsity into non-ReLU LLMs. There may exist many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

Limitations

Firstly, more comprehensive studies on huge-scale models (e.g., 70B or more) should be included in the future given sufficient computing resources. Moreover, we only focus on the sparsity-based acceleration of step (2) and step (3) of FFN, leaving a considerable ratio of LLM computation unoptimized. Actually, there already exist preliminary works in the sparsification of the attention layers (Shen et al., 2023; Wortsman et al., 2023). Methods such as pruning and low-rank decomposition may also be helpful in optimizing the FFN step (1) (Ji et al., 2024). For future works, we will continue to explore how to introduce and enhance sparsity in the attention layer as well as the acceleration issue of the FFN step (1).

Acknowledgments

This work is supported by National Natural Science Foundation of China (No. 62236004, No. 62236011, No. 62302479), China Postdoctoral Science Foundation (2023M733566), and Institute Guo Qiang at Tsinghua University.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, et al. 2023. The Falcon series of open language models. *arXiv preprint arXiv:2311.16867*.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. 2022. 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.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Haoli Bai, Lu Hou, Lifeng Shang, Xin Jiang, Irwin King, and Michael R Lyu. 2022. Towards efficient post-training quantization of pre-trained language models. *Advances in Neural Information Processing Systems*, 35:1405–1418.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

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

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023a. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Jiajun Chen, Yin Gao, Zhuang Liu, and Dapeng Li. 2023b. Future vision on artificial intelligence assisted green energy efficiency network. *ZTE Communications*, 21(2).

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. PaLM: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936.

Jonathan H Clark, Eunsol Choi, Michael Collins, Dan Garrette, Tom Kwiatkowski, Vitaly Nikolaev, and Jennimaria Palomaki. 2020. TyDi QA: A benchmark for information-seeking question answering in typologically diverse languages. *Transactions of the Association for Computational Linguistics*, 8:454–470.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *International Conference on Machine Learning*, pages 933–941. PMLR.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional Transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. 2023a. Enhancing chat language models by scaling high-quality instructional conversations. *arXiv preprint arXiv:2305.14233*.

Yahao Ding, Mohammad Shikh-Bahaei, Zhaohui Yang, Chongwen Huang, and Weijie Yuan. 2023b. Secure federated learning over wireless communication networks with model compression. *ZTE Communications*, 21(1):46.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

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

Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2020. The Pile: An 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Georgios Georgiadis. 2019. Accelerating convolutional neural networks via activation map compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7085–7095.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2023. Knowledge distillation of large language models. *arXiv preprint arXiv:2306.08543*.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28.

Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. 2009. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005.

Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. 2022. Unnatural instructions: Tuning language models with (almost) no human labor. *arXiv preprint arXiv:2212.09689*.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *arXiv preprint arXiv:2305.02301*.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. 2024. MiniCPM: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713.

Yixin Ji, Yang Xiang, Juntao Li, Wei Chen, Zhongyi Liu, Kehai Chen, and Min Zhang. 2024. Feature-based low-rank compression of large language models via bayesian optimization. *arXiv preprint arXiv:2405.10616*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

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

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from Transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Patrick Lewis, Yuxiang Wu, Linqing Liu, Pasquale Minervini, Heinrich Küttler, Aleksandra Piktus, Pontus Stenetorp, and Sebastian Riedel. 2021. PAQ: 65 million probably-asked questions and what you can do with them. *Transactions of the Association for Computational Linguistics*, 9:1098–1115.

Gen Li, Yuantao Gu, and Jie Ding. 2020. The efficacy of l_1 regularization in two-layer neural networks. *arXiv preprint arXiv:2010.01048*.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. StarCoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, et al. 2022. The lazy neuron phenomenon: On emergence of activation sparsity in Transformers. In *The Eleventh International Conference on Learning Representations*.

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

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. 2023. The Flan collection: designing data and methods for effective instruction tuning. In *Proceedings of the 40th International Conference on Machine Learning*. JMLR.org.

Ilya Loshchilov and Frank Hutter. 2016. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*.

Rongrong Ma, Jianyu Miao, Lingfeng Niu, and Peng Zhang. 2019. Transformed l_1 regularization for learning sparse deep neural networks. *Neural Networks*, 119:286–298.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. LLM-Pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2023. SpecInfer: Accelerating generative LLM serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*.

Iman Mirzadeh, Keivan Alizadeh, Sachin Mehta, Carlo C Del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. 2023. ReLU strikes back: Exploiting activation sparsity in large language models. *arXiv preprint arXiv:2310.04564*.

Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. 2019. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1325–1334.

OpenAI. 2023. ChatGPT.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534.

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

Yogi Prasetyo, Novanto Yudistira, and Agus Wahyu Widodo. 2023. Sparse then prune: Toward efficient vision Transformers. *arXiv preprint arXiv:2307.11988*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text Transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. 2011. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8732–8740.

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, et al. 2021. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019. SocialIQA: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473.

Simone Scardapane, Danilo Comminiello, Amir Hussain, and Aurelio Uncini. 2017. Group sparse regularization for deep neural networks. *Neurocomputing*, 241:81–89.

Noam Shazeer. 2020. GLU variants improve Transformer. *arXiv preprint arXiv:2002.05202*.

Kai Shen, Junliang Guo, Xu Tan, Siliang Tang, Rui Wang, and Jiang Bian. 2023. A study on ReLU and Softmax in Transformer. *arXiv preprint arXiv:2302.06461*.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2023. PowerInfer: Fast large language model serving with a consumer-grade GPU. *arXiv preprint arXiv:2312.12456*.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. 2019. Distilling task-specific knowledge from bert into simple neural networks. *arXiv preprint arXiv:1903.12136*.

Robert Tibshirani. 1996. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image Transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. LLaMA 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Huan Wang, Qiming Zhang, Yuehai Wang, Lu Yu, and Haoji Hu. 2019. Structured pruning for efficient ConvNets via incremental regularization. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Yiding Wang, Kai Chen, Haisheng Tan, and Kun Guo. 2023. Tabi: An efficient multi-level inference system for large language models. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pages 233–248.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, et al. 2022. Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5085–5109.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29.

Wikimedia Foundation. 2022. Wikimedia downloads.

Mitchell Wortsman, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. 2023. Replacing softmax with ReLU in vision Transformers. *arXiv preprint arXiv:2309.08586*.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2023. Sheared LLaMA: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR.

Zhewei Yao, Cheng Li, Xiaoxia Wu, Stephen Youn, and Yuxiong He. 2023. A comprehensive study on post-training quantization for large language models. *arXiv preprint arXiv:2303.08302*.

Ming Yuan and Yi Lin. 2006. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 68(1):49–67.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022a. OPT: Open pre-trained Transformer language models. *arXiv preprint arXiv:2205.01068*.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2022b. MoEfication: Transformer feed-forward layers are mixtures of experts. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 877–890.

Zhengyan Zhang, Yixin Song, Guanghui Yu, Xu Han, Yankai Lin, Chaojun Xiao, Chenyang Song, Zhiyuan Liu, Zeyu Mi, and Maosong Sun. 2024. ReLU^2 wins: Discovering efficient activation functions for sparse llms. *arXiv preprint arXiv:2402.03804*.

Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. 2016. Loss functions for image restoration with neural networks. *IEEE Transactions on computational imaging*, 3(1):47–57.

Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. 2019. Improving neural network quantization without retraining using outlier channel splitting. In *International Conference on Machine Learning*, pages 7543–7552. PMLR.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2023. AGIEval: A human-centric benchmark for evaluating foundation models. *arXiv preprint arXiv:2304.06364*.

Mingjian Zhu, Yehui Tang, and Kai Han. 2021. Vision Transformer pruning. *arXiv preprint arXiv:2104.08500*.

A Extended Related Works

L_1 Regularization In statistical learning such as linear regression, L_1 regularization has been long adopted as a classical and effective technique for sparsification (Tibshirani, 1996; Hastie et al., 2009). With the advent of deep learning, researchers also explore paradigms of applying L_1 regularization to neural networks. One prominent usage is model pruning (Cheng et al., 2017). Specifically, a term of loss calculated as the L_1 norm of the sparsification target is added to the optimization target function to prompt sparse weights for faster computation. This has helped acceleration in various conventional neural networks (Han et al., 2015; Zhao et al., 2016; Wen et al., 2016; Scardapane et al., 2017; Ma et al., 2019; Wang et al., 2019) as well as Transformer-based models (Zhu et al., 2021; Prasetyo et al., 2023). Inspired by these works, some researchers also try to adopt L_1 regularization for activation sparsity, mainly in ReLU-activated convolutional networks (Georgiadis, 2019; Kurtz et al., 2020) and Transformer-based architectures (Li et al., 2022).

To the best of our knowledge, ProSparse is the first work using a dynamic L_1 regularization factor for prompting activation sparsity in neural networks. By contrast, a majority of the former works adopt fixed factors. For more adaptive control, some of them introduce group regularization (Yuan and Lin, 2006), namely using different factors for different parameter groups. Nevertheless, without dynamic factors, these paradigms can cause a substantial shift in activation distribution and thus potentially risk performance degradation. The work most related to ProSparse is IncReg (Wang et al., 2019), which introduces incremental regularization factors that change for different parameter groups at each iteration. While they focus on the pruning of convolutional networks, ProSparse handles a distinct scenario of prompting activation sparsity in Transformer-based LLMs and adopts a completely different strategy consisting of a progressively incremental factor.

Difference between Activation Sparsity and Pruning Generally, pruning realizes sparsity by removing certain elements (e.g., neurons, weights, or structured blocks) in LLMs. However, the sparsity introduced by pruning is statically limited to model weights and independent of the inputs. High static sparsity is often accompanied by considerable performance degradation compared to the original dense model (Frantar and Alistarh, 2023; Xia et al., 2023). By contrast, activation sparsity is dynamically determined by the input data and thus potentially compromises less model capacity and downstream task performance.

B Detailed Algorithm for Progressive Sparsity Regularization

The detailed algorithm for scheduling the factor λ in progressive sparsity regularization is listed in Algorithm 1.

Algorithm 1 Progressive factor scheduling adopted in ProSparse

Require: The total number of stages $S \geq 1$.
Require: A sequence of peak λ values $\{\lambda_i\}_{i=1}^{S}$, s.t. $0 < \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_S$.
Require: Accumulated step numbers of each stage $\{T_i\}_{i=1}^{S}$, s.t. $0 < T_1 < T_2 < ... < T_S$.

```
// warmup stage
for t ← 1 to T_1 do
    λ ← λ_1
    update model by loss L_lm + L_reg(λ)
end for
// incremental stages
for i ← 2 to S do
    for t ← T_{i-1} + 1 to T_i do
        η ← 1/2 [sin(-π/2 + (t - T_{i-1})/(T_i - T_{i-1}) π) + 1]
        λ ← λ_{i-1} + η(λ_i - λ_{i-1})
        update model by loss L_lm + L_reg(λ)
    end for
end for
```

C Extented Introduction of Approximate Acceleration Algorithms

Existing approximate algorithms are mostly dependent on activation predictors, which are small neural networks to predict the intermediate activations $\mathbf{x}_1$ based on the input hidden states $\mathbf{x}$ (Liu et al., 2023; Song et al., 2023). If one element at a specific position of $\mathbf{x}_1$ is predicted to be zero, then all the computations associated with this position can be saved with little or no hardware resources allocated. This is the key mechanism with which approximate algorithms realize acceleration.

Nevertheless, such a predictor-dependent acceleration effect is largely dependent on the performance of the pre-trained activation predictors. For example, a typical bad case is that an actually activated element in $\mathbf{x}_1$ is predicted to be inactivated.

This can bring about negative results including unwise hardware resource allocation and erroneously ignored intermediate logits, which limits the practical speedup ratios and even causes inference inaccuracies. Therefore, a sparse LLM can gain more benefits from approximate algorithms if its activation distribution is more predictable.

To test a sparse LLM's practical acceleration value with approximate algorithms, we involve the predictability of its activation distribution, which is evaluated by the performance of its specifically pre-trained activation predictor. This involves two key metrics: the activation recall and the predicted sparsity.

The activation recall refers to the average ratio of correctly predicted activated elements among all the truly activated elements in $\mathbf{x}_1$. The predicted sparsity is calculated as the ratio of predicted inactivated elements among all the elements in $\mathbf{x}_1$. A predictor with higher recall will miss less truly activated elements, therefore reducing inference inaccuracies and bringing about wiser hardware allocation. Under comparable recalls, a higher predicted sparsity indicates fewer elements to be falsely predicted activated, which largely alleviates the waste of computational resources. These can help an acceleration framework obtain a better grasp of activation distribution and thus make wiser policies for faster inference as well as low inference inaccuracies (Liu et al., 2023).

D Implementation Details of Sparse GPU Operators

Input-Side Sparse Operator. The input-side sparse operator is a sparse matrix-vector multiplication operator for accelerating $\mathbf{x}_1\mathbf{W}_2^T$, where the input $\mathbf{x}_1$ is sparse. Due to the sparsity of input, any operation involving a zero element in $\mathbf{x}_1$ can be omitted. Compared with a standard implementation of matrix-vector multiplication, both memory access and computation of the sparse operator will decrease with the sparsity increasing.

Output-Side Sparse Operator. The output-side sparse operator is a fused operator consisting of ReLU, sparse matrix-vector multiplication, and element-wise multiplication, for accelerating $\mathbf{s} \odot (\mathbf{x}\mathbf{W}_1^T)$, where $\mathbf{s}$ is sparse. The sparsity of $\mathbf{s}$ can be propagated to the output of $\mathbf{x}\mathbf{W}_1^T$ through element-wise multiplication, which means that the computation of matrix-vector multiplication in $\mathbf{x}\mathbf{W}_1^T$ can be skipped whenever a result element will be multiplied by zero of sparse $\mathbf{s}$. In addition, we postpone the ReLU activation function in $\sigma(\mathbf{x}\mathbf{W}_s^T)$ into this operator so that σ can be implicitly performed along with the element-wise multiplication. These operations are fused into a single operator, thereby reducing the data movement between operations.

For implementation, we first load the result of $\mathbf{x}\mathbf{W}_s^T$, determine which elements are greater than zero (or a positive threshold after activation threshold shifting), and then select the corresponding columns of $\mathbf{W}_1^T$ to load from GPU memory, performing multiplication operations with $\mathbf{x}$. As the matrix $\mathbf{W}_1^T$ is sparse by column, we store the matrix in a column-major format to coalesce memory access and fully utilize vectorized loads/store instructions. After this step, we get the sparse result vector of $\mathbf{x}\mathbf{W}_1^T$ and multiply the corresponding elements with activated elements of $\mathbf{s}$, with other elements filled with zeros directly. Finally, the result vector $\mathbf{x}_1$ is obtained.

E Training and Evaluation Datasets

Training Datasets Our mixed training data consists of both language modeling datasets and instruction tuning datasets. The language modeling datasets are directly cleaned and filtered from raw corpus, including StarCoder (Li et al., 2023), Wikipedia (Wikimedia Foundation, 2022), Pile (Gao et al., 2020), and other collected datasets. The instruction tuning datasets mainly involve input instructions and annotated target answers, including UltraChat (Ding et al., 2023a), multiple-choice QA data of P3 (Sanh et al., 2021) (Choice P3), PAQ (Lewis et al., 2021), Unnatural Instructions (Honovich et al., 2022), Flan (Longpre et al., 2023), Super-Natural Instructions (Wang et al., 2022), and other collected datasets.

Evaluation Benchmarks To evaluate the task-specific performance of the LLMs obtained by ProSparse, we adopt the following comprehensive benchmarks.

(1) *Code Generation*: We compute the average pass@1 scores on HumanEval (0-shot) (Chen et al., 2021) and MBPP (3-shot) (Austin et al., 2021). (2) *Commonsense Reasoning*: We report the average 0-shot accuracies on PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2020), and COPA (Roemmele et al., 2011). (3) *Reading Comprehension*: We compute the average 0-shot accura-

Setting	HumanEval	MBPP	PIQA	SIQA	HellaSwag	WinoGrande	COPA	BoolQ	LAMBADA	TyDi QA
Original-7B	10.98	21.77	78.40	47.70	75.67	67.17	79.00	75.99	72.81	36.82
ReluLLaMA-7B	12.20	19.51	77.86	49.54	72.85	64.96	83.00	78.10	70.33	63.18
ProSparse-7B*	16.46	22.48	75.79	43.50	71.08	64.09	77.00	62.48	67.73	59.77
ProSparse-7B	16.46	22.38	75.68	43.55	71.09	64.01	77.00	62.51	68.21	59.77
Original-13B	16.46	23.92	79.38	47.90	79.12	70.48	86.00	82.54	76.21	55.91
ReluLLaMA-13B	17.07	23.31	78.40	47.13	76.60	69.06	81.00	81.16	73.49	65.23
ProSparse-13B*	25.61	32.44	77.04	45.14	75.91	68.67	82.00	79.27	71.08	52.27
ProSparse-13B	23.78	33.06	77.26	45.29	75.88	68.35	82.00	78.93	71.36	50.45
MiniCPM-1B	41.46	32.24	74.10	48.87	67.27	59.12	69.00	69.30	58.18	55.23
ProSparse-1B*	48.78	33.98	73.99	47.54	68.46	59.75	73.00	74.92	53.74	53.41
ProSparse-1B	50.61	33.47	74.16	47.54	68.40	59.75	72.00	75.08	53.74	53.36

Table 3: The performance (%) on each independent benchmark.

Setting	Accumulated Tokens (B)	Average Sparsity (%)	Average Performance (%)	Accumulated Tokens (B)	Average Sparsity (%)	Average Performance (%)
Vanilla ReLU	34.60	66.04	41.40	89.13	64.93	41.52
Shifted ReLU	34.60	69.59	41.33	89.13	68.35	41.40
ProSparse	34.60	89.32	38.46	89.13	88.29	40.67

† Note that ProSparse with 89.13B tokens applies different hyper-parameters (i.e., more training stages and step numbers) for a smoother trend of regularization factor increase and thus obtains higher performance than the 34.60B setting.

Table 4: Comparison of ProSparse with two regularization-free former ReLUfication methods (7B). The bias b for shifted ReLU is tuned to ensure the best performance.

cies on BoolQ (Clark et al., 2019), LAMBADA (Paperno et al., 2016), and TyDi QA (Clark et al., 2020). (4) *Other Popular Benchmarks*: We report the average accuracies on GSM8K (8-shot) (Cobbe et al., 2021), MMLU (5-shot) (Hendrycks et al., 2020), Big Bench Hard (BBH) (3-shot) (Suzgun et al., 2022), and AGI-Eval (0-shot) (Zhong et al., 2023). Refer to Appendix K for more details.

F Training Details of Activation Predictors

Following Deja Vu (Liu et al., 2023), the predictor is a two-layer FFN, composed of two linear projection layers with a ReLU activation in between them. Notably, as each layer of a sparse LLM has different activation distributions, we should introduce the same number of predictors as that of Transformer layers. For predictor training, we first collect about 400,000 pairs of input hidden states $\mathbf{x}$ and intermediate activations $\mathbf{x}_1$ at the corresponding layer. Next, we train the predictor on 95% pairs with the binary cross entropy loss and compute the predictability metrics on the remaining 5% pairs. We reserve the checkpoint with the highest recall to ensure the best inference accuracy with the least falsely ignored activations.

G Performance on Independent Benchmarks

In this section, we report the performance on each independent benchmark of Code Generation, Commonsense Reasoning, and Reading Comprehension, as displayed in Table 3.

H Performance Comparison between ProSparse and Baselines

Given the different amounts of training tokens, we compare ProSparse with the other two baselines (i.e., vanilla ReLU and shifted ReLU) in terms of the average sparsity and performance. As shown in Table 4, ProSparse can achieve far higher sparsity than two baselines. Besides, with more training tokens given, ProSparse is able to apply a smoother trend of regularization increase and thus better mitigate performance degradation. This is why the performance gap between ProSparse and two regularization-free baselines is bridged when the tokens increase from 34.60B to 89.13B. The additional training costs, namely 54.53B tokens, only account for about 2.73% of the pre-training costs of the original LLaMA2 (Touvron et al., 2023b) and are well acceptable.

I Ablation Studies of Progressive Sparsity Regularization

Here we provide more detailed experimental results about the ablation of progressive sparsity regularization, as shown in Table 5. Note that for ablation settings, we keep the regularization factor constant without progressive increase. More specifically, the whole training process consists of three steps: activation function substitution, continual training

Scale	λ	Average Sparsity (%)	Average Performance (%)	Scale	λ	Average Sparsity (%)	Average Performance (%)
7B	$5e-2$	85.95	37.83	13B	$1e-2$	86.23	43.87
7B	$1e-1$	88.62	36.34	13B	$2e-2$	88.96	42.85
7B	$5e-1$	93.15	33.86	13B	$5e-2$	92.65	40.13
7B	ProSparse	89.32	**38.46**	13B	ProSparse	88.80	**44.90**

Table 5: Ablation study results about progressive sparsity regularization, the second step of ProSparse. λ refers to the constant regularization factor in the second stage of ablation settings.

with a constant regularization factor, and activation threshold shifting.

J SFT for Sparsely Activated Models

The key problem for SFT sparsely activated models is how to feed instruction following knowledge into the model while maintaining the sparsity simultaneously. From the above observations about training dynamics, the regularization factor is still indispensable during SFT to avoid a considerable drop in sparsity. Moreover, the factor is probably not equal to the one used in the final progressive regularization stage (i.e., λ_S), as the data distribution has shifted radically.

Our practice of training ProSparse-1B can provide empirical answers to this problem. Concretely, while ProSparse-7B and ProSparse-13B are directly trained from the original LLaMA2 on mixed data through the three-step ProSparse, the training of ProSparse-1B includes an extra decay stage and an SFT stage, following the original practice of MiniCPM for better performance (see Table 7). The decay stage is conducted on the mixed data, with a decreasing learning rate and a fixed regularization factor of value λ_S. By contrast, the SFT stage is performed only on the instruction tuning data. We find that **the regularization factor for SFT should be empirically smaller than λ_S in order to accommodate newly injected knowledge from SFT data and avoid performance degradation**. For ProSparse-1B, an SFT factor of $1e-2$ works the best with an average performance of 44.72%, while the performance drops to 44.32% with $\lambda_S = 5e-2$. Therefore, **SFT can be applied to sparsely activated models obtained by ProSparse with a well-chosen regularization factor**.

K Evaluation Details

For evaluation benchmarks including PIQA, SIQA, HellaSwag, WinoGrande, COPA, BoolQ, LAMBADA, TyDi QA, and AGI-Eval, we obtain the predicted answers based on maximized perplexity. Specifically, the predicted answer to a given question corresponds to the candidate that produces the lowest perplexity when it is concatenated to the question. For GSM8K, MMLU, and BBH, the predicted answers are determined by the option numbers directly generated by the models.

L Important Hyperparameters

We provide the important hyperparameters for ProSparse training, as shown in Table 6 and Table 7. Note that the peak regularization factors of two contiguous stages can be set to the same value to introduce an extra constant-factor stage, mainly for stability requirements. For ProSparse-7B and ProSparse-13B, We use a cosine annealing learning rate scheduler throughout the training process and the peak learning rates are $3e-5$ and $5e-5$ for 7B and 13B respectively. For ProSparse-1B, we use exactly the same hyper-parameter settings as MiniCPM-1B (Hu et al., 2024) except for the L_1 regularization. After pre-training on the language modeling dataset with the paradigm of ProSparse, following the original practice, we add an extra decay stage (mixed data with a decreasing learning rate) and an SFT stage (only instruction tuning data with a fixed learning rate). Each of the additional stages has a constant regularization factor. The context length is 4,096 for all settings. Considering cost issues, the hyper-parameters for ProSparse are set to appropriate values to just match the original Swish-activated versions in terms of benchmark performance.

All the 7B models are trained with the AdamW optimizer on 8 A100 80GB GPUs for about 10 days. All the 13B models are trained on 32 A100 80GB GPUs for about 20-30 days. The LLMs of each method involved in this paper are trained once due to the formidable training costs.

M Dataset-Wise Sparsity Distribution

Despite the satisfactory average sparsity, there still exist gaps between the mixed training dataset and the actual input texts that the model will encounter

ProSparse-7B				ProSparse-13B			
Stage Number i	λ_i	T_i	Accumulated Tokens (B)	Stage Number i	λ_i	T_i	Accumulated Tokens (B)
0	0	5,000	10.49	0	0	5,500	46.14
1	$5e-3$	6,000	12.58	1	$5e-3$	6,750	56.62
2	$5e-2$	10,000	20.97	2	$1e-2$	10,750	90.18
3	$5e-2$	12,000	25.17	3	$1e-2$	11,000	92.27
4	$2e-1$	16,000	33.55	4	$2e-2$	15,000	125.83
5	$2e-1$	16,500	34.60	5	$2e-2$	16,000	134.22

Table 6: The important hyperparameters for training ProSparse-7B and ProSparse-13B. For simplicity, the 0th stage refers to the continual training in activation function substitution. The 1st stage is the warmup stage with a fixed regularization factor λ_1. The remaining stages are incremental stages with an increasing factor.

ProSparse-1B			
Stage Number i	λ_i	T_i	Accumulated Tokens (B)
0	0	10,000	49.15
1	$1e-3$	15,000	73.73
2	$5e-3$	20,000	98.30
3	$5e-3$	25,000	122.88
4	$5e-2$	35,000	172.03
decay	$5e-2$ (fixed)	95,000	466.94
SFT	$1e-2$ (fixed)	101,000	473.02

Table 7: The important hyperparameters for ProSparse-1B. Compared with the other two settings, we follow the original practice of MiniCPM-1B (Hu et al., 2024), appending an extra decay stage and an SFT stage. Note that each of the additional stages has a constant regularization factor.

in real-life applications. To investigate the sparsity of our model under different scenarios, we compute the sparsity on each component of the mixed training data respectively.

As demonstrated in Table 8, the sparse LLMs obtained through ProSparse have a pronounced property of inconsistent dataset-wise sparsity. Concretely, the sparsity on instruction tuning datasets is significantly higher than those on language modeling datasets (i.e., StarCoder, Wikipedia, and Pile). Considering the contents of datasets, we come to the following assumption: **the more formatted a dataset is, the less hybrid knowledge is needed for generation, and thus the L_1-regularized models can achieve higher activation sparsity with fewer neurons activated.** Plain text datasets including Wikipedia and Pile have the lowest sparsity, followed by the more formatted code dataset StarCoder. Among instruction tuning datasets, QA datasets (e.g., Choice P3) with the most monotonic input-output formats obtain the highest sparsity. By contrast, the sparsity is relatively lower on UltraChat and Flan, covering general dialogues and a wide variety of tasks respectively. Notably, dialogues and tasks with formatted instructions cover a majority of input contents of conversational AI, the mainstream application form of LLMs. Such higher sparsity on instruction tuning data will endow ProSparse with more significant practical values.

N Layer-Wise Sparsity Distribution

Another problem worth concern is the layer-wise sparsity, which potentially impacts the load balance and the inference efficiency. Therefore, we compute the sparsity of each layer for ProSparse models, as shown in Figure 4.

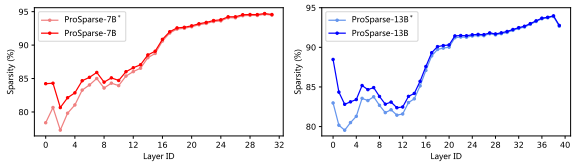

Figure 4: The layer-wise sparsity of ProSparse models. The marker "*" denotes the settings without activation threshold shifting.

From the tendency of the line chart, we clearly observe layer-wise sparsity imbalance in that lower layers are significantly denser than higher layers. Nevertheless, the activation threshold shifting can considerably improve the sparsity of lower layers with little impact on higher layers. Although this technique only contributes marginally to the average sparsity, it is still indispensable in alleviating the layer-wise sparsity imbalance issue.

O Effect of Different Thresholds in Activation Threshold Shifting

As mentioned in Section 3.2, the threshold t is an important hyper-parameter in activation threshold shifting, the last step of ProSparse. In the overall experimental results, we choose $t = 0.01$ for both ProSparse-7B and ProSparse-13B to balance the sparsity and performance. Here we list the results

Setting	Mixed	StarCoder	Wikipedia	Pile	UltraChat	Choice P3	PAQ	Flan	Unnatural Instructions	Super-Natural Instructions
ReluLLaMA-7B	66.98	66.60	67.16	67.35	67.91	67.35	66.98	67.35	66.42	66.98
ProSparse-7B*	88.11	88.20	83.30	84.24	91.23	97.94	96.74	90.76	93.00	95.71
ProSparse-7B	89.32	89.13	84.33	85.35	93.66	98.33	97.28	91.74	93.80	96.32
ReluLLaMA-13B	71.56	71.33	71.45	71.56	72.27	71.80	71.21	71.56	70.85	71.33
ProSparse-13B*	87.97	87.50	81.64	83.06	92.45	98.41	97.54	91.65	92.92	96.40
ProSparse-13B	88.80	88.63	83.65	84.12	92.65	98.73	97.99	92.54	93.66	96.92
ProSparse-1B*	86.25	86.84	82.72	83.23	88.50	89.83	83.36	83.93	90.16	90.70
ProSparse-1B	87.89	88.44	84.71	85.17	89.93	91.01	85.36	85.82	91.36	91.76

Table 8: The average sparsity (%) on our mixed training dataset (denoted as "Mixed") and its components, divided into language modeling datasets and instruction tuning datasets.

Setting	Average Sparsity	Code Generation	Commonsense Reasoning	Reading Comprehension	GSM8K	MMLU	BBH	AGI Eval	Average Performance
ProSparse-7B*	88.11	19.47	66.29	63.33	12.74	45.21	33.59	27.55	38.31
ProSparse-7B $t = 0.005$	88.62	19.68	66.23	62.59	12.05	44.95	34.43	27.46	38.20
ProSparse-7B $t = 0.01$	89.32	19.42	66.27	63.50	12.13	45.48	34.99	27.46	38.46
ProSparse-7B $t = 0.02$	90.35	18.39	66.09	62.93	12.59	45.02	34.34	27.14	38.07
ProSparse-7B $t = 0.03$	90.95	18.65	66.24	62.72	12.13	44.83	34.92	27.36	38.12
ProSparse-13B*	87.97	29.03	69.75	67.54	25.40	54.78	40.20	28.76	45.07
ProSparse-13B $t = 0.005$	88.24	29.04	69.69	67.62	26.23	54.75	39.52	28.74	45.08
ProSparse-13B $t = 0.01$	88.80	28.42	69.76	66.91	26.31	54.35	39.90	28.67	44.90
ProSparse-13B $t = 0.02$	89.40	29.29	69.63	65.28	24.94	54.88	39.79	28.88	44.67
ProSparse-13B $t = 0.03$	90.23	28.12	69.28	64.79	25.85	54.68	40.08	28.71	44.50

Table 9: The sparsity (%) and performance (%) under different thresholds t of the activation threshold shifting step.

under other thresholds in Table 9. As can be observed, a small t results in a quite limited sparsity improvement compared with the version without activation threshold shifting, while a larger t can cause more performance degradation. Therefore, we choose $t = 0.01$ to strike a balance.