Enhancing One-Shot Pruned Pre-trained Language Models through Sparse-Dense-Sparse Mechanism

Guanchen Li[1,*], **Xiandong Zhao**[1,*], **Lian Liu**[1], **Zeping Li**[1], **Yixing Xu**[1], **Dong Li**[1], **Lu Tian**[1], **Jie He**[2], **Ashish Sirasao**[1], **Emad Barsoum**[1],
[1]Advanced Micro Devices, Inc., [2] University of Science and Technology Beijing
{guanchen.li, emad.barsoum}@amd.com

Abstract

Pre-trained language models (PLMs) are engineered to be robust in contextual understanding and exhibit outstanding performance in various natural language processing tasks. However, their considerable size incurs significant computational and storage costs. Modern pruning strategies employ retraining-free one-shot techniques to compress PLMs; however, these approaches often lead to an indispensable reduction in performance. In this paper, we propose **SDS**, a **S**parse-**D**ense-**S**parse pruning framework to enhance the performance of the pruned PLMs from a weight distribution optimization perspective. We outline the pruning process in three steps. Initially, we prune less critical connections in the model using conventional one-shot pruning methods. Next, we reconstruct a dense model featuring a pruning-friendly weight distribution by reactivating pruned connections with *sparse regularization*. Finally, we perform a second pruning round, yielding a superior pruned model compared to the initial pruning. Experiments demonstrate that SDS outperforms the state-of-the-art pruning techniques SparseGPT and Wanda under an identical sparsity configuration. For instance, SDS reduces perplexity by 5.16 on Raw-Wikitext2 and improves average accuracy by 3.86% across multiple zero-shot benchmarks for LLaMA-3-8B compared to Wanda with 2:4 sparsity.

1 Introduction

Pre-trained language models (PLMs) (Vaswani et al., 2017) have revolutionized various applications in natural language processing. However, the considerable size of PLMs results in notable drawbacks, such as increased latency and energy consumption. Compression methods for vision models, which perform *pre-training, compression, and fine-tuning* workflow with quantization or pruning (Liang et al., 2021), may entail substantial time and energy costs for PLMs due to their massive training requirements.

PLM	Dense	2:4 Sparse	Re-dense
OPT-125M	27.66	60.43	**27.94**
OPT-350M	22.01	51.11	**22.25**

Table 1: **Pruning PLMs Incurs Resumable Knowledge Loss.** We apply 2:4 sparsity to OPTs with SparseGPT, and their performance decreases on Raw-WikiText2. However, upon layer-wise reactivating the sparse weights with only 128 samples from C4 (Subsection 2.2 provides re-dense details; note here is no sparse regularization present), a substantial performance improvement is observed.

Recent pruning research, such as SparseGPT (Frantar and Alistarh, 2023) and Wanda (Sun et al., 2023), has introduced effective one-shot compression techniques for PLMs. These methods can compress up to 50% of the parameters in the fully connected layers of over-parameterized PLMs like OPT-175B without an obvious performance drop. However, their effectiveness diminishes when applied to compact ones, which show a less pronounced redundancy. For instance, SparseGPT, the state-of-the-art pruning method, yields a perplexity of 31.58 when applied to prune 50% of the weights in OPT-350M. This score is worse than the 27.66 perplexity observed in OPT-125M, a dense model with roughly half the parameters of OPT-350M. Furthermore, when stricter sparsity constraints are employed, such as higher sparsity (60% - 80%) or semi-structured n:m sparsity (2:4, 4:8) (Mishra et al., 2021) for computational acceleration, the performance deteriorates even further. Therefore, it is essential to optimize the poorly pruned PLMs.

Macroscopically, PLMs are not designed to be aware of subsequent pruning since they lack pruning-related regularization during pre-training. As a result, pruning PLMs while maintaining their performance proves challenging.

[*]Equal contribution.

Proceedings of the 31st International Conference on Computational Linguistics, pages 1718–1735
January 19–24, 2025.

Neurons in the human brain show sparse-to-dense-to-sparse connectivity as they grow (Herculano-Houzel et al., 2010). This observation inspired us to perform a similar process to achieve a better pruning scheme that benefits from pruning friendliness. We preliminarily explored layer-wise dense reconstruction to find a performance upper bound. Intriguingly, we discovered that sparse models could bounce back to performance levels equivalent to their dense counterparts using only a few samples (cf., Table 1). It reveals two key insights: first, pruned PLMs can be optimized with limited resources; second, the amount of knowledge lost during the pruning process is restorable. These insights lay the foundation for the work presented in this paper.

We propose a three-step **S**parse-**D**ense-**S**parse (**SDS**) pruning framework to enhance the performance of pruned pre-trained language models. **In the first step**, we employ conventional one-shot pruning methods on a PLM to remove irrelevant connections. **In the second step**, we perform a dense reconstruction of the sparse model to reactivate the pruned connections, aiming to identify a dense model with enhanced pruning awareness. This process is aided by a multidimensional sparse regularization strategy, which optimally guides the weight distribution, rendering it more pruning-friendly for the subsequent step. **In the third step**, we further prune and adjust the weights of the second-pruned model. Importantly, SDS requires only a limited number of samples for calibration, identical to conventional one-shot methods. Experimental results demonstrate that SDS outperforms SparseGPT and Wanda under the same sparsity configuration. For example, SDS reduces perplexity by 5.16 on Raw-Wikitext2 and increases accuracy by an average of 3.86% across multiple zero-shot downstream tasks for LLaMA-3-8B with 2:4 sparsity compared to Wanda. The pruned PLMs achieve up to 1.87x acceleration on an AMD R7 Pro CPU. The main contributions of the paper are summarized as follows:

- We introduce SDS, a three-step Sparse-Dense-Sparse framework. It involves weight redistribution and pruning, enhancing the performance of the one-shot pruned pre-trained language models.
- We design sparse regularization strategies that improve the effectiveness of re-dense weight reconstruction and find a more pruning-friendly weight distribution.
- Experimental results demonstrate that SDS outperforms existing pruning methods in both language modeling and downstream tasks.

2 Sparse-Dense-Sparse Framework

This section presents the Sparse-Dense-Sparse (SDS) framework to perform optimization for pruned pre-trained language models (PLMs). Firstly, we provide a brief overview of the core Transformer architecture, which is fundamental to most PLMs. A standard Transformer block consists of two main modules: a multi-head attention (MHA) layer and a feed-forward network (FFN). Let $\mathbf{X}_{\ell-1} \in \mathbb{R}^{d\times n}$ represent the input of the ℓ-th Transformer block, where n is the sequence length, and d is the size of the hidden state. The block output $\mathbf{X}_\ell$ can be formulated as:

$$\begin{aligned}\mathbf{X} &= \mathrm{MHA}\left(\text{ LayerNorm }\left(\mathbf{X}_{\ell-1}\right)\right)+\mathbf{X}_{\ell-1},\\ \mathbf{X}_\ell &= \mathrm{FFN}(\text{ LayerNorm }(\mathbf{X}))+\mathbf{X}.\end{aligned} \tag{1}$$

MHA consists of h heads, represented as $\mathbf{W}^{\mathrm{O}} \cdot \mathtt{concat}(\mathrm{head}_1, \mathrm{head}_2, \ldots, \mathrm{head}_h)$, with $\mathbf{W}^{\mathrm{O}}$ responsible for the output projection. Specifically, the i-th head can be expressed as:

$$\begin{aligned}\mathrm{head}_i &= \mathrm{Attn}([\mathbf{W}^{\mathrm{Q}}\mathbf{X}]_i, [\mathbf{W}^{\mathrm{K}}\mathbf{X}]_i, [\mathbf{W}^{\mathrm{V}}\mathbf{X}]_i, \boldsymbol{M}),\\ \mathrm{Attn}(\mathbf{Q},\mathbf{K},\mathbf{V},\mathbf{M}) &= \mathrm{softmax}\left(\mathbf{M}\odot\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_{\mathrm{K}}}}\right)\mathbf{V},\end{aligned} \tag{2}$$

where $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ represent the query, key, and value sequences, respectively, and their corresponding projection weights are $\mathbf{W}^{\mathrm{Q}}$, $\mathbf{W}^{\mathrm{K}}$, and $\mathbf{W}^{\mathrm{V}}$. d_{K} is the dimension of the key vectors, and $\mathbf{M}$ is the mask matrix to selectively ignore or give weight to specific tokens in the input sequence. FFN expands and contracts input dimensions through hidden layers, introducing non-linearities to enhance representation learning, which consists of several fully connected layers, with their weights represented as $\mathbf{W}^{\mathrm{Up}}$, $\mathbf{W}^{\mathrm{Down}}$, and $\mathbf{W}^{\mathrm{Gate}}$ (optional) respectively.

The SDS framework consists of three steps: initial pruning (Subsection 2.1), re-dense weight reconstruction (Subsection 2.2), and a second round of pruning (Subsection 2.3). By optimizing weight distribution through these steps, the SDS framework enhances the performance of pruned PLMs. Figure 1 provides an overview of SDS.

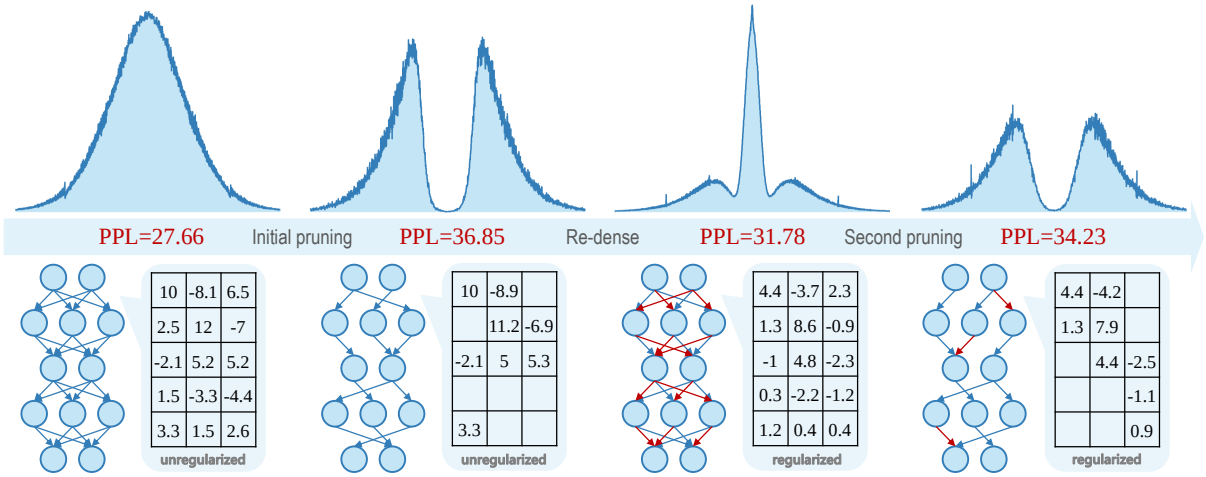

Figure 1: **An Overview of the Steps of the SDS Framework**, divided into initial pruning, re-dense weight reconstruction, and a second round of pruning. The upper figure shows the weight distribution variation within the SDS framework, and the lower figure demonstrates the variation in weight connections. The weights are extracted from the FFN in the 12-th transformer block of OPT-125M, with 50% sparsity configuration. Initially, the dense weights follow a Gaussian distribution. After being pruned by SparseGPT, a concentrated, bimodal distribution emerges (zero values are omitted in sparse weight distributions for better clarity). Followed by connection reconstruction with sparse regularization, a three-peaked distribution materializes. Finally, the second pruning round attenuates the sharp peaks, resulting in a softer bimodal distribution. Perplexity (PPL) is evaluated on Raw-WikiText2. The second pruned model achieves a lower perplexity than the initially pruned one.

2.1 Initial Pruning

The SDS framework initiates by eliminating the less critical connections in PLMs using conventional one-shot pruning methods. SparseGPT (Frantar and Alistarh, 2023) leverages second-order information to perform a weight update dependent pruning. Concretely, during column-wise pruning, SparseGPT compensates for the pruning error of the pruned columns (prior to column c) by updating the unpruned columns (subsequent to column c) of the weight matrix ($\mathbf{W}_{:,c:} = \mathbf{W}_{:,c:} - \mathbf{\Delta}$). Wanda (Sun et al., 2023) switches the perspective to pruning mask selection and realizes weight update-free pruning by considering both weight and activation magnitude. The salience metrics and weight update rules for the two preferred pruning methods are shown in Table 2.

Method	Salience Metric	Weight Update $\mathbf{\Delta}$
SparseGPT	$\frac{\mathbf{W}^{\text{dense}\,2}_{:,c}}{[\mathbf{H}^{-1}]^2_{c,c}}$	$\frac{\mathbf{W}^{\text{dense}}_{:,c} - \mathbf{W}^{\text{sparse}}_{:,c}}{[\mathbf{H}^{-1}]_{c,c}} \cdot [\mathbf{H}^{-1}]_{c,:}$
Wanda	$\lvert\mathbf{W}^{\text{dense}}\rvert \odot \lVert\mathbf{X}\rVert_2$	- Update Free -

Table 2: Saliency Metrics and Weight Update Rules for Conventional Pruning Method.

SparseGPT and Wanda demonstrate robust performance on high-redundancy models at 100B+ scale, achieving negligible performance drop with 50% sparsity. However, its efficacy diminishes when applied to compact, lower over-parameterized ones. In macro terms, the weight distribution of the pre-trained models is inappropriate for direct pruning due to the lack of sparse regularization. Thus, we take SparseGPT / Wanda as the initial step and identify a superior sparse model from the perspective of weight distribution optimization in the subsequent steps.

2.2 Re-dense Weight Reconstruction

Table 1 demonstrates that only a small number of calibration samples are needed to restore the performance of pruned PLMs. Thus the weights of dense models with similar performance are not unique, and we aim to find a **pruning-friendly** solution in the solution space of the dense model.

Concretely, we implement layer-wise knowledge distillation with the same samples as the initial pruning step to reactivate the pruned connections. By aligning layer outputs in a knowledge distillation manner without applying task-specific losses, this approach is not only parameter efficient (loading only the under-optimized layer), but also reduces biases that may arise from overfitting. We introduce three sparse regularization strategies for pruning friendliness. **a) Residual sparse characteristics**: the initial pruning cannot be omitted; it

provides prior information about which weights are relatively important for re-dense weight reconstruction. **b) Data-based regularization**: higher-loss sparse-model activations are used as the inputs of re-dense weight reconstruction. Not only does such activations already contain a priori filtering of the importance of the neurons, it also avoids overfitting. **c) Weight-based regularization**: typical weight regularization is employed to endow the re-dense weights with sparse features and increase pruning friendliness. We choose the L1 and L2 regularization (Tibshirani, 1996; Loshchilov and Hutter, 2019) to meet the requirement.

According to the above considerations, the re-dense weight reconstruction process is specified in the following. Given the original dense weight $\mathbf{W}_\ell^{\text{dense}}$ in layer ℓ, the sparse weight $\mathbf{W}_\ell^{\text{sparse}}$ from the initial pruning step, and $\mathbf{X}_{\ell-1}$ collected during the forward propagation of the sparse model for data-based regularization, the re-dense weight $\widehat{\mathbf{W}}_\ell^{\text{re-dense}}$ is obtained by:

$$
\begin{aligned}
&L_{\text{base}}(\mathbf{W}_\ell^{\text{sparse}}) = \left\|\mathbf{W}_\ell^{\text{dense}}\mathbf{X}_{\ell-1} - \mathbf{W}_\ell^{\text{sparse}}\mathbf{X}_{\ell-1}\right\|_2^2, \\
&L_{\text{reg}}(\mathbf{W}_\ell^{\text{sparse}}) = \lambda_1\|\mathbf{W}_\ell^{\text{sparse}}\|_1 + \lambda_2\|\mathbf{W}_\ell^{\text{sparse}}\|_2, \\
&L_{\text{total}}(\mathbf{W}_\ell^{\text{sparse}}) = L_{\text{base}}(\mathbf{W}_\ell^{\text{sparse}}) + L_{\text{reg}}(\mathbf{W}_\ell^{\text{sparse}}), \\
&\widehat{\mathbf{W}}_\ell^{\text{re-dense}} = \operatorname{argmin}_{\mathbf{W}_\ell^{\text{sparse}}}\left(\left(L_{\text{total}}(\mathbf{W}_\ell^{\text{sparse}})\right)\right),
\end{aligned}
\tag{3}
$$

where λ_1 and λ_2 are used to control the ratio of weight-based regularization. The distribution of $\widehat{\mathbf{W}}_\ell^{\text{re-dense}}$ are shown in Figure 1. Firstly, the parameters obtained through re-dense weight reconstruction show a clear three-peaked distribution. This distribution displays a higher sharpness around zero than the original dense model, which is a terrific phenomenon. It indicates that the increased concentration of values around zero makes irrelevant weights easier to identify, a trait referred to as pruning-friendliness (Han et al., 2017). Secondly, the re-dense weight reconstruction yields a model with performance slightly below that of the original dense model but significantly better than the initial sparse model, aligning with our expectations: under the constraints of regularization, it is straightforward that the performance of the re-dense model struggles to maintain consistency with the pre-trained dense model. Appendix A.2 provides a detailed analysis for sparse regularization.

2.3 Second Pruning

Directly adjusting weights in a Sparse-to-Sparse manner seems intuitive for enhancing a sparse model's performance; however, when applied to a model after the initial pruning stage, it only results in minor performance gains on the lightest models (cf., Table 8). This approach simply introduces a first-order loss term to guide the layer-wise optimization, which is insufficient.

Considering the aforementioned challenges, we introduce sparse weight adjusting as the concluding step in the SDS framework. The re-dense model obtained with sparse regularization guidance may inevitably perform inferior to the pre-trained model. As a result, directly pruning it might not be ideal. Therefore, we perform weight adjustment for the second-pruned model. To elaborate, we first prune the re-dense model using the same method employed during the initial pruning, yielding $\mathbf{W}^{\text{sparse-2nd}}$. Subsequently, weight adjusting is conducted utilizing a soft sparse mask:

$$
\begin{aligned}
&L_{\text{base}}(\mathbf{W}_\ell^{\text{sparse-2nd}}) = \left\|\mathbf{W}_\ell^{\text{dense}}\mathbf{X}_{\ell-1} - \mathbf{W}_\ell^{\text{sparse-2nd}}\mathbf{X}_{\ell-1}\right\|_2^2, \\
&\widehat{\mathbf{W}}_\ell^{\text{SDS}} = \mathbf{Mask}_\ell^{\text{soft}} \odot \operatorname{argmin}_{\mathbf{W}_\ell^{\text{sparse-2nd}}}\left(L_{\text{base}}(\mathbf{W}_\ell^{\text{sparse-2nd}})\right),
\end{aligned}
\tag{4}
$$

where the weight alignment target is $\mathbf{W}_\ell^{\text{dense}}$ instead of $\mathbf{W}_\ell^{\text{re-dense}}$, for avoiding the inevitable loss of information in the initial pruning and re-dense process. $\mathbf{X}_{\ell-1}$ is collected from the forward propagation of the second pruned model. $\mathbf{Mask}_\ell^{\text{soft}}$ represents a soft sparse mask, which is dynamically selected by $|\mathbf{W}_\ell^{\text{sparse-2nd}}|$ in each iteration. Due to the inherent awareness of activation information from backpropagation, this magnitude (absmin) (Hagiwara, 1994) mask selection metric can achieve results similar to the elaborate salience metric in SparseGPT and Wanda. In both steps of weight adjustment, the L2 loss is utilized, inherently emphasizing the loss in regions with outliers (Xiao et al., 2023), which plays a pivotal role in the performance of language models. Therefore, outliers can be protected and less affected by weight adjustments. Notably, for most models above three billion parameters, it is possible to do direct one-shot pruning of the re-dense model and exceed the performance of initial pruning. At this point, we choose to *early-exit*, skipping the weight adjustment process and, therefore, more efficient.

As shown in Figure 1, the weights presented after the second pruning became moderate, i.e., the weight distribution of the second-pruned model is smoother and more uniform than that of the initial pruning step, which means that the model parameters have suitable values in different ranges, possessing a stronger ability to adapt to unseen data.

The process of SDS (using SparseGPT as the base pruning method as an example) is shown in Algorithm 1.

3 Experiments

3.1 Experimental Settings

Models. We utilize the widely adopted Open Pre-trained Transformers (OPTs) (Zhang et al., 2022) and LLaMAs (Touvron et al., 2023a,b; Dubey et al., 2024), with focused model sizes ranging from millions to seventy billion parameters. The modules to be pruned are the computationally intensive `self_attn.q_proj`, `self_attn.k_proj`, `self_attn.v_proj`, `self_attn.out_proj`, `fc1`, `self_attn.up_proj`, `self_attn.gate_proj`, `fc2` and `self_attn.down_proj` modules constructed from fully connected layers, which is consistent with baseline.

Calibration. For the data used in calibration, we adhere to the approach outlined in SparseGPT and Wanda, selecting 128 segments of 2048 tokens each from the initial partition of the C4 dataset (Raffel et al., 2020). The C4 dataset, sourced from a broad array of internet text, guarantees that our experiments are zero-shot, as no task-specific information is exposed during our optimization process. More analysis on calibration samples can be found in Appendix A.5.

Datasets and Evaluation. Regarding evaluation metrics, our primary emphasis is on perplexity, which remains a challenging and reliable metric well suited for evaluating the language modeling capability of compressed models (Frantar and Alistarh, 2022; Frantar et al., 2022; Yao et al., 2022). We consider the Raw-WikiText2 (Merity et al., 2017) test set for perplexity validation. To explore the impact of compression on a broad range of downstream tasks, we also provide zero-shot accuracy results for COPA (Wang et al., 2019), Lambada (Paperno et al., 2016), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), RTE (Wang et al., 2018), StoryCloze (Sharma et al., 2018) and Winogrande (Sakaguchi et al., 2019).

Implementation Details. We implement the SDS framework in PyTorch (Paszke et al., 2019) and use the HuggingFace Transformers / Datasets library (Wolf et al., 2020) for managing models and datasets. We follow the conventional method (SparseGPT / Wanda) to prune the pre-trained model in the initial pruning step. In the re-dense weight reconstruction step, we use 128 samples as inputs to perform layer-wise knowledge alignment: the number of alignment epochs is 200, the learning rate is 0.1, the loss function is the L2 loss, and the regularization strategy contains L1 and L2 regularization with a ratio of 0.1. The optimization between adjacent layers is achieved by directly using the output of the initial pruned layer as the input for the next layer, eliminating the need for additional forward propagation to obtain the reconstructed layer's output and, thereby, no accumulation of pruning errors. In the second pruning step, we use the same pruning method to prune the re-dense model and use the same configuration as in the previous step without weight regularization to further adjust the weights of the pruned model with a soft sparse mask (exit early and skip weight adjustment when secondary one-shot pruning outperforms initial pruning). The SDS framework uses the same samples throughout while ensuring that no sample is overloaded; it also coincides with the advantage of requiring just a small amount of samples.

PLM	Dense	Sparsity	SDS workflow		
			$\mathbf{S}_{DS}$	$_{s}\mathbf{D}_{s}$	$_{SD}\mathbf{S}$
OPT-125M	27.66	50%	36.85	31.78	**34.23**
		2:4	60.43	44.46	**51.30**
		4:8	44.77	37.82	**41.66**
OPT-350M	22.01	50%	31.58	24.78	**29.36**
		2:4	51.11	31.58	**46.23**
		4:8	39.59	26.15	**34.18**
OPT-1.3B	14.62	50%	17.46	17.39	**17.07**
		2:4	24.34	20.00	**22.67**
		4:8	20.05	18.06	**19.34**

Table 3: **Perplexity on Raw-WikiText2 among the SDS Workflow.** $\mathbf{S}_{DS}$ represents the initially SparseGPT pruned baseline. $_{s}\mathbf{D}_{s}$ represents the dense model obtained in the re-dense weight reconstruction step. $_{SD}\mathbf{S}$ represents the second round pruned model.

3.2 Performance Variations in the SDS Workflow

Table 3 presents the changes in language modeling perplexity on Raw-WikiText2 after each step of the SDS framework, with SparseGPT as the baseline pruning method and primarily covering several compact OPT models. The focus sparsity configurations include 50% sparsity, 2:4 sparsity, and 4:8 sparsity. After the re-dense step ($_{s}\mathbf{D}_{s}$), the perplexity significantly decreases, averaging around 28.0 across all models, which is closer to the dense

Sparsity	Method	OPT					LLaMA-1		LLaMA-2		LLaMA-3-8B
		125M	350M	1.3B	2.7B	6.7B	7B	13B	7B	13B	
0%	Dense	27.66	22.01	14.62	12.47	10.86	5.68	5.09	5.47	4.88	6.14
50%	SparseGPT	36.85	31.58	17.46	13.48	11.55	7.36	6.21	6.72	6.03	9.51
	SDS-SparseGPT	**34.23**	**29.36**	**17.07**	**13.38**	**11.49**	**7.22**	**6.16**	**6.69**	**5.95**	**9.33**
	Wanda	39.79	41.88	18.51	14.38	11.99	7.26	6.15	6.92	5.97	9.83
	SDS-Wanda	**35.05**	**33.07**	**17.15**	**13.70**	**11.66**	**7.19**	**6.10**	**6.86**	**5.91**	**9.53**
4:8	SparseGPT	44.77	39.59	20.05	14.98	12.56	8.72	7.43	8.49	7.01	12.49
	SDS-SparseGPT	**41.66**	**34.18**	**19.34**	**14.81**	**12.39**	**8.37**	**7.39**	**8.11**	**6.93**	**12.03**
	Wanda	53.97	62.49	22.33	16.80	13.59	8.57	7.41	8.61	7.00	14.56
	SDS-Wanda	**43.58**	**47.31**	**19.82**	**15.45**	**12.74**	**8.39**	**7.33**	**8.55**	**6.94**	**13.43**
2:4	SparseGPT	60.43	51.11	24.34	17.18	14.20	11.32	9.12	10.88	8.74	17.88
	SDS-SparseGPT	**51.30**	**46.23**	**22.67**	**16.78**	**13.81**	**10.48**	**8.91**	**10.18**	**8.69**	**15.43**
	Wanda	82.47	113.17	28.33	21.20	15.99	11.54	9.61	12.14	8.98	24.29
	SDS-Wanda	**59.17**	**73.56**	**23.94**	**17.99**	**14.37**	**10.80**	**9.52**	**11.34**	**8.74**	**19.13**

Table 4: **Perplexity on Raw-WikiText2.** SparseGPT and Wanda form the components of the SDS framework, represented as SDS-SparseGPT and SDS-Wanda, respectively.

Sparsity	Method	OPT					LLaMA-1		LLaMA-2		LLaMA-3-8B
		125M	350M	1.3B	2.7B	6.7B	7B	13B	7B	13B	
0%	Dense	50.82	54.12	60.83	62.81	64.98	66.71	72.78	68.53	72.24	72.86
50%	SparseGPT	48.85	52.33	55.89	61.14	64.32	64.75	70.75	65.78	70.08	69.85
	SDS-SparseGPT	**50.80**	**54.51**	**58.42**	**61.78**	**64.96**	**66.16**	**71.26**	**66.98**	**70.41**	**71.48**
	Wanda	48.46	48.90	56.18	59.36	63.21	66.26	70.51	67.08	70.17	68.06
	SDS-Wanda	**49.78**	**51.40**	**57.58**	**60.92**	**64.11**	**66.89**	**71.41**	**67.58**	**71.24**	**69.60**
4:8	SparseGPT	48.29	49.85	54.94	60.24	63.36	64.40	68.70	65.19	68.29	66.11
	SDS-SparseGPT	**49.67**	**52.25**	**57.92**	**61.48**	**64.06**	**65.61**	**69.19**	**66.05**	**68.82**	**67.46**
	Wanda	46.28	46.41	55.04	58.21	61.58	64.60	67.33	66.21	69.16	62.93
	SDS-Wanda	**47.70**	**48.61**	**56.12**	**59.91**	**63.71**	**65.65**	**69.39**	**66.42**	**69.64**	**64.83**
2:4	SparseGPT	47.56	48.34	53.57	58.48	62.72	62.58	67.69	64.73	66.15	63.00
	SDS-SparseGPT	**49.59**	**50.50**	**56.67**	**59.96**	**63.21**	**63.22**	**68.11**	**65.43**	**66.63**	**63.19**
	Wanda	45.69	44.77	52.86	55.51	61.01	61.58	65.65	61.93	65.16	57.17
	SDS-Wanda	**47.09**	**46.69**	**54.44**	**58.98**	**63.01**	**63.03**	**65.94**	**63.51**	**66.85**	**61.03**

Table 5: **Multitasking Zero-shot Performance.** Accuracy (%) was averaged over seven downstream tasks, including COPA, Lambada (OPTs and LLaMA3 only), OpenbookQA, PIQA, RTE, StoryCloze, and Winogrande.

models (average perplexity of 21.4), with minor performance gap mainly due to regularization effects. Following the second round of pruning ($_{SD}$**S**), the performance improves beyond the initial pruning baseline (**S**$_{DS}$), averaging around 32.9 compared to the initial average of 36.2. This suggests that the re-dense process successfully produces a more pruning-friendly model.

3.3 Performance

Performance on Language Modeling and Zero-shot Benchmarks. We first focus on the typical sparsity configurations, including 50% sparsity for model compression and 2:4 and 4:8 sparsity for both compression and computational acceleration. The results in Tables 4 and 5 demonstrate that the SDS-enhanced models consistently outperform their baselines across all sparsity levels (Table 13 provides detailed accuracy results for each zero-shot downstream task). For instance, in the 50% sparsity setting, SDS-SparseGPT achieves a perplexity of 29.36 on OPT-350M, outperforming SparseGPT's 31.58. At the 4:8 sparsity level, SDS-Wanda reaches an accuracy of 69.39% on LLaMA-13B, surpassing both SparseGPT and Wanda. The advantage of the SDS framework is also evident at the 2:4 sparsity level; SDS-SparseGPT achieves the lowest perplexity of 15.43 on LLaMA-3-8B, while also improving accuracy to 63.19%. These results highlight the robustness of the SDS framework in enhancing the performance of pruned PLMs across

Sparsity	Method	OPT		LLaMA-1				LLaMA-2			LLaMA-3	
		6.7B	13B	7B	13B	30B	65B	7B	13B	70B	8B	70B
0%	Dense	10.86	10.13	5.68	5.09	4.77	3.56	5.47	4.88	3.12	6.14	2.86
60%	SparseGPT	13.42	12.97	10.56	8.39	6.67	5.80	10.23	8.25	5.28	15.34	7.90
	SDS-SparseGPT	**13.12**	**12.69**	**10.02**	**8.31**	**6.58**	**5.69**	**9.91**	**8.12**	**5.23**	**14.77**	**7.49**
	Wanda	15.26	15.89	10.71	8.76	6.55	5.91	10.79	8.40	5.24	23.58	9.63
	SDS-Wanda	**13.95**	**13.12**	**10.11**	**8.67**	**6.41**	**5.84**	**10.35**	**8.32**	**5.18**	**17.83**	**8.21**
70%	SparseGPT	20.58	19.13	27.04	19.03	12.53	10.16	27.76	19.61	9.29	41.38	15.04
	SDS-SparseGPT	**19.73**	**18.75**	**22.73**	**17.54**	**12.09**	**9.49**	**23.58**	**17.46**	**8.90**	**37.50**	**14.16**
	Wanda	168.28	48.90	84.45	54.13	17.32	15.13	75.07	46.17	10.52	126.18	25.35
	SDS-Wanda	**54.42**	**26.65**	**55.83**	**32.67**	**13.85**	**12.72**	**34.02**	**30.47**	**10.38**	**60.51**	**16.96**
80%	SparseGPT	97.32	72.61	184.71	103.37	54.91	33.40	112.47	99.22	28.12	186.68	48.73
	SDS-SparseGPT	**82.19**	**64.64**	**130.08**	**88.76**	**46.64**	**28.56**	**86.25**	**85.64**	**23.47**	**136.65**	**39.96**
	Wanda	3742.93	14463.86	7649.31	3993.33	2248.03	1672.99	2738.57	1540.65	150.60	893.56	203.85
	SDS-Wanda	**1703.81**	**7330.33**	**889.77**	**1188.50**	**248.43**	**259.46**	**276.95**	**697.75**	**91.11**	**330.10**	**119.32**

Table 6: **Perplexity on Raw-WikiText2 under higher sparsity.**

Sparsity	Method	OPT		LLaMA-1				LLaMA-2			LLaMA-3	
		6.7B	13B	7B	13B	30B	65B	7B	13B	70B	8B	70B
0%	Dense	64.98	66.08	66.71	72.78	73.88	74.72	68.53	72.24	75.36	72.86	76.11
60%	SparseGPT	64.09	63.03	64.41	66.83	70.35	73.06	64.34	68.51	74.85	63.95	73.14
	SDS-SparseGPT	**64.52**	**64.57**	**65.72**	**67.20**	**70.74**	**73.43**	**64.95**	**69.45**	**75.20**	**64.52**	**73.88**
	Wanda	60.36	62.14	64.62	67.13	69.69	72.90	63.31	67.99	73.55	58.97	69.33
	SDS-Wanda	**62.93**	**64.01**	**65.09**	**67.81**	**70.26**	**73.42**	**64.51**	**68.53**	**74.54**	**61.42**	**72.60**
70%	SparseGPT	59.68	60.51	55.69	59.49	66.91	70.07	56.66	60.42	70.70	55.00	67.09
	SDS-SparseGPT	**60.50**	**61.12**	**57.73**	**60.46**	**67.98**	**70.40**	**57.93**	**60.90**	**71.49**	**56.02**	**67.97**
	Wanda	49.04	55.77	51.57	55.48	65.35	66.07	49.66	52.24	69.13	48.02	59.33
	SDS-Wanda	**52.73**	**56.27**	**52.51**	**56.38**	**65.89**	**68.50**	**51.50**	**53.84**	**69.43**	**49.03**	**63.53**
80%	SparseGPT	51.90	53.81	49.37	50.62	54.33	56.48	48.42	49.39	59.40	47.92	54.18
	SDS-SparseGPT	**53.27**	**54.73**	**50.23**	**51.77**	**55.41**	**57.41**	**50.24**	**50.19**	**60.40**	**49.01**	**55.85**
	Wanda	47.31	47.88	47.57	48.88	49.09	48.64	47.72	48.56	49.62	48.22	48.84
	SDS-Wanda	**48.85**	**47.98**	**49.04**	**49.09**	**49.45**	**49.20**	**48.98**	**49.45**	**51.31**	**48.83**	**49.61**

Table 7: **Multitasking Zero-shot Performance under higher sparsity.** Accuracy (%) was obtained by zero-shot evaluation and averaging over seven downstream tasks, including COPA, Lambada (OPTs and LLaMA3 only), OpenbookQA, PIQA, RTE, StoryCloze, and Winogrande.

different sparsity levels.

Higher Sparsity Performance. Higher sparsity (60%, 70%, and 80%) implies higher compression gains and increased performance drops. the SDS framework consistently outperforms baseline methods at a higher sparsity constraint, as shown in Tables 6 and 7. For instance, at 60% sparsity, SDS-SparseGPT achieves a perplexity of 12.69 and an accuracy of 64.57% on OPT-13B, outperforming SparseGPT, which records a perplexity of 12.97 and an accuracy of 63.03%. At 70% sparsity, SDS-Wanda achieves a perplexity of 16.96 on LLaMA-3-70B, considerably lower than Wanda's perplexity of 25.35. At 80% sparsity, the performance gap widens further, with SDS-SparseGPT achieving the highest accuracy of 60.40% on LLaMA-2-70B, outperforming Wanda, which suffers from severe degradation at this level. These comparisons underscore the robustness of the SDS framework in maintaining performance under high sparsity, with its benefits becoming increasingly evident as sparsity levels rise.

In summary, our evaluations convincingly demonstrate the robustness and efficacy of the SDS framework across a variety of sparsity configurations. Both language modeling and zero-shot downstream multitask performance metrics affirm the consistent superiority of SDS over the baselines. Therefore, SDS is an efficient and effective pruning method for PLMs.

3.4 Ablation Study

To validate the effectiveness of *the step composition* and *the sparse regularization* of the Sparse-Dense-Sparse (SDS) framework, we conducted a series of ablation experiments as shown in Table 8. The first two rows represent the dense and sparse baseline, respectively.

Rows 3 to 5 verify the effect of only performing one-shot pruning and sparse weight adjustment.

	Method	Wiki.↓	COPA↑	Lamb.↑	BookQ.↑	PIQA↑	RTE↑	Story.↑	Wino.↑	Avg acc.↑
1:	**Dense**	27.66	66	39.16	28.0	62.02	50.18	60.03	50.36	50.82
2:	**S**$_{DS}$	60.43	62	27.55	25.8	57.24	53.79	55.38	51.14	47.56
3:	$_{SD}$**S** w DD	58.63	62	27.03	26.0	58.11	52.35	55.25	50.75	47.36
4:	$_{SD}$**S** w SD	58.56	62	20.96	26.4	58.65	51.62	56.21	50.98	46.69
5:	$_{SD}$**S** w KD	56.82	63	30.04	26.2	58.76	53.79	55.63	49.64	48.15
6:	$_{S}$**DS**	57.98	62	26.68	26.2	59.30	48.74	56.27	50.98	47.17
7:	**SDS** w/o WR	51.96	63	30.04	26.2	58.92	51.95	56.46	51.38	48.29
8:	**SDS** w DD	57.72	62	26.99	26.0	59.30	**54.15**	55.19	**51.46**	47.87
9:	**SDS** w KD	57.32	61	29.15	26.4	59.51	54.01	54.17	51.38	47.95
10:	**SDS** w SD	**51.30**	**65**	**31.57**	**27.8**	59.85	**54.15**	**57.42**	51.46	**49.61**
11:	**SDS** w MSD	52.06	64	30.35	27.0	**60.01**	50.18	57.16	**52.57**	48.75

Table 8: **Comparison of Different Configurations of the SDS Framework.** We compare the language understanding perplexity and accuracy of OPT-125M on eight tasks in a 2:4 sparse configuration. The gray characters represent the skipped steps; DD stands for dense data, which uses the activations generated by the dense model as inputs for weight adjustment; SD stands for sparse data, which uses the activations generated by the sparse model as inputs for weight adjustment; KD stands for KD-aware data, which uses the activations of the model after weight adjustment as inputs for the next layer of weight adjustment; WR represents weight regularization; MSD stands for multiple sparse data, which means that different samples are used for each step of the SDS process.

This approach reflects the effect of pruning in the case where the loss is formally computed instead of the "Hessian approximation". Overall, $_{SD}$**S** was only able to outperform SparseGPT on three tasks. Comparing the different input data used in the $_{SD}$**S** case, $_{SD}$**S** w KD can outperform SparseGPT on seven tasks, which is considered better than choosing the other two data types. Thus it can be concluded that *$_{SD}$**S** mode has limited optimization for SparseGPT* and *selection of data with low loss (KD) is more suitable for $_{SD}$**S** mode than selection of data with high loss (DD or SD).*

Row 6 verifies the effect of the second round pruning of the dense model after injecting it directly with sparse regularization, skipping the initial pruning, i.e. residual sparse characteristics. $_{S}$**DS** outperforms the performance of SparseGPT on five tasks, but it does not yet reach the superior performance of **SDS** w SD. This observation demonstrates that *residual sparse characteristics are effective*.

Rows 7 to 10 verify the role of weight-based and data-based regularization in **SDS**, respectively. Unlike $_{SD}$**S**, SD is a more suitable data choice for **SDS**, and this harder data serves the purpose of regularization while avoiding the challenge of learning hard data in multiple steps. Also, it can be argued that residual sparse characteristics and data regularization dominate in sparse regularization compared to weight regularization. Section A.3 provides an analysis from a distributional perspective.

Row 11 shows the impact of using different samples at each step of SDS. The optimization is closer to SDS w SD, but only two tasks outperform it. This indicates that SDS achieves favorable results and does not necessitate additional samples.

3.5 Efficiency Analysis

To illustrate the enhanced efficiency of pruned models, we present the inference speed of dense and sparse models on AMD CPU. We use the DeepSparse library (Kurtic et al., 2023) and apply 50% unstructured pruning on OPTs and LLaMAs in this experiment. Table 9 indicates that the pruned models can achieve 1.19x $\sim$ 1.87x speedup compared to their dense counterparts. This significant boost in inference speed underscores the critical importance of model pruning in practical applications. More efficiency analysis can be found in Appendix A.7.

4 Related Work

Pruning for Language Model Compression. The surging complexity of Transformer-based language models, which now feature hundreds of billions of parameters, has accentuated the urgent need for effective and efficient model pruning methods (Han et al., 2016, 2015; Hassibi et al., 1993). These pruning methods can be broadly classified into structured and unstructured approaches. Structured

Model	OPT-1.3B		OPT-2.7B		OPT-6.7B		LLaMA-7B	
Metric	Throughput	Latency	Throughput	Latency	Throughput	Latency	Throughput	Latency
Dense	14.1	71.19±1.5	6.9	145.17±3.1	2.9	345.39±7.1	2.3	409.98±5.5
Sparse	16.8	59.55±2.1	9.3	107.07±1.9	4.4	225.67±2.8	4.3	229.52±2.3
Speedup	**1.19x**		**1.35x**		**1.52x**		**1.87x**	

Table 9: **Inference Speed Comparison of Pre- and Post-pruned PLMs** using DeepSparse on AMD Ryzen 7 PRO 5850U @ 1.90 GHz with batchsize = 1 and seqlen = 2048, throughput (batch/sec) and latency (ms/batch) are tested.

pruning is more hardware-friendly, as it directly prunes entire segments of weights, thereby removing consecutive computations (Ma et al., 2023; Liu et al., 2023). Unstructured pruning (sparsification) is also receiving interest, particularly as hardware advancements increasingly support the acceleration of sparse patterns such as 2:4 or 4:8 sparse (Mishra et al., 2021). Techniques such as SparseGPT (Frantar and Alistarh, 2023) extend the OBS (Hassibi et al., 1993) methodology to column-wise prune weights, allowing the modification of values in the unpruned columns to compensate for the pruning errors. Prune-and-Tune (Syed et al., 2023) enhance SparseGPT by incorporating minimal iterative task fine-tuning during the pruning process, demonstrating performance improvements at high sparsity levels. Wanda (Sun et al., 2023) introduces a simple yet effective pruning strategy that prunes weights based on their magnitudes and corresponding activations. OWL (Yin et al., 2024) considers the inhomogeneous distribution of interlayer outliers and further extends Wanda to a non-uniform sparsity distribution, achieving better performance. DS∅T (Zhang et al., 2024) and SPP (Lu et al., 2024), as tuning methods designed for sparse models, can improve the performance of pruned PLMs within limited complexity.

Weight Distribution Optimization. Various techniques have been employed to understand and optimize weight distributions in the quest for more efficient neural networks. The Dense-Sparse-Dense training method (Han et al., 2017) provides a three-step flow: an initial dense training to learn connection weights, a sparsity-inducing phase that prunes unimportant connections, and a final re-dense step. This process improves performance across various network architectures and underscores the importance of parameter distribution in achieving better local optima. Ji et al., 2024a treat the process of Dense-Sparse-Dense as a regularized entirety, improving the model's cross-domain few-shot classification capability. Regularization methods serve as pivotal tools for optimizing the parameter distribution. SPDF (Thangarasa et al., 2023) adopts a language model construction paradigm of sparse pre-training and dense fine-tuning, which both introduce sparse regularization to the training process and improve the training efficiency. Dropout (Srivastava et al., 2014) is a form of ensemble learning of neural networks. It implicitly changes the parameter distribution by randomly zeroing out weights during training, encouraging a sparse representation. Yoshida and Miyato, 2017 focus on constraining the spectral norm of weights matrices to improve the generalization capabilities of neural networks. This method plays a crucial role in shaping the parameter space, making it more amenable to sparse approximations.

In this paper, the proposed Sparse-Dense-Sparse (SDS) framework first regularizes the weights into a pruning-friendly dense distribution and prunes the models, aiming to enhance the language comprehension and multitasking performance of the state-of-the-art conventional pruning methods.

5 Conclusion

We introduced the Sparse-Dense-Sparse (SDS) framework for optimizing pruned pre-trained language models (PLMs), consisting of initial pruning, re-dense weight reconstruction, and a second pruning round. The SDS framework focuses on weight distribution optimization and incorporates sparse regularization elements—including residual sparse characteristics, data-based regularization, and weight-based regularization. As a result, SDS not only enhances the model's pruning friendliness but also achieves state-of-the-art pruning results. Experimental results show that SDS reduces perplexity by 5.16 on Raw-Wikitext2 and enhances accuracy by an average of 3.86% across various zero-shot benchmarks for LLaMA-3-8B compared to Wanda with a 2:4 sparsity configuration.

References

Abhinav Bandari, Lu Yin, Cheng-Yu Hsieh, Ajay Jaiswal, Tianlong Chen, Li Shen, Ranjay Krishna, and Shiwei Liu. 2024. Is C4 dataset optimal for pruning? an investigation of calibration data for LLM pruning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 18089–18099. Association for Computational Linguistics.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Vladimír Boza. 2024. Fast and effective weight update for pruned large language models. *Trans. Mach. Learn. Res.*, 2024.

Peijie Dong, Lujun Li, Zhenheng Tang, Xiang Liu, Xinglin Pan, Qiang Wang, and Xiaowen Chu. 2024. Pruner-zero: Evolving symbolic pruning metric from scratch for large language models. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Elias Frantar and Dan Alistarh. 2022. Optimal brain compression: A framework for accurate post-training quantization and pruning. In *NeurIPS*.

Elias Frantar and Dan Alistarh. 2023. 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.

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

Masafumi Hagiwara. 1994. A simple and effective method for removal of hidden units and weights. *Neurocomputing*, 6(2):207–218.

Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, Bryan Catanzaro, and William J. Dally. 2017. DSD: dense-sparse-dense training for deep neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1135–1143.

Babak Hassibi, David G. Stork, and Gregory J. Wolff. 1993. Optimal brain surgeon and general network pruning. In *Proceedings of International Conference on Neural Networks (ICNN'88), San Francisco, CA, USA, March 28 - April 1, 1993*, pages 293–299. IEEE.

Suzana Herculano-Houzel, Bruno Mota, Peiyan Wong, and Jon H Kaas. 2010. Connectivity-driven white matter scaling and folding in primate cerebral cortex. *Proceedings of the National Academy of Sciences*, 107(44):19008–19013.

Fanfan Ji, Yunpeng Chen, Luoqi Liu, and Xiao-Tong Yuan. 2024a. Cross-domain few-shot classification via dense-sparse-dense regularization. *IEEE Trans. Circuits Syst. Video Technol.*, 34(3):1352–1363.

Yixin Ji, Yang Xiang, Juntao Li, Qingrong Xia, Ping Li, Xinyu Duan, Zhe-Feng Wang, and Min Zhang. 2024b. Beware of calibration data for pruning large language models. *CoRR*, abs/2410.17711.

Eldar Kurtic, Denis Kuznedelev, Elias Frantar, Michael Goin, and Dan Alistarh. 2023. Sparse fine-tuning for inference acceleration of large language models. *CoRR*, abs/2310.06927.

Tailin Liang, John Glossner, Lei Wang, and Shaobo Shi. 2021. Pruning and quantization for deep neural network acceleration: A survey. *CoRR*, abs/2101.09671.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Ré, and Beidi Chen. 2023. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 22137–22176. PMLR.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Xudong Lu, Aojun Zhou, Yuhui Xu, Renrui Zhang, Peng Gao, and Hongsheng Li. 2024. Spp: Sparsity-preserved parameter-efficient fine-tuning for large language models. In *Forty-first International Conference on Machine Learning*.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *CoRR*, abs/2305.11627.

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

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*.

Asit K. Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. 2021. Accelerating sparse deep neural networks. *CoRR*, abs/2104.08378.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. *CoRR*, abs/1606.06031.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *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 8024–8035.

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. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*.

Rishi Sharma, James Allen, Omid Bakhshandeh, and Nasrin Mostafazadeh. 2018. Tackling the story ending biases in the story cloze test. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 752–757, Melbourne, Australia. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

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

Aaquib Syed, Phillip Huang Guo, and Vijaykaarti Sundarapandiyan. 2023. Prune and tune: Improving efficient pruning techniques for massive language models. In *The First Tiny Papers Track at ICLR 2023, Tiny Papers @ ICLR 2023, Kigali, Rwanda, May 5, 2023*. OpenReview.net.

Vithursan Thangarasa, Abhay Gupta, William Marshall, Tianda Li, Kevin Leong, Dennis DeCoste, Sean Lie, and Shreyas Saxena. 2023. SPDF: sparse pre-training and dense fine-tuning for large language models. In *Uncertainty in Artificial Intelligence, UAI 2023, July 31 - 4 August 2023, Pittsburgh, PA, USA*, volume 216 of *Proceedings of Machine Learning Research*, pages 2134–2146. PMLR.

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

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. 2023a. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti

Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *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.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Miles Williams and Nikolaos Aletras. 2024. On the impact of calibration data in post-training quantization and pruning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 10100–10118. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*, pages 38–45. Association for Computational Linguistics.

Guangxuan Xiao, Ji Lin, Mickaël 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, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 38087–38099. PMLR.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. In *NeurIPS*.

Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Gen Li, AJAY KUMAR JAISWAL, Mykola Pechenizkiy, Yi Liang, et al. 2024. Outlier weighed layerwise sparsity (owl): A missing secret sauce for pruning llms to high sparsity. In *Forty-first International Conference on Machine Learning*.

Yuichi Yoshida and Takeru Miyato. 2017. Spectral norm regularization for improving the generalizability of deep learning. *CoRR*, abs/1705.10941.

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. 2022. OPT: open pre-trained transformer language models. *CoRR*, abs/2205.01068.

Yuxin Zhang, Lirui Zhao, Mingbao Lin, Yunyun Sun, Yiwu Yao, Xingjia Han, Jared Tanner, Shiwei Liu, and Rongrong Ji. 2024. Dynamic sparse no training: Training-free fine-tuning for sparse llms. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

A Appendix

A.1 The sparse-dense-sparse framework

Algorithm 1 The Sparse-Dense-Sparse (SDS) Framework

Input: Pre-trained dense model $\mathbb{W}^{\text{dense}} = \{\mathbf{W}_1^{\text{dense}}, \mathbf{W}_2^{\text{dense}}, ..., \mathbf{W}_L^{\text{dense}}\}$
Output: Final pruned model $\widehat{\mathbb{W}}^{\text{SDS}} = \{\widehat{\mathbf{W}}_1^{\text{SDS}}, \widehat{\mathbf{W}}_2^{\text{SDS}}, ..., \widehat{\mathbf{W}}_L^{\text{SDS}}\}$

Initial Pruning

Require: Original unlabeled samples $\mathbf{X}$; sparsity
for each $\mathbf{W}^{\text{dense}}$ **in** $\mathbb{W}^{\text{dense}}$ **do**
  $\mathbf{W}^{\text{sparse}} = \text{empty}(\mathbf{W}^{\text{dense}})$
  $\mathbf{H} = \mathbf{X}\mathbf{X}^\top$
  for $c = 1$ **to** column_size($\mathbf{W}^{\text{dense}}$) **do**
    $s = \text{sort}\left(\frac{\mathbf{W}_{:,c}^{\text{dense}\,2}}{[\mathbf{H}^{-1}]_{c,c}^2}\right)$
    $\mathbf{M} = \mathbb{1}\,(s > \text{sparsity})$
    $\mathbf{W}_{:,c}^{\text{sparse}} = \mathbf{M}_{:,c} \odot \mathbf{W}_{:,c}^{\text{dense}}$ ▷ Prune one column with mask $\mathbf{M}$
    $\mathbf{W}_{:,c+1:}^{\text{dense}} = \mathbf{W}_{:,c+1:}^{\text{dense}} - \frac{\mathbf{W}_{:,c}^{\text{dense}} - \mathbf{W}_{:,c}^{\text{sparse}}}{[\mathbf{H}^{-1}]_{c,c}} \cdot [\mathbf{H}^{-1}]_{c,c:}$ ▷ Error Compensation
  end for
  $\mathbf{X} \leftarrow \mathbf{W}^{\text{sparse}}\mathbf{X}$ ▷ Error Accumulation
end for

Re-dense Weight Reconstruction

Require: Pre-trained dense model $\mathbb{W}^{\text{dense}} = \{\mathbf{W}_1^{\text{dense}}, \ldots, \mathbf{W}_L^{\text{dense}}\}$;
  Initial pruned sparse model $\mathbb{W}^{\text{sparse}} = \{\mathbf{W}_1^{\text{sparse}}, \ldots, \mathbf{W}_L^{\text{sparse}}\}$;
  Original unlabeled samples $\mathbf{X}$; Learning rate η;
  L1 regularization ratio λ_1; L2 regularization ratio λ_2
for $\ell = 1$ **to** L **do**
  while not converged **do**

$$\mathbf{W}_\ell^{\text{re-dense}(t)} = \mathbf{W}_\ell^{\text{re-dense}(t-1)} - \eta\nabla\left\|\mathbf{W}_\ell^{\text{dense}}\mathbf{X} - \mathbf{W}_\ell^{\text{re-dense}(t-1)}\mathbf{X}\right\|_2^2 - \eta\lambda_1\nabla\left\|\mathbf{W}_\ell^{\text{re-dense}(t-1)}\right\|_1 - \eta\lambda_2\nabla\left\|\mathbf{W}_\ell^{\text{re-dense}(t-1)}\right\|_2^2$$

  end while
  $\mathbf{X} \leftarrow \mathbf{W}_\ell^{\text{sparse}}\mathbf{X}$ ▷ Error Accumulation
end for

Second pruning: sparse weight adjustment

Require: Pre-trained dense model $\mathbb{W}^{\text{dense}} = \{\mathbf{W}_1^{\text{dense}}, \ldots, \mathbf{W}_L^{\text{dense}}\}$;
  Re-dense trained model $\mathbb{W}^{\text{re-dense}} = \{\mathbf{W}_1^{\text{re-dense}}, \ldots, \mathbf{W}_L^{\text{re-dense}}\}$;
  Original unlabeled samples $\mathbf{X}$;
  Learning rate η;
  Sparsity
Repeat the pruning process and yield $\mathbb{W}^{\text{sparse-2nd}} = \{\mathbf{W}_1^{\text{sparse-2nd}}, ..., \mathbf{W}_L^{\text{sparse-2nd}}\}$
for $\ell = 1$ **to** L **do**
  while not converged **do**
    $s = \text{sort}\left(\left|\mathbf{W}_\ell^{\text{SDS}(t)}\right|\right)$
    $\mathbf{M} = \mathbb{1}\,(s > \text{sparsity})$

$$\mathbf{W}_\ell^{\text{SDS}(t)} = \mathbf{M} \odot \left(\mathbf{W}_\ell^{\text{SDS}(t-1)} - \eta\nabla\left\|\mathbf{W}_\ell^{\text{dense}}\mathbf{X} - \mathbf{W}_\ell^{\text{SDS}(t-1)}\mathbf{X}\right\|_2^2\right)$$

  end while
  $\mathbf{X} \leftarrow \mathbf{W}_\ell^{\text{sparse-2nd}}\mathbf{X}$ ▷ Error Accumulation
end for

A.2 Error Accumulation and Data-based Regularization

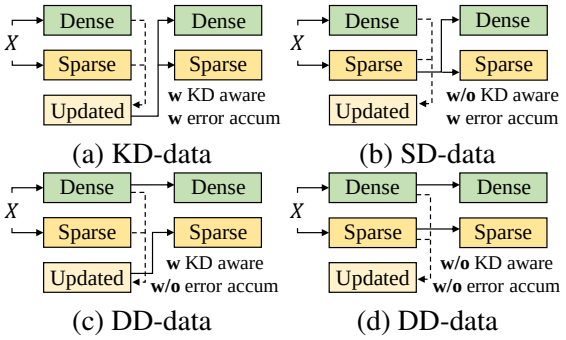

Figure 2: **Four Data Selection Paradigms in Weight Adjustment.** Straight lines represent forward propagation and dashed lines represent knowledge distillation.

The input data used in weight adjustment can be categorized in two ways: whether to perform error accumulation and whether to be aware of the knowledge distillation (KD) process. Figure 2 presents four different data selection patterns for weight adjustment.

(a) Weight adjustment with KD aware and error accumulation, this paradigm corresponds to *KD-data* in our ablation study (cf., Subsection 3.4): after applying KD to the sparse layer, a subsequent forward propagation is needed to generate inputs for the next layer. These inputs are solely based on the former layer's outputs, thus accumulating errors. Since KD aims to reduce loss, this extra forward propagation simplifies the data, making it easier for the subsequent layer to learn. **(b)** Weight adjustment with error accumulation but without KD aware, this paradigm corresponds to *SD-data* in our ablation study: unlike paradigm (a), this approach abandons the additional forward propagation to account for changes in the layer updated by KD. This results in the next layer of learning from data corresponding to a higher loss, making learning more challenging than in paradigm (a). KD-data and SD-data, which use sparse model activations as inputs, further provide a priori on the importance of the parameters. **(c)** and **(d)** are two ways of adjusting the weights without accumulating errors. The presence or absence of KD awareness has a minimal impact on either, as the optimization direction is constrained by the dense model in both cases. The *DD-data* paradigm in our ablation study employs paradigm (d).

From the perspective of data difficulty, *DD-data* is the most difficult because it requires each layer to compensate for the errors accumulated in all previous layers. This difficulty is more prominent in the KD process under sparsity constraints. In the ablation study (cf., Subsection 3.4), the optimization of the sparse model using *DD-data* cannot achieve the best results, verifying the above observation. *KD-data* is the easiest because the weights of the sparse model are updated in the direction of lower loss during knowledge distillation. The use of *KD-data* has yielded relatively good results only in single-step optimization of the sparse model due to the fact that simple data carries less data regularization and a relatively low upper bound for optimization. *SD-data* is relatively moderate in difficulty and comes with sparse regularization and hence achieved an ideal result in SDS's optimization of the sparse model. The reason why *SD-data* did not achieve an ideal result in the single-step optimization could be the challenge of the difficult data.

A.3 Distribution Analysis

Figure 3 visualizes the impact of several pertinent optimizations performed on pruned PLMs from the perspective of weight distribution changes.

Magnitude-based one-shot pruning method is ineffective on PLMs primarily because it focuses only on the absolute value of the weights. This simplistic approach tends to create a truncated bimodal distribution of the model weights, concentrating them at extreme positive and negative values. Distribution truncation can lead to model instability, as removing near-zero weights disrupts the model's ability to make subtle, nuanced adjustments. Due to the large amount of information lost in the pruning process, the model's performance can only be recovered to a limited extent after weight adjustment. In contrast, modern pruning methods like SparseGPT take into account higher-order rather than zero-order information, which manages to maintain an untruncated bimodal distribution similar to what magnitude pruning plus subsequent weight adjustment would achieve. However, they do it in a single step and are able to achieve better performance.

As shown in Figure 3b, the model has a relatively sharp bimodal peak in its distribution after being pruned by SparseGPT, which challenges the model's generalization ability, optimization space and stability. Direct adjustment of the pruned model's weights yields limited performance and optimization of the weight distribution. Therefore,

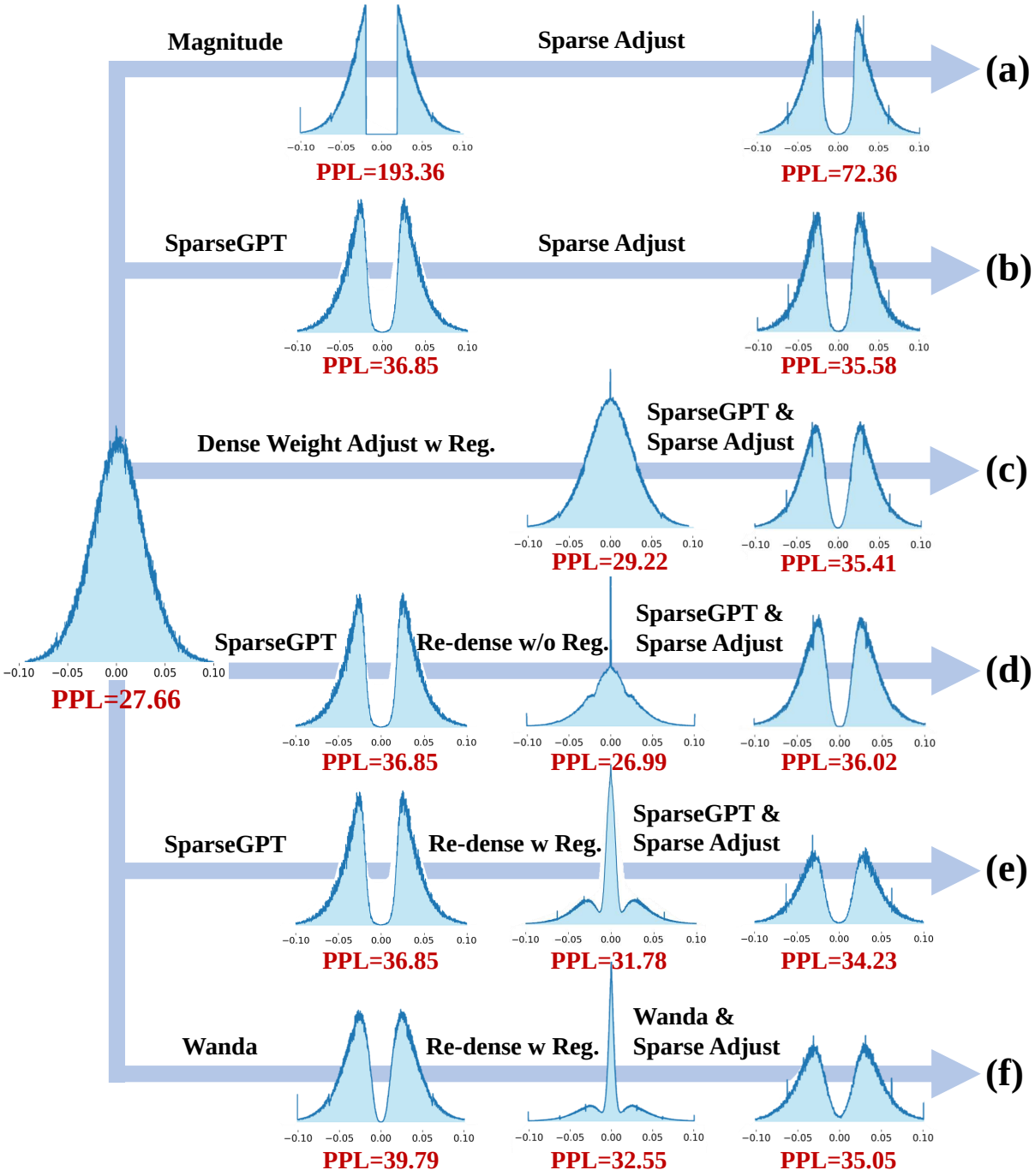

Figure 3: **Changes in Distributions During Optimization of Pruned PLMs.** The distribution observations are from the last layer of OPT-125m with 50% pruning. **(a)** represents the process of first pruning the model by magnitude (*absmin*) (Hagiwara, 1994) and then optimizing the pruned model using SD-data. **(b)** represents the SDS w KD in the ablation study (cf., Subsection 3.4). **(c)** represents the SDS. **(d)** represents the **SDS** w KD. **(e)** represents the **SDS** w SD. **(f)** represents the **SDS** w SD and with Wanda as the pruning method. Zero values are omitted in sparse weight distributions for better clarity.

it is necessary to consider the SDS process.

Before attempting the SDS process, Figure 3c and Figure 3d show the trend of weight distribution changes for only injecting regular regularization or residual sparse characteristics into the model, respectively. Both find a new dense solution to some extent: a dense model with a smoother distribution and more zeros can be found by using data-based regularization and weight-based regularization for dense weight adjustment, and a dense model that converges to a multi-peaked distribution with more zeros is obtained after re-dense reconstruction of

Model	OPT-1.3B		OPT-2.7B		OPT-6.7B		LLaMA(1&2)-7B	
Metric	Throughput	Latency	Throughput	Latency	Throughput	Latency	Throughput	Latency
Dense	29.5	29.56±0.9	14.7	68.03±0.8	5.5	181.21±0.9	5.3	189.83±0.7
Sparse	39.1	33.88±0.8	22.9	43.65±0.5	11	90.65±0.9	12.4	80.75±0.6
Speedup	**1.33x**		**1.56x**		**2.0x**		**2.34x**	

Table 10: **Inference Speed Comparison of Pre- and Post-pruned PLMs** using DeepSparse on Intel(R) Xeon(R) Platinum 8372C CPU @ 3.20GHz with batchsize = 1 and seqlen = 2048, throughput (batch/sec) and latency (ms/batch) are tested.

the sparse model without regular regularization. After a second round of pruning, both approaches lead to a recovery in the model's performance. However, they are not as effective as the SDS process that uses a combination of data-based regularization, weight-based regularization, and residual sparse characteristics, as shown in Figure 3e and Figure 3f. This illustrates the effectiveness and mutual reinforcement effect of regularization techniques in the SDS framework. An interesting phenomenon is that when regular regularization is not used, it is possible to reconstruct the pruned model to equal or even higher performance than the pre-trained dense model. This is perhaps due to the absence of regularization techniques, which allowed the re-dense model to overfit the behavior of the pre-trained dense model. The limited performance improvement of the re-dense model after a second round of pruning also supports the above deduction.

A.4 SDS Steadily Optimizes Multiple Pruning Methods

Method	PPL ↓ (Dense: 6.14)			ACC. ↑ (Dense: 72.86)		
	2:4	4:8	70%	2:4	4:8	70%
SparseGPT	17.88	12.49	41.38	63.00	66.11	55.00
SDS-SparseGPT	**15.43**	**12.03**	**37.50**	**63.19**	**67.46**	**56.02**
Wanda	24.29	14.56	126.18	57.17	62.93	48.02
SDS-Wanda	**19.13**	**13.43**	**60.51**	**61.03**	**64.83**	**49.03**
Admm	13.71	10.56	29.39	62.48	64.56	55.30
SDS-Admm	**13.59**	**10.49**	**26.94**	**63.53**	**65.23**	**55.88**
Zero	22.99	12.98	260.21	59.15	62.95	48.62
SDS-Zero	**16.47**	**11.51**	**69.38**	**61.86**	**64.22**	**50.05**
DS∅T	21.97	14.01	119.35	59.26	63.27	48.32
SDS-DS∅T	**17.28**	**12.64**	**57.60**	**61.89**	**64.17**	**49.82**

Table 11: **SDS Steadily Optimizes Multiple Pruning Methods on LLaMA-3-8B.** The performance of various pruning methods, including AdmmPruner (Admm), PrunerZero (Zero), and DS∅T (with Wanda as the base pruning method), is validated with and without SDS optimization. The results show that SDS consistently improves both perplexity (PPL) and accuracy (ACC., averaged across COPA, Lambada, OpenbookQA, PIQA, RTE, StoryCloze, and Winogrande), demonstrating its effectiveness and generalizability in optimizing a wide range of pruning methods.

Table 11 demonstrates that SDS has a significant optimizing effect on a wide range of pruning methods, including AdmmPruner (Boza, 2024), PrunerZero (Dong et al., 2024), and even the post-pruning fine-tuning method DS∅T (Zhang et al., 2024). For instance, AdmmPruner achieves a low PPL of 29.39 after 70% pruning, yet SDS further enhances its post-pruning accuracy, reducing the PPL to 26.94. The consistent improvements across diverse methods highlight SDS's strong generalizability. SDS is a versatile optimization technique that improves performance across various models and pruning strategies.

A.5 SDS Steadily Optimizes Pruning Effect with Different Calibration Samples

Some works (Williams and Aletras, 2024; Ji et al., 2024b; Bandari et al., 2024) have found that calibration samples affect language model compression. Here we try different calibration datasets to verify the effect of calibration sample variation on SDS optimization effect, as shown in Table 12.

Calibration	Method	PPL ↓ (Dense: 6.14)		ACC. ↑ (Dense: 72.86)	
		2:4	4:8	2:4	4:8
C4	SparseGPT	17.88	12.49	63.00	**66.11**
	SDS-SparseGPT	**15.43**	**12.03**	**63.19**	**67.46**
	Wanda	24.29	14.56	57.17	62.93
	SDS-Wanda	**19.13**	**13.43**	**61.03**	**64.83**
Wiki2	SparseGPT	12.26	9.88	62.35	65.80
	SDS-SparseGPT	**11.31**	**9.20**	**63.01**	**66.93**
	Wanda	21.58	13.37	56.87	61.67
	SDS-Wanda	**17.57**	**12.19**	**61.14**	**64.32**
Pile	SparseGPT	18.71	13.66	63.10	66.08
	SDS-SparseGPT	**15.89**	**12.79**	**63.32**	**67.55**
	Wanda	24.97	14.88	57.29	63.22
	SDS-Wanda	**18.88**	**14.15**	**60.94**	**65.01**

Table 12: **SDS Steadily Optimizes Pruning Effort with Different Calibration Samples on LLaMA-3-8B.** The results show that SDS consistently improves both perplexity (PPL) and accuracy (ACC., averaged across COPA, Lambada, OpenbookQA, PIQA, RTE, StoryCloze, and Winogrande) regardless of the calibration dataset chosen, demonstrating the robustness of SDS to calibration samples.

The results show that SDS improves both perplexity (PPL) and accuracy (ACC.) on LLaMA-

Sparsity	Method	PPL	Lamb.	COPA	BookQ	PIQA	RTE	Story.	WinoG.	Acc AVG
0	-	6.14	75.55	89.00	45.00	80.90	67.51	78.87	73.16	72.86
50%	SparseGPT	9.51	74.18	88.00	**43.00**	**78.07**	59.84	75.62	70.27	69.85
	SDS-SparseGPT	**9.33**	**74.50**	**91.00**	41.40	77.04	**69.31**	**76.56**	**70.56**	**71.48**
	Wanda	9.83	72.62	84.00	40.00	76.82	58.84	73.01	71.11	68.06
	SDS-Wanda	**9.53**	**76.50**	**88.00**	**40.80**	**77.20**	**59.93**	**73.65**	**71.82**	**69.60**
4:8	SparseGPT	12.49	70.18	82.00	**36.40**	**74.76**	58.12	72.18	68.14	67.11
	SDS-SparseGPT	**12.03**	**70.77**	**86.00**	**36.40**	74.54	**62.45**	**73.77**	**68.27**	**67.46**
	Wanda	14.56	63.05	81.00	**35.20**	72.36	**53.07**	69.19	66.61	62.93
	SDS-Wanda	**13.43**	**68.68**	**85.00**	35.00	**73.12**	52.71	**71.29**	**68.03**	**64.83**
2:4	SparseGPT	17.88	64.62	**82.00**	**35.20**	**71.76**	53.07	70.53	63.85	63.00
	SDS-SparseGPT	**15.43**	**65.22**	**82.00**	33.20	**71.76**	**54.15**	**70.66**	**65.35**	**63.19**
	Wanda	24.29	45.95	76.00	32.00	68.28	52.71	64.86	60.38	57.17
	SDS-Wanda	**19.13**	**58.59**	**79.00**	**32.80**	**70.02**	**57.04**	**67.54**	**62.19**	**61.03**

Table 13: **Detailed Zero-Shot Results on LLaMA-3-8B.** SDS outperforms baselines on most individual zero-shot downstream tasks, demonstrating higher accuracy and strong generalizability.

3-8B. At 2:4 sparsity, on the C4 dataset, SDS-SparseGPT reduces PPL from 17.88 to 15.43 and increases accuracy from 63.00 to 63.19. On Pile, it lowers PPL from 18.71 to 15.89 and boosts accuracy from 63.10 to 63.32. These results highlight SDS's effectiveness in optimizing pruning across different calibration datasets.

A.6 Detailed Zero-Shot Downstream Results

Table 13 provides a detailed breakdown of the zero-shot accuracy on individual downstream tasks for LLaMA-3-8B. SDS consistently outperforms baseline methods across various pruning strategies. For example, at 50% sparsity, SDS-SparseGPT achieves a lower perplexity of 9.33 compared to the baseline's 9.51, while improving accuracy on tasks like COPA from 88.00 to 91.00 and on RTE from 59.84 to 69.31. Similarly, SDS-Wanda shows improvements at 2:4 sparsity, reducing perplexity from 24.29 to 19.13, and boosting accuracy on tasks such as StoryCloze from 60.38 to 62.19. These results demonstrate that SDS enhances performance across multiple zero-shot downstream tasks, highlighting its effectiveness in optimizing pruning methods while ensuring strong generalizability.

A.7 SDS Efficiency Analysis

Table 10 further illustrate the enhanced **efficiency of pruned pre-trained language models (PLMs) on CPUs**. This table meticulously enumerates the inference latency improvements of PLMs subjected to 50% unstructured pruning on Intel CPU. The results indicate that the pruned models can achieve a maximum speed increase of up to 2.3 times compared to their unpruned, dense counterparts. This significant boost in inference speed underscores the critical importance of model pruning in efficiency increase.

Tables 14 and 15 respectively display the **kernel-level speedup ratio** and the **end-to-end level speedup ratio for the entire network**, both achieved through the utilization of semi-structured sparse matrix multiplication (SpMM) kernels supported by the CUTLASS library on NVIDIA Ampere and newer GPUs. These findings indicate that operations using sparse matrices can surpass the efficiency of dense operations, achieving more than a twofold increase in speed. Furthermore, even when considering the broader context of end-to-end speedups, the pruned model is observed to outperform the dense model, with potential speed enhancements reaching up to 1.2 times without other optimization. This data underscores the tangible advantages of pruning methodologies, particularly in contexts where computational resources are constrained.

Weight	Q/K/V/Out	Up/Gate/FC1	Down/FC2
Dense	1.574	9.956	12.705
Sparse	1.077	6.026	6.002
Speedup	**1.46x**	**1.65x**	**2.12x**

Table 14: **CUTLASS 2:4 Sparse Kernel Speedup** on NVIDIA A100 PCIe 40GB with seqlen = 1024 and hidden_size = 12288, latency (ms) tested.

Execution Efficiency of the SDS Framework. Table 16 demonstrates the time required to perform SDS optimization (no consideration of early exit scenarios, c.f., Subsection 2.3). The time consumption for layer-serial SDS optimization using

Batchsize	OPT-1.3B			OPT-2.7B			LLaMA-7B		
	Dense	Sparse	Speedup	Dense	Sparse	Speedup	Dense	Sparse	Speedup
1	10999	12193	**1.11x**	8568	8581	**1.00x**	5866	6612	**1.12x**
4	15995	17889	**1.12x**	10710	12450	**1.16x**	6889	7723	**1.12x**
8	18419	19950	**1.08x**	11339	13105	**1.16x**	7029	7637	**1.09x**
16	18970	21909	**1.15x**	10251	12946	**1.26x**	6270	8033	**1.28x**

Table 15: **CUTLASS 2:4 Sparse End-to-end Speedup** on NVIDIA A100 PCIe 40GB with seqlen in [512, 1024, 2048], throughput (tokens/s, averaged) tested.

Type	1.3B	2.7B	7B	13B
Serial	$\sim$ 3.3 hours	$\sim$ 6.4 hours	$\sim$ 12.9 hours	$\sim$ 25.0 hours
Parallel	$\sim$ 26 minutes	$\sim$ 41 minutes	$\sim$ 1.7 hours	$\sim$ 3.1 hours

Table 16: **Time Consuming of SDS Optimization.** There is time consumption of single-device serial optimization and multi-device parallel optimization as the weight adjustment type.

a single NVIDIA V100 32GB GPU ranges from 3 to 25 hours as the model grows larger. Since SDS optimization uses *SD-data* (cf., Section A.2), which is available in advance, and SDS does not need to perform an additional forward propagation for each optimized layer, each layer in the network can perform optimization **in parallel**. On eight V100 GPUs, we optimize layers within individual GPUs serially and layers between GPUs in parallel, so it only took us from 26 minutes to 3 hours to perform the SDS optimization. Figure 4 shows the wall-clock running time of SDS-SparseGPT on LLaMA-7B and its detailed time decomposition.

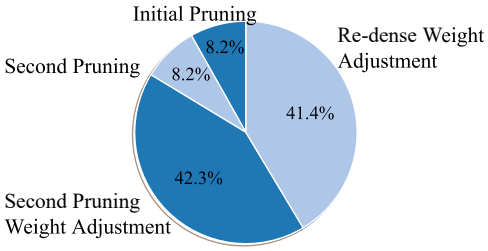

Figure 4: Time Decomposition for SDS-SparseGPT Optimization on LLaMA-7B (1.68 hours in total).

A.8 Limitation

The Sparse-Dense-Sparse (SDS) framework improves pruned PLMs through the perspective of weight distribution optimization. Our approach surpasses the previous SOTA methods including SparseGPT and Wanda under the same sparsity configuration. One limitation is that our sparsity optimization process consumes more computation overhead. However, we note that the introduced optimization time is still small compared to training a language model. Once the sparse model is obtained and deployed, one can benefit from the acceleration on the specific hardware. The effect of adopting different base pruning methods on SDS and the efficiency enhancement of the SDS process will be our future research directions.