DSEE: Dually Sparsity-embedded Efficient Tuning of Pre-trained Language Models

Xuxi Chen[1], Tianlong Chen[1], Weizhu Chen[2], Ahmed Hassan Awadallah[2]
Zhangyang Wang[1], Yu Cheng[2]
[1]University of Texas at Austin, [2]Microsoft Corporation
{xxchen,tianlong.chen,atlaswang}@utexas.edu
{wzchen,hassanam,yu.cheng}@microsoft.com

Abstract

Gigantic pre-trained models have become central to natural language processing (NLP), serving as the starting point for fine-tuning towards a range of downstream tasks. However, two pain points persist for this paradigm: (a) as the pre-trained models grow bigger (e.g., 175B parameters for GPT-3), even the fine-tuning process can be time-consuming and computationally expensive; (b) the fine-tuned model has the same size as its starting point by default, which is neither sensible due to its more specialized functionality, nor practical since many fine-tuned models will be deployed in resource-constrained environments. To address these pain points, we propose a framework for resource- and parameter-efficient fine-tuning by leveraging the sparsity prior in both weight updates and the final model weights. Our proposed framework, dubbed **D**ually **S**parsity-**E**mbedded **E**fficient Tuning (DSEE), aims to achieve two key objectives: (i) *parameter efficient fine-tuning* - by enforcing sparsity-aware low-rank updates on top of the pre-trained weights; and (ii) *resource-efficient inference* - by encouraging a sparse weight structure towards the final fine-tuned model. We leverage sparsity in these two directions by exploiting both unstructured and structured sparse patterns in pre-trained language models via a unified approach. Extensive experiments and in-depth investigations, with diverse network backbones (i.e., BERT, RoBERTa, and GPT-2) on dozens of datasets, consistently demonstrate impressive parameter-/inference-efficiency, while maintaining competitive downstream performance. For instance, DSEE saves about 25% inference FLOPs while achieving comparable performance, with 0.5% trainable parameters on BERT. Codes are available at https://github.com/VITA-Group/DSEE.

1 Introduction

Most recent NLP applications have been following the pre-train then fine-tune paradigm, starting from a gigantic pre-trained model and fine-tuning it towards downstream tasks. Conventional *fine-tuning* works through updating all of the parameters in the pre-trained model. However, as the size of pre-trained models grows, updating all parameters becomes less feasible in most practical scenarios, due to the expensive memory and computational requirements. For example, $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2019) has 110M trainable parameters, while GPT-2 (Radford et al., 2019) has up to 1.5B and the largest version of GPT-3 (Radford et al., 2019) has an astonishing 175B trainable parameters. As such, conventional fine-tuning of the larger models could require hundreds of GPU hours. Another downside of this paradigm is that it requires storing as many parameters as in the large-scale pre-trained models for each downstream task, which poses impediments to the deployment in real-world resource-constrained environments.

One solution to address the extensive resource requirement of conventional fine-tuning is model pruning (LeCun et al., 1990; Han et al., 2015; Ren et al., 2018; He et al., 2017; Liu et al., 2017), where unnecessary weights are eliminated to shrink the model size. For example, Chen et al. (2021b) leverages ℓ_1 regularization to remove insignificant attention heads and gains $35 \sim 45\%$ training time with comparable performance; Chen et al. (2021a); Dao et al. (2022) leverage sparse matrices with fixed structures to reduce pretrained models' sizes. All these studies indicate the rise of sparsity naturally during fine-tuning a general-purpose pre-trained model, to some specialized downstream functionality. One potential interpretation, of why sparsity arises, is that different subsets of the parameters may be responsible for different downstream tasks and data domains (Sanh et al., 2020). However, identifying appropriate sparse masks can be burdensome: fine-tuning a large pre-trained language model like GPT-3 for just one step consumes at least 1.2TB of VRAM and requires 96 pieces of

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 8208–8222

NVIDIA Tesla (Hu et al., 2021), and these methods either require access to pre-trained weights or introduce additional learnable coefficients (such as importance scores of attention heads).

A parallel alternative is to design *parameter-efficient* fine-tuning algorithms, which aim at optimizing a small portion of weights while fixing most of them when fine-tuning on downstream tasks. Pioneering works along this line, which utilize adapters (Houlsby et al., 2019), learnable embeddings (Li and Liang, 2021; Liu et al., 2021), low-rank decomposition (Hu et al., 2021) or their combination (He et al., 2021), can significantly reduce the number of trainable parameters while preserving good fine-tuning performance. Although these methods can substantially improve the storage and deployment efficiency of models, there are two major hurdles: (i) they does not yield any inference efficiency gains since the full pre-trained weights are still required to calculate outputs; and (ii) current methods assume the updates on pretrained weights to be either sparse (Guo et al., 2020) or low-rank (Hu et al., 2021), yet those assumptions might be oversimplified (Yu et al., 2017) and overly restricted to allow for effective updates. These observations have inspired us to explore better parameter-efficiency methods.

To improve both resource- and parameter-efficiency during model fine-tuning, we explicitly draw on the prior of sparsity for *both weight updates and the final weights*, and establish a *dually sparsity-embedding efficient tuning* (**DSEE**) framework. Starting from a pre-trained model, DSEE first adopts a sparsity-aware low-rank weight update to achieve *parameter-efficiency* of the fine-tuning process; and then enforces a sparse weight structure directly from weight updates by masking to achieve *resource-efficiency* of the fine-tuned model at inference time. Our contributions can be summarized as follows:

- We propose the dually sparsity-embedding efficient tuning, which unifies sparsity-aware parameter-efficient weight update and sparse pre-trained weight in fine-tuning gigantic pre-trained models. It is the first attempt toward jointly optimizing both parameter-efficiency of the fine-tuning process and the resource-efficiency of the fine-tuned model.

- Both unstructured and structured sparse priors are investigated in our proposed DSEE algorithm. For weight updates, the injected sparsity prior enhances existing parameter-efficient update schemes (e.g., low-rank decomposition). As for the final weights, we draw superior sparse masks, either unstructured or structured, directly from the weight updates, which requires neither additional parameters nor access to the pre-trained weights and saves the sparsification cost.

- Extensive experiments demonstrate the effectiveness of our proposal across various representative pre-trained language models (BERT, GPT-2, and RoBERTa) and on diverse evaluation benchmarks (E2E, DART, WebNLG, and GLUE). On GPT-2, our methods can achieve a BLUE score of $\{69.5,54.9,47.5\}$ with 0.1% of trainable parameters on {E2E, WebNLG, DART} with 20% parameter removed in pre-trained weights. On BERT, DSEE can fine-tune only 0.5% parameters and save about 25% inference FLOPs, while losing less than 2% performance.

2 Related Work

Pruning and Sparsification Pruning is a classical model compression technique that can reduce the number of parameters, which can bring training and inference efficiency. Researchers have proposed several pruning methods for pre-trained language models: McCarley et al. (2019); Chen et al. (2021b) pruned attention heads that had less contribution during finetuning; Sanh et al. (2020) proposed a pruning criterion targeting the weight change after training, which suits the transfer learning better; Wang et al. (2020) incorporated low-rank factorization and ℓ_0 regularization for pruning. Recently, there is a series of sparsification works that utilize sparse masks with specific structures, called Butterflies, and achieve high efficiency in pretraining models (Chen et al., 2021a) or fine-tuning on downstream tasks (Dao et al., 2022). However, these methods do not allow for parameter-efficient updates.

Low-rank decomposition Low-rank approximation (Ye, 2005) has broad applications in the machine learning community and is vastly studied. One classical scenario is the robust principal component analysis (Candès et al., 2011), which decomposes a matrix into a low-rank plus a sparse component. Existing literature shows that in deep learning, the learned over-parameterized models often naturally bear approximate low-rank weight structures (Oymak et al., 2019; Yu et al., 2017).

Some (Jaderberg et al., 2014; Povey et al., 2018; Sainath et al., 2013; Zhang et al., 2014; Zhao et al., 2016) have explicitly imposed the low-rank constraint during training. Wang et al. (2020); Hu et al. (2021) utilized low-rank decomposition to shrink the model size and trim down the trainable parameters during fine-tuning. However, to our best knowledge, integrating sparsity and low-rank structures has never been studied before for efficient fine-tuning of pre-trained language models.

Parameter-efficient adaptation. Parameter-efficient fine-tuning aims to reduce the number of trainable parameters when fine-tuning the models across different downstream domains. Unlike pruning, it aims at adapting models with fewer parameters instead of building sparse models. Various approaches are proposed to achieve this goal: Rebuffi et al. (2017); Houlsby et al. (2019) inserted and only trained adapters between existing layers, whose parameters are much less compared to the pretrained models. Guo et al. (2020) leveraged ℓ_0 regularization to limit the number of non-zero elements in the update vectors. Lester et al. (2021); Li and Liang (2021); Liu et al. (2021) introduced efficient prompt tuning which optimizes only a small continuous task-specific vector. Zaken et al. (2021) fine-tunes only the bias terms inside models. Hu et al. (2021) proposed a low-rank decomposition-based method, and He et al. (2021) combined low-rank and adapter-based methods for efficient finetuning. However, fine-tuned models yielded by these methods have the same amount of weights as the pre-trained models; hence they contribute no resource efficiency.

3 Methodology

In this section, we describe our notations and definitions of sparsity generation and parameter-efficient fine-tuning in Section 3.1, and then introduce the dually sparsity-embedded efficient fine-tuning algorithms in Sections 3.2 and 3.3.

3.1 Preliminaries

Sparsification and resource-efficient fine-tuning. We adopt both unstructured and structured pruning methods to produce sparsity. They can lead to resource-efficiency including memory and computation savings.

Given $\mathcal{W} \in \mathbb{R}^{m\times n}$ a weight matrix, pruning aims at finding a binary mask $\mathcal{M} \in \{0,1\}^{m\times n}$ which is applied to $\mathcal{W}$ and generating a sparse weight $\mathcal{W} \odot \mathcal{M}$. The weights at the positions where $\mathcal{M}$ have "0" value are considered as pruned. Pruning methods can be classified into two classes by the structure of $\mathcal{M}$: For unstructured pruning where $\mathcal{M}$ does not have sparse structures such as rows and columns, the memory cost is saved due to fewer number of nonzero parameters; for structured pruning, it also helps save computational cost since the sparse weights can be smaller in size. One of the most widely used unstructured pruning methods is the weight magnitude (Han et al., 2015), *i.e.*, remove the weights with the smallest absolute values. One common structured pruning method in the NLP field is the head pruning (McCarley et al., 2019), which tries to remove unimportant attention heads from the model.

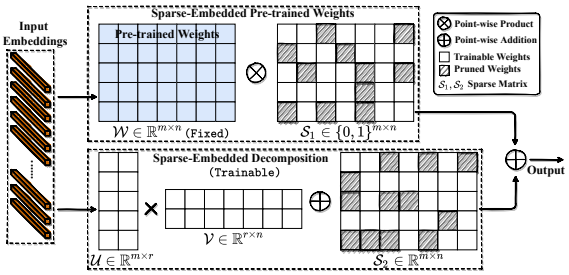

Figure 1: The overview of our proposal. The sparse masks can have unstructured or structured patterns, which leads to resources efficiency. During the fine-tuning, we only train decomposed matrices $\mathcal{U}$, $\mathcal{V}$ and non-zero elements in $\mathcal{S}_2$.

Parameter-efficient fine-tuning. To leverage the knowledge in pre-trained weights $\mathcal{W}$, downstream models learn task-specific weight update $\Delta\mathcal{W}$ via fine-tuning and generate predictions with weights $\mathcal{W} + \Delta\mathcal{W}$, where the output of models is calculated as $(\mathcal{W} + \Delta\mathcal{W})x$ with x as the input. Since $\Delta\mathcal{W}$ has the same size as $\mathcal{W}$, learning the update matrices usually requires massive resources as the size of the pre-trained model increases. Parameter-efficient fine-tuning tries to solve this problem by using as few trainable parameters as possible to represent $\Delta\mathcal{W}$, while maintaining competitive downstream fine-tuning performance. Previous literature reaches the goal via either sparsifying weight update matrices $\Delta\mathcal{W}$ (Guo et al., 2020) or leveraging low-rank decomposed matrices to compute $\Delta\mathcal{W}$ (Hu et al., 2021), while in our work we combine both of them.

3.2 Sparsity-Embedded Parameter-Efficient Fine-tuning

A recent study (Hu et al., 2021) enforces low-rank constraint to weight update tensors $\Delta\mathcal{W}$, and obtains a satisfactory trade-off between parameter-efficiency and model quality. However, as re-

Algorithm 1: Sparsity-Embedded Low-Rank Decomposition

Input: Pretrained weights $\mathcal{W}$, number of non-zero elements N, number of weights to decompose n.

Output: Indices sets $\Omega_i, i = 1, 2, \ldots, n$.

Initialize each Ω_i to be an empty set.

for *each weight matrix* $\mathcal{W}_i$ *in* $\mathcal{W}$ **do**

`/* Decomposition */`

Perform matrix decomposition $\mathcal{W}_i \approx \mathcal{AB} + \mathcal{S}'$ by solving the optimization problem 1.

`/* Extract important elements from` $\mathcal{S}'$ `into` Ω_i. `*/`

Perform thresholding on $\mathcal{S}'$: Keep N elements in $\mathcal{S}'$ with top magnitudes, and append their locations into Ω_i.

end

vealed experimentally by (Yu et al., 2017), a part of the important information in the trained weights scatters outside the low-rank subspace, creating sparse "residuals". Inspired by this observation, we investigate a new sparsity-aware low-rank subspace of $\Delta\mathcal{W}$, and introduce the first component of our proposal in Figure 1, *i.e.*, sparsity-embedded parameter-efficient fine-tuning.

Specifically, the weight updates $\Delta\mathcal{W}$ are consisted of two components as illustrated in Figure 1,: (1) a low-rank component $\Delta\mathcal{W}_l$ built by the multiplication of two matrices $\mathcal{U} \in \mathbb{R}^{m\times r}$ and $\mathcal{V} \in \mathbb{R}^{r\times n}$; and (2) a sparse residual $\Delta\mathcal{W}_s = \mathcal{P}_\Omega(\mathcal{S})$ where $\mathcal{S} \in \mathbb{R}^{m\times n}$ is a learnable matrix, $\mathcal{P}_\Omega(\mathcal{S}) = \begin{cases} s_{i,j}, & (i,j) \in \Omega \\ 0, & (i,j) \in \Omega^C \end{cases}$, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$, $w_{i,j}$ is the parameter of $\mathcal{S}$ at location (i, j), and Ω is a indices set containing the positions of non-zero elements in $\mathcal{S}$. The update matrix $\Delta\mathcal{W}$ is expressed as $\Delta\mathcal{W}_l + \Delta\mathcal{W}_s$, with $\mathcal{U}$, $\mathcal{V}$ and $\mathcal{S}$ as the learnable parameters while Ω is fixed once determined. Compared to the full fine-tuning which has $m \times n$ trainable parameters for a matrix with size $m \times n$, our method only has $(m + n) \times r + \text{card}(\Omega)$ trainable parameters. If r is smaller than $\frac{m\times n-\text{card}(\Omega)}{m+n} \approxlesssim 0.5 \min\{m, n\}$, our method is capable of reducing trainable parameters for downstream fine-tuning. In practice, the value of r is very small compared to m and n so the savings are significant.

One question for the above pipeline is how to find a high-quality indices set Ω. Inspired by the observation that the low-rank component $\Delta\mathcal{W}_l$ is highly correlated with the low-rank structure of $\mathcal{W}$ (Hu et al., 2021), we hypothesize that the indices set Ω should be highly correlated as well. More concretely, we hypothesize that the sparse residuals that are not in the low-dimensional subspace of $\mathcal{W}$ may also lay outside $\Delta\mathcal{W}_l$, which motivates the design of sparse update $\Delta\mathcal{W}_s$. We formulate the problem of discovering the sparse residuals of $\mathcal{W}$ as a Robust Principal Component Analysis (Candès et al., 2011). Formally, we aim at solving the following optimization problem:

$$\begin{aligned} \min_{U,V,S} \ & \frac{1}{2}\|W - UV - S\|_F^2 \\ \textit{s.t.} \ & \text{rank}(U) \le r, \ \text{rank}(V) \le r, \\ & \text{card}(S) \le c. \end{aligned} \tag{1}$$

where $\text{rank}(\cdot)$ and $\text{card}(\cdot)$ indicate the rank and the number of non-zero elements of a matrix, respectively. $\mathcal{S}'$ represents the sparse residuals that cannot be fit in the low-rank component $\mathcal{AB}$, and we acquire the locations of elements with non-zero magnitude into Ω. To solve Problem 1 efficiently, we adopt an SVD-free algorithm called GreBsmo (Zhou and Tao, 2013) (refer to Section A.2). Algorithm 1 summarizes the detailed procedure of constructing sparse indices sets Ω. Empirically, we set the size of Ω (*i.e.*, c) to be 16 since it yields high test performance (refer to Section 4.2) while imposing little overhead on parameters. The initial values of $\mathcal{V}$ and $\mathcal{S}$ are set as 0 so these matrices do not affect outputs at the beginning of training.

3.3 Dually Sparsity-Embedded Efficient Tuning (DSEE)

Adapting pre-trained models with $\Delta\mathcal{W}_l$ and $\Delta\mathcal{W}_s$ can bring significant parameter-efficiency, but does not directly bring any resource-efficiency such as memory or computational cost. Motivated by such, we propose a unified framework called DSEE pursuing both parameter- and resource-efficiency simultaneously. We leverage the sparsity in pre-trained models' weights to enhance the resource efficiency, as demonstrated in Figure 1. More specifically, we derive sparse masks $\mathcal{M}$ *directly* from the parameter-efficient updates $\Delta\mathcal{W}$, and apply the sparse masks by pruning the pre-trained weights $\mathcal{W}$ to seek resource-efficiency. It requires no additional parameter for sparsifying the model and no access to the underlying pretrained weights, which is favorable due to the lower sparsification cost.

As shown in Algorithm 2, DSEE handles unstructured and structured pruning at the same time: for unstructured pruning, we sort the magnitude of $\Delta\mathcal{W}$, generate a sparse mask $\mathcal{M}$ by assigning "1" to the position where $\Delta\mathcal{W}$ have largest magnitude and "0" to the rest; for structured pruning, we sum the magnitude of $\Delta\mathcal{W}$ of each head and remove those with least scores. We also shrink $\Delta\mathcal{W}$ accordingly by removing the corresponding weight columns in $\mathcal{V}$ and $\Delta\mathcal{W}_s$ to match the shape while keeping $\mathcal{U}$ intact. A comparison of different pruning criteria is shown in Section 4.2.1, which demonstrates that $\Delta\mathcal{W}$ is a superior choice due to the high downstream task performance and no access to the pretrained weights $\mathcal{W}$.

Given a parameter budget, the number of parameters per module decreases if we choose to adapt more modules, which imposes a trade-off. We study different choices of modules to adapt in Section 4.2.2, and we find the optimal modules to adapt are W_q and W_v, where W_q and W_v stand for the projection weights for query and value in attention heads. Since some modules are not adapted during fine-tuning (*i.e.*, $\Delta\mathcal{W} = 0$), we prune them separately according to the magnitude of the corresponding pre-trained weights. After applying the mask $\mathcal{M}$ to the pretrained weights $\mathcal{W}$, we conventionally tune $\Delta\mathcal{W}_l(= \mathcal{UV})$ and $\Delta\mathcal{W}_s(= \mathcal{P}_\Omega(\mathcal{S}))$ for several epochs to recover the performance (Han et al., 2015).

Algorithm 2: DSEE

Input: Pretrained weights $\mathcal{W}$, number of non-zero elements N, desired sparsity s, loss function $\mathcal{L}$.

Output: Sparse mask $\mathcal{M}$, matrices $\mathcal{U}, \mathcal{V}, \mathcal{S}$.

Derive Ω from pretrained weights $\mathcal{W}$.

Initialization: $\mathcal{U} = 0, \mathcal{V} \sim \mathcal{N}(0, 0.02)$, and $\mathcal{S} = 0$.

`/* I: train before pruning */`

Train $\mathcal{U}, \mathcal{V}, \mathcal{S}$ with respect to $\mathcal{L}$ under the constraint of $P_{\Omega^C}(\mathcal{S}) = 0$.

`/* II: pruning the model */`

if using unstructured pruning **then**

  Prune $(1 - s\%)$ weights in $\mathcal{W}$ by sorting the magnitude of $\Delta\mathcal{W}$.

else

  Prune $(1 - s\%)$ heads by sorting the aggregated magnitude of $\Delta\mathcal{W}$ of heads. Shrink $\mathcal{V}$ and $\mathcal{S}$ accordingly to match the shape.

end if

`/* III: tuning after pruning */`

Fine-tune $\mathcal{U}, \mathcal{V}, \mathcal{S}$ to recover the performance.

4 Experiment Results

Datasets and models. We use three classical pre-trained language models in our experiments: $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2019), $\text{RoBERTa}_{\text{LARGE}}$ (Liu et al., 2019) and GPT-2 (Radford et al., 2019), which have $12/24/24$ layers with hidden size of $768/1024/1024$ and $110/380/354$M trainable parameters, respectively. For BERT and RoBERTa, we evaluate on the GLUE benchmarks (Wang et al., 2018), and for GPT-2 we use E2E (Novikova et al., 2017), WebNLG (Gardent et al., 2017) and DART (Nan et al., 2021).

Training and evaluation details. For BERT and RoBERTa, we follow the default settings in Wolf et al. (2019); Devlin et al. (2019). We use the AdamW (Loshchilov and Hutter, 2017) optimizer for downstream fine-tuning, and a batch size of 32 for BERT and RoBERTa, and a batch size of 2 for GPT-2. The rest hyper-parameters for training are reported in Table 11.

Evaluation Metrics. For the GLUE benchmark, we report the accuracy, Matthew's correlation, and Pearson's correlation in the evaluation. On GPT-2, we use BLEU (Papineni et al., 2002), METEOR (Denkowski and Lavie, 2014), TER (Snover et al., 2006) and NIST (Doddington, 2002) as the evaluation metrics. To evaluate the efficiency of models, we report the number of trainable parameters to measure the parameter-efficiency, the number of total parameters (the number of non-zero parameters in the model) to measure the resource-efficiency, and FLOPs for the computational efficiency.

Baselines. On BERT and RoBERTa, we conduct comprehensive experiments with the following baseline methods: ❶ Fine-tune: directly fine-tuning the full model; ❷ EarlyBERT (Chen et al., 2021b): learn importance scores for heads and perform pruning based on them afterwards; ❸ BERT Tickets (Chen et al., 2020): IMP-based unstructured pruning; ❹ P-Tuning v2 (Liu et al., 2021); ❺ Bitfit (Zaken et al., 2021): fine-tuning bias terms only; and ❻ LoRA: low-rank decomposition, which learns $\Delta\mathcal{W}_l$ only (Hu et al., 2021). On GPT-2, we conduct comparisons with multiple baseline methods: ❶ Adapters (Houlsby et al., 2019): insert

Table 1: Performance comparison with $\text{BERT}_{\text{BASE}}$ on SST-2, RTE, CoLA, and MRPC. We report both the median and the standard deviation from five runs.

$\Delta\mathcal{W} =$	# Trainable Parameters	SST-2	RTE	CoLA	MRPC
$\Delta\mathcal{W}_l$	589.8K	92.55 (0.35)	68.95 (2.02)	60.34 (1.69)	86.27 (0.88)
$\Delta\mathcal{W}_l + \Delta\mathcal{W}_s$	590.2K	**92.78 (0.34)**	**70.04 (1.35)**	60.31 (1.04)	86.52 (0.57)
$\Delta\mathcal{W}_l$	294.9K	92.32 (0.36)	68.23 (1.43)	58.48 (1.61)	86.52 (0.72)
$\Delta\mathcal{W}_l + \Delta\mathcal{W}_s$	295.3K	**92.66 (0.06)**	**69.31 (2.08)**	**58.85 (0.92)**	**87.01 (0.79)**

adapters after linear layers; ❷ FT-Top2: fine-tune the top 2 layers only; ❸: Prefix: prefix tuning introduced by Li and Liang (2021); and ❹ LoRA.

4.1 Efficient Tuning with DSEE

Parameter-efficiency with sparse residuals. To verify that using a simple low-rank component $\Delta\mathcal{W}_l$ has limitations, we compare its performance with our sparsity-embedded efficient fine-tuning. Table 1 shows that on four benchmarks (*i.e.*, SST-2, RTE, CoLA, and MRPC), adding a sparse residual in weight updates can bring performance gain: at the level of approximately 600K trainable parameters, adding sparse residuals with only 384 nonzero elements ($12 \times 2 \times 16 = 384$) can increase the validation performance on all benchmarks except CoLA by $0.23\% \sim 1.09\%$; at the level of approximately 300K trainable parameters, adding sparse residuals can bring performance gain ranged from 0.34% to 1.08% on all four benchmarks.

We further verify that adding sparse residuals $\Delta\mathcal{W}_s$ could benefit NLG tasks with GPT-2. Table 2 shows that under different levels of parameters, adding sparse residuals $\Delta\mathcal{W}_s$ yields higher performance for most metrics on three tasks. At the level of 0.39M parameters, adding sparse residuals can improve all metrics on WebNLG and DART, and sightly boost the NIST score on E2E. At the level of 0.20M parameters, $\Delta\mathcal{W}_s$ helps increase all metrics across three tasks. We also show the standard deviation in Table 10.

Resource- and parameter-efficiency with unstructured sparse masks. We verify that DSEE is capable of enhancing both parameter- and resource-efficiency, while preserving performance on downstream tasks, on various architectures. Table 3 summarizes the experiment results on $\text{BERT}_{\text{BASE}}$, and we observe that introducing unstructured sparsity patterns inside pretrained weights not only brings resource-efficiency (manifested by the fewer number of total parameters) but also potentially improves the performance on downstream tasks. Specifically, at 80% and 70% of total parameters, DSEE can remain comparable performance on downstream tasks, and even present a performance boost on QQP, RTE, and SST-2 compared to LoRA. At the level of 50% parameters, performance on smaller datasets such as CoLA and RTE drops by a wider margin; but on larger datasets such as QQP, DSEE can maintain comparable performance ($< 1.5\%$ gap) after sparsification.

On GPT-2, we observe a similar trend as shown in Table 4. DSEE can achieve superior performance with unstructured sparse patterns with 80% total parameters compared to finetuning the entire model, and remain highly competitive with other baselines with fewer parameters in the model. Using only 50% of parameters in pre-trained weights, DSEE can achieve comparable performance with the full fine-tuning on E2E and DART.

Finally, we validate if DSEE can work on the larger model $\text{RoBERTa}_{\text{LARGE}}$. We conduct experiments on four datasets (CoLA, SST-2, QNLI, and RTE), and present the results in Table 5. Compared to full-finetuning, LoRA, and Adapter, our method reaches comparable performance on these four downstream tasks and saves resources at the same time. The performance gap is maximal 1% but 30% parameters in the models are removed.

Resource- and parameter-efficiency with structured sparse masks. DSEE can directly perform structured pruning on weights without additional parameters such as importance scores of heads. In Table 6 we show the performance of structured pruned $\text{BERT}_{\text{BASE}}$ on several tasks in the GLUE benchmark, where we study the testing accuracy after removing 3, 6 and 9 attention heads on SST-2, MNLI, QNLI and QQP, as well as the inference FLOPs ratios of the model. Firstly, removing 3 heads from the model reaches comparable performance against full fine-tuning (improved on SST-2, MNLI, and QNLI) and LoRA (improved on SST-2 and QQP), while taking advantage of reduced inference FLOPs. Secondly, removing 6 heads from the model will lead to lower performance since half of the parameters in the projection matrices are eliminated. However, the performance of DSEE is still higher than EarlyBERT. Lastly, DSEE with 9 heads removed from the model leads to comparable performance with EarlyBERT, but the number of trainable parameters is substantially smaller (0.6M versus 66M).

Table 2: Performance comparison of different decomposition on GPT-2 with different weight update terms. We report the median value of BLEU, MET, NIST and TER from five runs.

Forms	# Trainable Parameters	E2E			WebNLG			DART		
		BLEU	MET	NIST	BLEU	MET	TER	BLEU	MET	TER
$\Delta\mathcal{W} = \Delta\mathcal{W}_l$	0.39M	70.38	46.89	8.844	55.29	0.414	0.394	48.23	0.392	0.469
$\Delta\mathcal{W} = \Delta\mathcal{W}_l + \Delta\mathcal{W}_s$	0.39M	70.29	46.65	8.858	55.50	0.416	0.392	48.17	0.397	0.467
$\Delta\mathcal{W} = \Delta\mathcal{W}_l$	0.20M	69.17	45.90	8.741	55.23	0.413	0.396	46.49	0.387	0.477
$\Delta\mathcal{W} = \Delta\mathcal{W}_l + \Delta\mathcal{W}_s$	0.20M	69.70	46.85	8.824	55.56	0.413	0.392	47.47	0.393	0.475

Table 3: Performance comparison of different methods on GLUE benchmarks with $\text{BERT}_{\text{BASE}}$. We use the unstructured pruning and report the median value from five runs. †: results taken from Chen et al. (2020).

Methods	# Trainable Parameters	# Total Parameters	Dataset							
			CoLA	STS-B	MNLI	QQP	QNLI	MRPC	RTE	SST-2
Fine-tune†	110M	100%	54.5	88.4	82.4	90.2	89.1	85.2	66.2	92.1
BERT Tickets†	$33 \sim 55$M	$30 \sim 50\%$	53.8	88.2	82.6	90.0	88.9	84.9	66.0	91.9
P-Tuning v2	0.3M	100%	59.37	89.36	82.15	88.50	90.59	84.80	67.51	92.20
Bitfit	0.1M	100%	58.61	88.74	78.80	85.93	89.22	87.55	72.20	92.07
LoRA	0.6M	100%	59.99	89.09	83.32	89.48	90.72	86.27	68.95	92.32
DSEE	0.6M	80%	59.94	89.22	83.29	90.00	90.46	86.27	70.76	92.66
DSEE	0.6M	70%	58.69	89.08	83.09	89.97	90.68	86.27	71.48	91.97
DSEE	0.6M	50%	48.49	87.72	81.84	89.55	90.12	81.13	63.90	91.17

4.2 Ablation and Visualization

We study several choices of parameters and provide visualization in this section.

4.2.1 Different criteria for sparse masks

We find the magnitude of weight updates (*i.e.*, $|\Delta\mathcal{W}|$) is an effective solution for preserving performance with both unstructured and structured pruning. We conduct experiments on the adapted weights (*i.e.*, W_q and W_v), and compare against two baselines: ❶ Random: perform random pruning on the adapted modules; ❷ $|\mathcal{W} + \Delta\mathcal{W}|$: perform pruning based on the magnitude of final adapted weights. Table 7 shows the results on RTE and SST-2 with $\text{BERT}_{\text{BASE}}$. We can see from the table that: ❶ performing unstructured pruning without accessing the pretrained weights can achieve comparable performance on RTE and SST-2, only slightly weaker than pruning with final adapted weights; ❷ performing structured pruning according to $\Delta\mathcal{W}$ yields the highest performance on both datasets after training. These observations verify the effectiveness of our proposal.

4.2.2 Different choices of modules to adapt

We study the choices of modules to adapt for DSEE on RTE. We choose possible modules to adapt within W_q, W_k, W_v, and W_o, representing the projection matrix for query, key, value, and output, respectively. We hold the number of trainable parameters at the same level and set the sparsity level at 30%. Table 9 summarizes the performance with different adapted weights, which demonstrates that adapting W_q and W_v yields the highest performance. Each module will be given fewer parameters when adapting more modules and the model may not be sufficiently fine-tuned when adapting fewer modules and leading to inferior performance.

Different methods for identifying Ω. We compare our proposal against various methods to identify Ω from pretrained weights $\mathcal{W}$: ❶ *Magnitude*, which selects the position of elements with highest magnitude into Ω; ❷ *Random*, which randomly samples positions into Ω. The results are shown in Figure 2. We can observe that our proposal can identify high-quality Ω for finetuning on downstream tasks, shown by the consistently higher performance with different sizes of the indices set Ω.

Different sizes of Ω. We search over $8 \sim 256$ to find the optimal size of Ω. Ω with a smaller size brings fewer performance gains, and Ω with a larger size may harm the efficiency. Figure 2 shows the relationship between the size of Ω and the performance on SST-2. We find the optimal choice for this task is 16 where the model achieves the highest performance. Consequently, we by default set the size of Ω to 16 for simplicity.

Table 4: Performance comparison of different methods on GPT-2 on E2E, WebNLG and DART. ‡: Results taken from Hu et al. (2021).

Methods	# Trainable Parameters	# Total Parameters	E2E			WebNLG			DART		
			BLEU	MET	NIST	BLEU	MET	TER	BLEU	MET	TER
Fine-tune‡	354.92M	100%	68.2	0.462	8.62	47.6	0.39	0.50	46.0	0.39	0.46
Adapters‡	11.48M	100%	68.9	0.461	8.71	55.2	0.41	0.39	45.4	0.38	0.46
FT-Top2‡	25.19M	100%	68.1	0.460	8.59	33.5	0.26	0.75	38.1	0.34	0.56
Prefix‡	0.35M	100%	69.7	0.461	8.81	54.4	0.41	0.41	45.7	0.38	0.46
LoRA‡	0.39M	100%	70.4	0.468	8.85	55.3	0.41	0.39	47.5	0.39	0.45
DSEE	0.39M	80%	69.4	0.465	8.78	54.9	0.44	0.39	47.5	0.39	0.46
DSEE	0.39M	50%	69.5	0.466	8.74	42.0	0.33	0.53	43.4	0.37	0.51

Table 5: Performance comparison of different methods on RoBERTa$_{\text{LARGE}}$ on CoLA, SST-2, MRPC and RTE. ‡: Results taken from Hu et al. (2021).

Methods	# Trainable Parameters	# Total in Parameters	Dataset			
			CoLA	SST-2	QNLI	RTE
Fine-tune‡	355.0M	100%	68.0	95.1	94.7	86.6
Adapter‡	0.8M	100%	66.3	96.3	94.7	72.9
LoRA‡	0.8M	100%	68.2	96.2	94.8	85.2
DSEE	0.8M	70%	67.2	96.1	94.4	84.9

Table 6: Performance comparison of different methods on GLUE benchmarks with BERT$_{\text{BASE}}$. We perform the structured pruning and report the median value from five runs. †: results taken from Chen et al. (2020).

Methods	FLOPs	# Trainable	SST-2	MNLI	QNLI	QQP
Fine-tune†	1.0×	110M	92.1	82.4	89.1	90.2
LoRA	1.01×	0.6M	92.32	83.32	90.72	89.48
EarlyBERT	0.63×	∼ 66M	90.71	81.81	89.18	90.06
DSEE (3 heads)	0.92×	0.6M	92.55	83.25	90.65	89.84
DSEE (6 heads)	0.84×	0.6M	92.32	82.32	90.01	89.11
DSEE (9 heads)	0.75×	0.6M	91.63	80.02	88.39	88.56

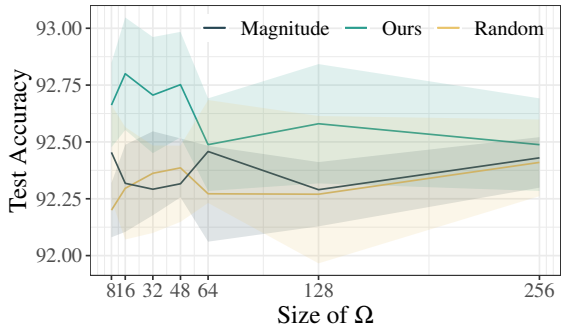

Figure 2: Testing performance on SST-2 with different sizes of Ω. We report the average accuracy and the 90% confidence interval of five runs.

5 Conclusion

This paper draws on the prior of sparsity and establishes the DSEE framework. It is the first attempt toward jointly optimizing both parameter-efficiency of the fine-tuning process, and the resource-efficiency of the fine-tuned model. On state-of-the-art large-scale language models (e.g., BERT, GPT, and RoBERTa) and across several datasets, DSEE consistently demonstrates highly impressive parameter and inference efficiency, in addition to preserving a competitive downstream transfer performance on various tasks. Our future work targets extending DSEE to the finetuning of large-scale computer vision and/or multi-modal pre-trained models.

Table 7: Performance of using different pruning criteria to generate unstructured masks. We only perform pruning on W_q and W_v. The first part applies unstructured pruning and the latter applies structured pruning.

Criterion	RTE	SST-2
$\|\Delta\mathcal{W}\|$	69.68 (1.37)	91.97 (0.26)
$\|\mathcal{W}+\Delta\mathcal{W}\|$	**70.76 (2.09)**	**92.78 (0.39)**
Random	64.62 (2.28)	91.63 (0.25)
$\|\Delta\mathcal{W}\|$	**70.40 (1.05)**	**92.55 (0.43)**
$\|\mathcal{W}+\Delta\mathcal{W}\|$	68.59 (1.60)	92.20 (0.60)
Random	68.23 (1.29)	91.97 (0.14)

Limitation The unstructured sparse patterns we introduce are not as hardware-friendly as the structured patterns, suggesting the speedup of using unstructured patterns maybe limited due to the implementation. The number of parameters of models we are studying are only at the level of $100 \sim 300$M, and the datasets are focus on GLUE, E2E, WebNLG, and DART. We will generalize to wider choices of datasets in future works.

6 Ethical and Broader Impacts

DSEE aims at reducing the number of trainable parameters when fine-tuning the models, which can help save the cost of saving new weights. This can be helpful to companies who are fine-tuning large-

scale language models on various downstream tasks, suggesting our work has potentially positive broader impact. On the other hand, our work does not have obvious ethical impacts, as we focusing on model tuning.

References

Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. 2011. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):1–37.

Beidi Chen, Tri Dao, Kaizhao Liang, Jiaming Yang, Zhao Song, Atri Rudra, and Christopher Re. 2021a. Pixelated butterfly: Simple and efficient sparse training for neural network models. *arXiv preprint arXiv:2112.00029*.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. 2020. The lottery ticket hypothesis for pre-trained bert networks. *arXiv preprint arXiv:2007.12223*.

Xiaohan Chen, Yu Cheng, Shuohang Wang, Zhe Gan, Zhangyang Wang, and Jingjing Liu. 2021b. Early-bert: Efficient bert training via early-bird lottery tickets. In *Proceedings of the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*.

Tri Dao, Beidi Chen, Nimit Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. 2022. Monarch: Expressive structured matrices for efficient and accurate training. *arXiv preprint arXiv:2204.00595*.

Michael Denkowski and Alon Lavie. 2014. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the ninth workshop on statistical machine translation*, pages 376–380.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. 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 4171–4186.

George Doddington. 2002. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*, pages 138–145.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. The webnlg challenge: Generating text from rdf data. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 124–133.

Demi Guo, Alexander M Rush, and Yoon Kim. 2020. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*.

Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1389–1397.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference. BMVA Press*.

Yann LeCun, John S Denker, and Sara A Solla. 1990. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

JS McCarley, Rishav Chakravarti, and Avirup Sil. 2019. Structured pruning of a bert-based question answering model. *arXiv preprint arXiv:1910.06360*.

Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, Yangxiaokang Liu, Nadia Irwanto, Jessica Pan, Faiaz Rahman, Ahmad Zaidi, Mutethia Mutuma, Yasin Tarabar, Ankit Gupta, Tao Yu, Yi Chern Tan, Xi Victoria Lin, Caiming Xiong, Richard Socher, and Nazneen Fatema Rajani. 2021. Dart: Open-domain structured data record to text generation.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The e2e dataset: New challenges for end-to-end generation. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 201–206.

Samet Oymak, Zalan Fabian, Mingchen Li, and Mahdi Soltanolkotabi. 2019. Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian. *arXiv preprint arXiv:1906.05392*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Daniel Povey, Gaofeng Cheng, Yiming Wang, Ke Li, Hainan Xu, Mahsa Yarmohammadi, and Sanjeev Khudanpur. 2018. Semi-orthogonal low-rank matrix factorization for deep neural networks. In *Interspeech*, pages 3743–3747.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 506–516.

Ao Ren, Tianyun Zhang, Shaokai Ye, Jiayu Li, Wenyao Xu, Xuehai Qian, Xue Lin, and Yanzhi Wang. 2018. Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction method of multipliers.

Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. 2013. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6655–6659. IEEE.

Victor Sanh, Thomas Wolf, and Alexander M Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. In *NeurIPS*.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pages 223–231.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2020. Structured pruning of large language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6151–6162.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Jieping Ye. 2005. Generalized low rank approximations of matrices. *Machine Learning*, 61(1-3):167–191.

Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. 2017. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

Yu Zhang, Ekapol Chuangsuwanich, and James Glass. 2014. Extracting deep neural network bottleneck features using low-rank matrix factorization. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 185–189. IEEE.

Yong Zhao, Jinyu Li, and Yifan Gong. 2016. Low-rank plus diagonal adaptation for deep neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5005–5009. IEEE.

Tianyi Zhou and Dacheng Tao. 2013. Greedy bilateral sketch, completion & smoothing. In *Artificial Intelligence and Statistics*, pages 650–658. PMLR.

A More Implementation Details

A.1 Hyper-parameters

We report the learning rates, the batch sizes, and the max sequence length for DSEE in Table 11. The device we used for experiments are various, including NVIDIA GeForce GTX 1080 Ti, GeForce RTX 2080 Ti, Titan RTX, and A6000. We follow (Hu et al., 2021) to set the evaluation protocols on E2E, WebNLG, and DART.

A.2 Decomposition Method

GreBsmo (Zhou and Tao, 2013) is an algorithm for solving the Robust PCA-like methods. The optimization of $\mathcal{U}$, $\mathcal{V}$ and $\mathcal{S}$ follows the following iterative rules:

$$\begin{cases} U_k = Q, \mathrm{QR}\left(\left(X - S_{k-1}\right) V_{k-1}^T\right) = QR \\ V_k = Q^T \left(X - S_{k-1}\right) \\ S_k = \mathcal{S}_\lambda \left(X - U_k V_k\right) \end{cases}, \tag{2}$$

where X is the original dense matrix, $\mathrm{QR}(\cdot)$ means the QR decomposition, $S_\lambda(\cdot)$ indicates the soft-threshold function (*i.e.*, $S_\lambda(x) = x\mathbf{1}_{|x|\geq\lambda}$) , and the subscripts k indicates the optimization step.

A.3 Statistics and Usage of Datasets

We report the statistics of datasets in Table 8. For GLUE tasks we report the sizes of the train, the dev and the test set, and for non-GLUE tasks we report the sizes of the train, validation (dev), and test set. We follow the conventional use of these datasets (Hu et al., 2021) and do not modify the conventional splits.

B More Experiments Results

B.1 Ablation Studies

Table 9 summarizes the performance with different adapted weights, which demonstrates that adapting W_q and W_v leads to the highest performance.

Performance of different Ω. We conduct additional ablation study experiments (three runs for each experiment) on the sizes of Ω on three datasets in GLUE (*i.e.*, STSB, QNLI and MRPC). The results shown in Table 12 below verify that our method are generalizable to other datasets. On STSB and QNLI, using a size of 16 can achieve the best performance, while on MRPC it can achieve a comparable test accuracy.

Compare with recent methods. We have conducted a set of experiment to compare our methods with MAM Adapter (He et al., 2021). We train a RoBERTa-large with their method on SST-2, QNLI, RTE, by following the same hyperparameters used in the original work. The results are shown in Table 13. We observe that our method, even with sparse models, achieves same-level performance with LoRA and MAM Adapter.

Other Pruning Methods. We apply the iterative magnitude pruning method on RTE. Specifically, we train the model for 10 epochs, prune 10% of the remaining weights, and fine-tune for 10 epochs before the next pruning. Table 14 shows that directly applying iterative magnitude pruning does not bring performance improvements over the one-shot pruning baseline.

Table 8: The statistics of datasets we used for experiments.

Name	Train	Dev	Test
GLUE			
CoLA	8,551	1,043	-
SST-2	67,349	872	-
MNLI	392,702	9,815	-
QNLI	104,743	5,463	-
QQP	363,846	40,430	-
STS-B	5,749	1,500	-
RTE	2,490	277	-
MRPC	3,668	408	-
non-GLUE			
E2E	42,061	4,672	4,693
WebNLG	18,025	2,258	4,928
DART	30,526	2,768	6,959

Table 9: Testing performance of $\mathrm{BERT}_{\mathrm{BASE}}$ on RTE with different adapted modules. We report the median values and the standard deviation from three runs.

Weights	Test Acc.	Weights	Test Acc.
W_q	68.59 (0.21)	W_k	67.87 (0.21)
W_v	68.23 (1.82)	W_o	68.23 (1.05)
W_q,W_k	68.95 (1.11)	W_q,W_v	**71.48 (2.16)**
W_k,W_v	70.04 (0.75)	W_q,W_k,W_v	69.31 (2.56)

Table 10: Performance comparison of different decomposition on GPT-2 with different weight update terms. We report the standard deviation of BLEU, MET, NIST and TER from five runs.

Forms	# Trainable Parameters	E2E			WebNLG			DART		
		BLEU	MET	NIST	BLEU	MET	TER	BLEU	MET	TER
$\Delta\mathcal{W} = \Delta\mathcal{W}_l$	0.39M	0.43	0.13	0.037	0.37	0.005	0.003	0.23	0.001	0.001
$\Delta\mathcal{W} = \Delta\mathcal{W}_l + \Delta\mathcal{W}_s$	0.39M	0.07	0.26	0.047	0.48	0.005	0.004	0.40	0.003	0.002
$\Delta\mathcal{W} = \Delta\mathcal{W}_l$	0.20M	0.23	0.03	0.043	0.26	0.005	0.007	0.06	0.002	0.001
$\Delta\mathcal{W} = \Delta\mathcal{W}_l + \Delta\mathcal{W}_s$	0.20M	0.61	0.19	0.029	0.52	0.006	0.004	0.15	0.001	0.001

Table 11: Hyper-parameters we used on different datasets and architectures.

Architecture	Method	Parameters	Dataset							
			MNLI	QNLI	QQP	SST-2	CoLA	MRPC	RTE	STS-B
$\text{BERT}_{\text{BASE}}$	DSEE (before pruning)	Learning Rate	2e-4	2e-4	2e-4	2e-4	1e-3	8e-4	6e-4	8e-4
$\text{BERT}_{\text{BASE}}$	DSEE (after pruning)	Learning Rate	2e-4	2e-4	2e-4	2e-4	1e-3	8e-4	6e-4	8e-4
$\text{BERT}_{\text{BASE}}$	DSEE	Batch Size	32							
$\text{BERT}_{\text{BASE}}$	DSEE	Max Sequence Length	128							
$\text{RoBERTa}_{\text{LARGE}}$	DSEE (before pruning)	Learning Rate	-	2e-4	-	4e-4	3e-4	-	4e-4	-
$\text{RoBERTa}_{\text{LARGE}}$	DSEE (after pruning)	Learning Rate	-	2e-4	-	4e-4	3e-4	-	4e-4	-
$\text{RoBERTa}_{\text{LARGE}}$	DSEE	Batch Size	-	32	-	32	16	-	32	-
$\text{RoBERTa}_{\text{LARGE}}$	DSEE	Max Sequence Length	-	512	-	512	128	-	512	-

Table 12: Performance on the three datasets using different sizes of Ω.

Dataset	$\Omega = 8$	$\Omega = 16$	$\Omega = 32$	$\Omega = 48$
STSB	89.26	89.28	89.10	89.16
MRPC	85.38	86.27	86.36	86.27
QNLI	91.01	91.06	91.00	90.87

Table 13: Compare with more methods.

Method	Total Parameters	SST-2	QNLI	RTE
LoRA	100%	96.2	94.8	85.2
MAM Adapter	100%	96.1	94.7	80.4
Ours	70%	96.1	94.4	84.9

Table 14: Applying iterative magnitude pruning (IMP) to prune models.

Remaining Weights	Accuracy
90%	70.02%
81%	70.76%
72.9%	63.90%
65.6%	61.01%

ACL 2023 Responsible NLP Checklist

A For every submission:

☐ A1. Did you describe the limitations of your work?
Left blank.

☐ A2. Did you discuss any potential risks of your work?
Left blank.

☐ A3. Do the abstract and introduction summarize the paper's main claims?
Left blank.

☐ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☐ Did you use or create scientific artifacts?

Left blank.

☐ B1. Did you cite the creators of artifacts you used?
Left blank.

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Left blank.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Left blank.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Left blank.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Left blank.

☐ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Left blank.

C ☐ Did you run computational experiments?

Left blank.

☐ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Left blank.

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☐ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Left blank.

☐ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Left blank.

☐ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Left blank.

D ☐ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
Left blank.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
Left blank.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
Left blank.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Left blank.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
Left blank.