Train Less, Infer Faster: Efficient Model Finetuning and Compression via Structured Sparsity

Jonathan Svirsky
Bar Ilan University
jonathan.svirsky@biu.ac.il

Yehonathan Refael
Tel Aviv University
refaelkalim@mail.tau.ac.il

Ofir Lindenbaum
Bar Ilan University
ofir.lindenbaum@biu.ac.il

Abstract

Fully finetuning foundation language models (LMs) with billions of parameters is often impractical due to high computational costs, memory requirements, and the risk of overfitting. Although methods such as low-rank adapters help address these challenges by adding small, trainable modules to the frozen LM, they also increase memory usage and do not reduce inference latency. We uncover an intriguing phenomenon: sparsifying specific rows and columns of the model enables efficient task adaptation without requiring weight tuning. We propose a scheme for effective finetuning via sparsification by training stochastic gates, which requires minimal trainable parameters, reduces inference time, and removes 20–40% of model parameters without significant performance loss. Empirical results show it outperforms recent finetuning baselines in efficiency and performance. Additionally, we provide theoretical guarantees for the convergence of this stochastic gating process and show that our method admits a simpler, better-conditioned optimization landscape than LoRA. Our results highlight sparsity as a compelling mechanism for task-specific adaptation in LMs.

1 INTRODUCTION

Foundation language models (LMs) have transformed natural language processing by enabling a wide variety of powerful and versatile applications. Pre-trained on extensive amounts of text data, these models demonstrate strong performance in tasks such as text generation, translation, and sentiment analysis. Nonetheless, fine-tuning is often essential to customize these models for specific applications. Finetuning enables the model to adapt to the nuances of a particular task by updating its parameters using a smaller, task-specific dataset. However, fine-tuning poses challenges, particularly when task-specific data are limited, which can constrain the model's ability to adapt to the target task and may lead to overfitting or suboptimal performance. Despite these challenges, finetuning remains crucial in leveraging the full potential of large language models for specialized applications.

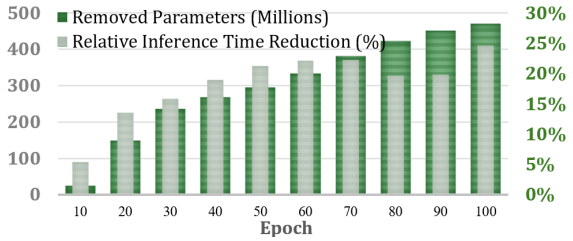

Figure 1: CPU inference time reduction (%) and number of removed parameters on the MRPC validation set while finetuning our method on the Llama3.2-1B backbone. See Section 6.6 for details.

Recently, several methods have been proposed to efficiently finetune foundation language models, addressing challenges arising from limited data and the high computational cost of updating the full set of model parameters. One such approach is LoRA (Low-Rank Adaptation) (Hu et al., 2021), which introduces a more efficient method for finetuning models by freezing the original parameters and replacing them with trainable, low-rank matrices in each transformer layer. This technique significantly reduces the number of parameters that must be updated during finetuning, making the process faster and less resource-intensive. Most recent efforts to make finetuning more efficient (Zhang et al., 2023a; Chavan et al., 2023; Xu et al., 2023; Li et al.,

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300.

2022; Lin et al., 2024a; Bałazy et al., 2024; Hu et al., 2021; Refael et al., 2025b,a) focus on adding new parameters while keeping the base model frozen, thereby retaining the original pre-trained knowledge (Rozner et al., 2024).

When training models using low-rank adapters (Kopiczko et al., 2023; Zhang et al., 2023a; Lin et al., 2024a; Bałazy et al., 2024; Hu et al., 2021), the number of optimized parameters is reduced, leading to faster convergence during finetuning. However, the inference runtime and memory requirements remain the same, as they depend on the dimensions of the base model parameters, which are frozen during adaptation. Several methods have been proposed for LM compression, such as quantization and pruning, to reduce the inference time for adapted models. Quantization reduces the memory usage of language models by converting their parameters into lower-bit data types (Bondarenko et al., 2024; Lin et al., 2024b). Although quantization reduces the memory consumption of language models, its speedups depend on specialized framework support, which limits flexibility and adaptability.

Pruning (Ma et al., 2023) aims to improve inference efficiency in language models by removing unimportant parameters. Structured pruning (Xia et al., 2022; Refael et al., 2024a) eliminates coherent blocks of parameters or entire model dimensions, leading to broad improvements in inference efficiency by reducing computational overhead. However, these methods often require substantial additional training time and increase the number of trainable parameters to enable effective self-distillation of knowledge from the layers of a large language teacher model to the smaller student model (Hinton, 2015; Xia et al., 2022).

In this work, we propose a simple yet effective approach to adapt language models via structured sparsification, achieving inference-time efficiency by pruning the overparameterized model and updating only the lightweight bias parameters. Instead of learning new parameters or updating main model weights, we learn a set of compact binary masking vectors (gates) that selectively activate a sparse subset of the model's existing weights. This approach leverages the inherent redundancy in overparameterized models, demonstrating that effective task adaptation can be achieved by identifying and utilizing the relevant subnetwork within the frozen base model. The outcome is a highly efficient adaptation mechanism that restructures the original model via sparsification, enabling impressive task-specific performance and inference-time speedups, as shown in Figure 1. Moreover, the proposed method is broadly applicable to both downstream tasks and pretraining scenarios.

Our gating mechanism is trained end-to-end, embedding task-specific information directly during the optimization process. This eliminates the need for post-training pruning or prolonged finetuning, yielding models that are both compact and effective, particularly well-suited for deployment in resource-constrained environments.

We evaluate our method[1] on Transformer-based models and demonstrate that up to 40% of the base model's parameters can be deactivated with minimal performance loss compared to full finetuning and several baselines. In the following sections, we describe the method in detail, present empirical results, and provide a theoretical analysis of convergence, including a comparison of the optimization landscapes of our gating approach and LoRA, showing that our method has a simpler, better-conditioned geometry.

2 RELATED WORK

2.1 Low-Rank Adaptation

Low-rank adaptation techniques aim to efficiently finetune foundation language models (LMs) under resource constraints by updating a small subset of parameters. This is typically achieved by introducing trainable components such as adapter layers (Pfeiffer et al., 2020; Houlsby et al., 2019), prompt or prefix embeddings (Lester et al., 2021; Li and Liang, 2021), or by imposing low-rank structures on the gradients during training (Refael et al., 2024b). A widely adopted method, LoRA (Hu et al., 2021), only trains low-rank matrices to reduce memory requirements. However, LoRA still introduces additional parameters and maintains the original model dimensionality, offering no compression benefits during inference. Moreover, it requires choosing a fixed rank hyperparameter, a choice based on heuristics that often results in suboptimal performance. Other approaches, such as SparseAdapter (He et al., 2022), dynamically adjust adapter sparsity during training, while AdaLoRA (Zhang et al., 2023b) gradually prunes singular values to reduce the number of updated parameters. Despite their efficiency during training, these methods do not yield inference-time benefits, as the underlying base model remains uncompressed and fully active.

2.2 Finetuning with Pruning

Pruning refers to the process of reducing a model's size by removing weights or structures deemed unnecessary for its performance. There are two main frameworks for pruning models during finetuning: *structured* and

[1] `https://github.com/jsvir/FineGates`

unstructured. Unstructured pruning (Sanh et al., 2020; Fang et al., 2024) removes the least important parameters in a model without following any specific order. In contrast, structured pruning (Xia et al., 2022; Zhao et al., 2024a) eliminates entire blocks, rows, or columns from the weight matrices. Additionally, a post-training pruning method proposed by (Frantar and Alistarh, 2023) aims to prune finetuned models with minimal additional cost, but it requires initialization from fully finetuned models. In contrast, our approach jointly performs task adaptation and structured pruning during training, allowing the model to specialize while removing up to 40% of the base parameters—without the need for full finetuning or post-hoc pruning. This results in a model that is both task-specific and resource-efficient, reducing inference cost with no additional training overhead.

2.3 Finetuning with Adaptive Pruning

A recent work that closely aligns with our goal was proposed by Zhao et al. (2024a), where the base model parameters are pruned concurrently with adapter training. Their approach introduces additional low-rank matrices, similar in spirit to LoRA (He et al., 2022), which must also be trained. However, this design has substantial drawbacks: the reported training time is nearly five times that of standard full-model finetuning, and compression is achieved through a costly sorting-and-binary-search procedure over weight blocks. Moreover, because the rank of the low-rank adaptation weights is adaptive, the optimization memory footprint remains heavy, reaching about 70% of that required for full-model finetuning. By contrast, our method introduces only lightweight gate vectors, which require orders of magnitude fewer parameters to optimize, incur negligible training-time overhead, and achieve compression without resorting to expensive search procedures. This makes our approach both more memory-efficient and more computationally practical than prior methods. In addition, the gates are trained end-to-end and integrate seamlessly into the training pipeline, making the approach simple to implement and deploy.

2.4 Conditional Normalization Methods

Feature-wise Linear Modulation (FiLM) (Perez et al., 2018) and related Conditional Normalization (CN) approaches apply learned affine transformations to intermediate activations in order to enable conditional computation. These methods are typically trained end-to-end and operate by scaling and shifting feature channels during the forward pass. While FiLM provides expressive feature-wise modulation, it does not modify the underlying weight matrices of a pretrained model. Consequently, even when certain activation channels are suppressed, the corresponding parameters remain present, and no structural sparsification or inference-time compression is achieved. In contrast, our method learns persistent row- and column-level sparsity directly in the weight matrices, enabling actual dimensionality reduction and deployment-time compression into a smaller dense model. Thus, whereas FiLM focuses on conditional feature modulation, the proposed method targets structured parameter elimination for efficient adaptation and acceleration.

3 PROBLEM FORMULATION

Assume we are given a pre-trained large language model $P_{\boldsymbol{\Theta}}(\mathbf{y}|\mathbf{x})$ parametrized by $\boldsymbol{\Theta}$ based on the Transformer architecture (Vaswani, 2017). The goal of finetuning is to adapt this pre-trained model for downstream natural language understanding tasks, such as question answering or sentiment analysis. The downstream task is represented by a training dataset of context-target pairs: $\mathcal{Z} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in[N]}$, where $\mathbf{x}_i$ is a sequence and $\mathbf{y}_i$ is a target label. For example, in the question-answering task (QQP) in the GLUE benchmark (Wang et al., 2019), $\mathbf{x}_i$ is a question, and $\mathbf{y}_i$ is its corresponding answer. To finetune the whole model parameters (full finetuning), the model is initialized to pre-trained weights $\boldsymbol{\Theta}_0$ and updated to $\boldsymbol{\Theta}_0 + \Delta\boldsymbol{\Theta}$ by repeatedly following the gradient updates to maximize the conditional language modeling objective:

$$\max_{\boldsymbol{\Theta}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{Z}} \sum_{t=1}^{|\mathbf{y}|} \log(P_{\boldsymbol{\Theta}}(y_t|\mathbf{x}, \mathbf{y}_{<t})). \tag{1}$$

In low-rank adaptation methods the task-specific parameter update $\Delta\boldsymbol{\Theta} = \Delta\boldsymbol{\Theta}(\boldsymbol{\Gamma})$ is further encoded by a much smaller-sized set of parameters $\boldsymbol{\Gamma}$ with $|\boldsymbol{\Gamma}| \ll |\boldsymbol{\Theta}_0|$. The task of finding $\Delta\boldsymbol{\Theta}$ thus becomes optimizing over $\boldsymbol{\Gamma}$ and not $\boldsymbol{\Theta}$,

$$\max_{\boldsymbol{\Gamma}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{Z}} \sum_{t=1}^{|\mathbf{y}|} \log(P_{\boldsymbol{\Theta}_0+\Delta\boldsymbol{\Theta}(\boldsymbol{\Gamma})}(y_t|\mathbf{x}, \mathbf{y}_{<t})). \tag{2}$$

While beneficial for preserving the base model across tasks, this approach retains non-useful information in the base model and still necessitates a forward pass over the large number of parameters during inference.

Finetuning by sparsification. In this work, we propose incorporating *gates* vectors $\boldsymbol{\omega}_r, \boldsymbol{\omega}_c$ such that $\boldsymbol{\omega}_r$ multiplies the rows of the weight matrix and $\boldsymbol{\omega}_c$ multiplies its columns in an element-wise way, thereby sparsifying rows and columns. Importantly, the entries of these gate vectors are binary (0/1), directly

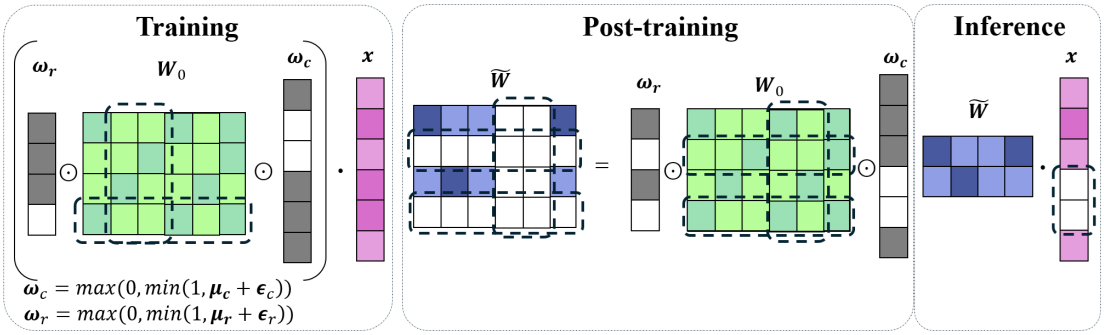

Figure 2: Overview of FineGates: Our method introduces structured sparsity in LM finetuning by training lightweight row and column gating vectors ($\boldsymbol{\omega}_c, \boldsymbol{\omega}_r$). These gates selectively retain the most informative weight dimensions, enabling efficient adaptation without modifying the base model's parameters. Unlike LoRA and other PEFT methods, which introduce additional trainable matrices, FineGates directly optimizes sparsification and updates biases, thereby reducing memory overhead and inference time while maintaining task performance.

enforcing structured sparsity. The parameters that construct these vectors for all adapted layers in the base model are denoted by $\boldsymbol{\Omega}$. This approach modifies the base model parameters by replacing $\boldsymbol{\Delta\Theta}$ with our gate vectors. Our method introduces the concept of *structured sparsity* in the base model (Wen et al., 2016), by excluding entire columns or rows from the weight matrices. Therefore, assuming that the number of compressed weight matrices in the base model is L, the goal of the finetuning task becomes:

$$\mathcal{L} = \min_{\boldsymbol{\omega}} \Bigg[- \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{Z}} \sum_{t=1}^{|\mathbf{y}|} \log\left(P_{\boldsymbol{\omega}_r \odot \boldsymbol{\Theta}_0 \odot \boldsymbol{\omega}_c}(y_t|\mathbf{x}, \mathbf{y}_{<t})\right) + \\ + \lambda \frac{1}{L} \sum_{i=1}^{L} \left[\max(||\boldsymbol{\omega}_r^i||_0, s_r^i) + \max(||\boldsymbol{\omega}_c^i||_0, s_c^i)\right] \Bigg], \quad (3)$$

where the parameter λ represents the magnitude of structured sparsity regularization. The term $||\cdot||_0$ refers to the ℓ_0 norm, while s_r^i and s_c^i denote the target sparsity ratios we aim to achieve. These ratios are defined as the number of zero parameters divided by the total number of parameters in the weight matrix i. Additionally, $\odot$ signifies the element-wise product between the gates at specific indices and the corresponding column or row vectors at those same indices.

To clarify, structured sparsity is achieved by training two gate vectors, $\boldsymbol{\omega}_r, \boldsymbol{\omega}_c$, where each element scales an entire row or column of a given weight matrix. These gate vectors are optimized without any additional low-rank weight matrices. Once trained, the base-model weights of the adapted layers are compressed by multiplying the gate vectors by the corresponding rows and columns of the weight matrix. This sparsification mechanism effectively reduces both the memory and time complexities of the attention layers.

Our empirical results indicate that it is possible to reduce the model size while increasing its efficiency, and still provide accurate predictions, even when training only a small set of gates.

4 THE METHOD

Consider the Transformer architecture (Vaswani, 2017) composed of L blocks, and each block consists of a multi-head self-attention (MHA) layer and a feed-forward (FFN) layer. An MHA layer with N_h heads takes an input $\mathbf{X} \in \mathbb{R}^{b\times t\times d}$, where b is the batch size, t is the sequence length (number of tokens), and d is the embedding dimension. This input is either the output of the former encoder layer or the embedding layer. The MHA layer outputs:

$$\text{MHA}(\mathbf{X}) = \sum_{i=1}^{N_h} \text{Att}[\mathbf{W}_q^{(i)}, (\mathbf{W}_k^{(i)}, \mathbf{W}_v^{(i)}, \mathbf{W}_o^{(i)}, \mathbf{X})],$$

where $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$ and $\mathbf{W}_o$ refer to the query/key/-value/output projection matrices, and $\text{Att}(\cdot)$ is an attention function. Following the attention head, the outputs are passed through the feed-forward layer, which consists of intermediate and output-projection layers, parameterized by $\mathbf{W}_{mlp}^i$ and $\mathbf{W}_{mlp}^o$:

$$\text{FFN}(X) = \text{gelu}(\mathbf{X}\mathbf{W}_{mlp}^i)\mathbf{W}_{mlp}^o.$$

Denote by $\mathbf{W}_0 \in \mathbb{R}^{k\times d}$ a pre-trained weight matrix out of $\{\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v, \mathbf{W}_o, \mathbf{W}_{mlp}^i, \mathbf{W}_{mlp}^o\}$. To enforce structured sparsity of the matrix $\mathbf{W}_0$, we propose to multiply it by the learnable *stochastic gates* vectors $\boldsymbol{\omega}_r \in \{0,1\}^{1\times k}, \boldsymbol{\omega}_c \in \{0,1\}^{1\times d}$ which are trained to converge into the binary representation. For simplicity,

we denote by $\boldsymbol{\omega}$ a general gate vector applied to rows or columns. To approximate binary values for vector $\boldsymbol{\omega}$, we learn a representation $\boldsymbol{\mu} \in [-1, 1]^{1\times d}$ which is then converted to the approximate Bernoulli variables $\boldsymbol{\omega}$, by utilizing a Gaussian-based relaxation of the Bernoulli variables (Yamada et al., 2020; Jana et al., 2023). The relaxation relies on the reparameterization trick (Miller et al., 2017; Figurnov et al., 2018) and has been demonstrated to be effective in several applications (Lindenbaum et al., 2021; Yang et al., 2023). During the training, the conversion is done by adding a random noise vector $\boldsymbol{\epsilon} \in \mathbb{R}^{1\times d}$ to the shifted by scalar 0.5 vector $\boldsymbol{\mu}$ and clipping the values by the range of $[0, 1]$,

$$\boldsymbol{\omega}(\boldsymbol{\mu}) = \max(0, \min(1, 0.5 + \boldsymbol{\mu} + \boldsymbol{\epsilon})), \tag{4}$$

where each value in the vector $\boldsymbol{\epsilon}$ is drawn from $\mathcal{N}(0, \sigma^2)$ and $\sigma = 0.5$ is fixed throughout training. To encourage the model to produce a sparse $\boldsymbol{\omega}$ vector, it is trained with the regularization loss term constrained by the given sparsity ratio s and is minimized during the training:

$$\mathcal{L}_{\text{sparse}}(\boldsymbol{\omega}) = \max(||\boldsymbol{\omega}||_0, s). \tag{5}$$

Assuming that $\boldsymbol{\omega}$ is a Bernoulli variable, we calculate its expected ℓ_0 norm as follows (the full derivation appears in Appendix A):

$$\mathbb{E}||\boldsymbol{\omega}||_0 = \frac{1}{d}\sum_j \mathbb{P}(\omega_j > 0) =$$

$$= \frac{1}{d}\sum_j \mathbb{P}(\mu_j + 0.5 + \epsilon_j > 0) =$$

$$= \frac{1}{d}\sum_j \left(\frac{1}{2} + \frac{1}{2}\,\text{erf}\left(\frac{\mu_j + 0.5}{\sqrt{2}\sigma}\right)\right).$$

When applying the sparsity term in Eq. 5, the model aims to sparsify the matrix $\mathbf{W}_0$ and retain only the essential parameters needed for the new task.

In addition, we rescale $\boldsymbol{\omega}$ by an additional score vector $\mathbf{k}$ in the sparsity loss term. This rescaling is designed to preserve vector outliers in the weight columns and rows, a crucial aspect for maintaining good performance in pruning and quantization tasks (Zhao et al., 2024a; Lin et al., 2024b; Dettmers et al., 2022):

$$\mathcal{L}'_{\text{sparse}}(\boldsymbol{\omega}) = \max(||\mathbf{k} \odot \boldsymbol{\omega}||_0, s), \tag{6}$$

where the ℓ_0 term is approximated by:

$$\mathbb{E}||\mathbf{k} \odot \boldsymbol{\omega}||_0 = \frac{1}{d}\sum_j k_j \left(\frac{1}{2} + \frac{1}{2}\,\text{erf}\left(\frac{\mu_j + 0.5}{\sqrt{2}\sigma}\right)\right),$$

and

$$k_j = e^{-\text{kurt}_j(\boldsymbol{O})} / \sum_j e^{-\text{kurt}_j(\boldsymbol{O})}, \tag{7}$$

where $\text{kurt}_j(\cdot)$ in Eq.7 is a Pearson's kurtosis (DeCarlo, 1997) computed for each column j in activations matrix $\boldsymbol{O} = [[\boldsymbol{\omega}_r \odot \mathbf{W} \odot \boldsymbol{\omega}_c]\mathbf{X}^T]$. More details can be found in the Appendix I. We require the model to close the gates (sparsify) for rows and columns with lower kurtosis and to attenuate gate closure for rows and columns with higher kurtosis.

Finally, assuming a latent representation is obtained by $\mathbf{H} = \mathbf{W}_0\mathbf{X}^T + \mathbf{b}$, our method's forward pass yields:

$$\mathbf{H} = [\boldsymbol{\omega}_r \odot \mathbf{W}_0 \odot \boldsymbol{\omega}_c]\mathbf{X}^T + \boldsymbol{\omega}_r \odot \mathbf{b}, \tag{8}$$

where $\boldsymbol{\omega}_r, \boldsymbol{\omega}_c$ are trainable parameters, and $\mathbf{b}$ is a bias parameter. Our method is depicted in Figure 2. At the start of training, we initialize the vectors $\boldsymbol{\omega}$ with all elements set to one. During training, we optimize the parameters $\boldsymbol{\omega}$ that are used to multiply the matrices $\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$, $\mathbf{W}_o$, and $\mathbf{W}_{mlp}$. Additionally, we conduct experiments with an extended version that also trains the $\boldsymbol{\Gamma}$ parameters. These parameters are used to assemble the matrices $\mathbf{W}_A$ and $\mathbf{W}_B$ for each layer, as proposed by (Hu et al., 2021).

Our method introduces a simple yet powerful mechanism for training binary gates that explicitly select the most informative rows and columns of the weight matrices. This targeted sparsification not only preserves but, in many cases, improves the accuracy of finetuning tasks compared to existing approaches. At the same time, it delivers substantial parameter reduction in the base model, with only negligible performance loss. In the following section, we present a comprehensive empirical evaluation demonstrating the effectiveness and robustness of our approach.

5 THEORETICAL ANALYSIS

5.1 FineGates vs LoRA Landscape

In Section 6, we provide empirical evidence of matching or superior performance of our method compared against LoRA, while requiring fewer trainable parameters. We provide the theoretical justification for such performance by comparing the optimization landscapes of LoRA and FineGates.

To that end, we consider a single layer and a smooth loss function (e.g., cross entropy or MSE) that satisfies the Polyak–Łojasiewicz (PL), a mild assumption often used in optimization, which guarantees that gradient descent on $\mathbf{W}$ converges linearly to a global optimum.

Definition 5.1 (Polyak–Łojasiewicz (PL) Condition)**.** A function $F(\mathbf{W})$ satisfies PL condition with parameter $\beta > 0$ if: $\frac{1}{2}||\nabla F(\mathbf{W})||^2 \geq \beta(F(\mathbf{W}) - F^*)$, where $F^* = \min_{\mathbf{W}} F(\mathbf{W})$.

Proposition 5.2 (FineGates vs LoRA optimization landscape)**.** *Let $\mathbf{W}_0 \in \mathbb{R}^{m\times n}$ to be a weights matrix in a single linear layer trained with a smooth loss function $\mathcal{L}(\mathbf{W})$ that satisfies the PL condition. Let*

- $W_{gates}(\boldsymbol{\omega}_r, \boldsymbol{\omega}_c) = Diag(\boldsymbol{\omega}_r)\mathbf{W}_0 Diag(\boldsymbol{\omega}_c)$*, where $\boldsymbol{\omega}_r \in \mathbb{R}^m, \boldsymbol{\omega}_c \in \mathbb{R}^n$ are trainable row/column gates.*
- $W_{LoRA}(\mathbf{A}, \mathbf{B}) = \mathbf{W}_0 + \mathbf{A}\mathbf{B}^\top$ *with* $\mathbf{A} \in \mathbb{R}^{m\times k}, \mathbf{B} \in \mathbb{R}^{n\times k}$.

Then the following holds:

1. *$\exists \beta_{gates} > 0$, s.t. $\forall(\boldsymbol{\omega}_r, \boldsymbol{\omega}_c)$, $\mathcal{L}(W_{gates}(\boldsymbol{\omega}_r, \boldsymbol{\omega}_c))$ satisfies PL condition with β_{gates}.*
2. *$\nexists \beta_{LoRA}$, s.t. $\mathcal{L}(W_{LoRA}(\mathbf{A}, \mathbf{B}))$ holds PL condition with β_{LoRA}.*

Compared to LoRA, the FineGates parameterization yields a substantially simpler optimization landscape. Each gate parameter directly scales a single row or column of the base weight matrix, which results in gradients that decompose into independent row- and column-wise statistics. This structure aligns well with the underlying loss geometry and, under mild assumptions such as smoothness and the PL condition, guarantees linear convergence of gradient-based training. In contrast, the LoRA parameterization introduces a bilinear factorization $\mathbf{A}\mathbf{B}^\top$, where many different choices of $(\mathbf{A}, \mathbf{B})$ correspond to the same effective update. This redundancy induces a k^2-dimensional family of flat directions in the loss surface, manifested as zero eigenvalues of the Hessian, and thereby weakens convergence guarantees. As a result, optimization with LoRA can be less stable and often requires additional normalization or careful tuning, while gating provides a more direct and well-conditioned optimization path. We provide the detailed proof in the Appendix B. In addition to the empirical results reported in Section 6, we perform a separate experiment on the MRPC dataset to further validate our proposition. Figure 3 shows the mean accuracy along with its standard deviation, computed after each training epoch while running both methods under identical hyperparameters across ten random initialization seeds.

5.2 FineGates Convergence

In this section, we present a proof of convergence for our method. This theoretical justification is necessary because including random noise in the gating mechanism could introduce challenges to the training process and affect convergence. To that end, we start by defining smoothness for functions parameterized by tensors.

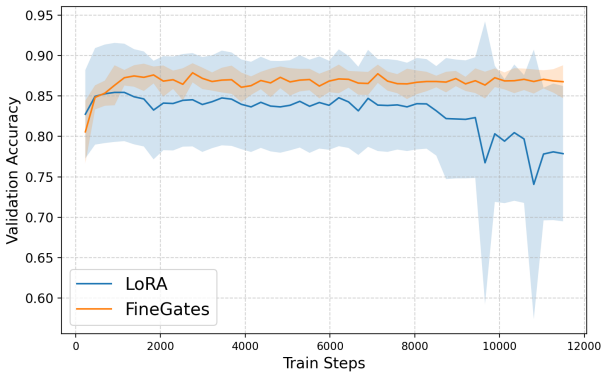

Figure 3: Validation accuracy trajectories of FineGates and LoRA on the MRPC dataset.

Definition 5.3 (L-continuity)**.** A function $\boldsymbol{h}(\mathbf{W})$ is a L-continues, if for any $\mathbf{W}_1$ and $\mathbf{W}_2$, $\|\boldsymbol{h}(\mathbf{W}_1) - \boldsymbol{h}(\mathbf{W}_2)\|_F \leq L\|\mathbf{W}_1 - \mathbf{W}_2\|_F$.

Lemma 5.4 (Convergence of FineGates)**.** *Consider the FineGates loss function $\mathcal{L}(\mathbf{W}, \boldsymbol{\omega}_r, \boldsymbol{\omega}_c)$ in (3), a network $\Phi(\cdot)$. Suppose that the composition $\mathcal{L} \equiv f \circ \Phi$ is a L-continuity, non-convex function, where f is the loss (e.g cross entropy), then $\lim_{t\to\infty} \|\nabla\mathcal{L}(\mathbf{W}_t, \boldsymbol{\omega}_{r,t}, \boldsymbol{\omega}_{c,t})\| = 0$. In particular, every accumulation point is a first-order stationary point of $\mathcal{L}$.*

The detailed proof of Proposition 5.4 can be found in Appendix H. The proof establishes the convergence of the loss function $\mathcal{L}(\mathbf{W}, \boldsymbol{\omega}_r, \boldsymbol{\omega}_c)$ under gradient descent. First, it shows that the function's gradient is Lipschitz continuous, meaning that a constant multiple of the parameter differences bounds the differences in the gradients. Then, it verifies that the regularization term is continuously differentiable and bounded, ensuring a smooth optimization process. Using the descent lemma, the proof demonstrates that the loss function decreases monotonically with each gradient step. Since the difference in loss is bounded below, the sum of gradient norms is shown to be finite, implying that the gradients approach zero. Consequently, gradient descent converges to a critical point where no further improvement is possible.

6 EXPERIMENTS

6.1 Experimental Setup and Datasets

We assess the performance of our method on downstream tasks using the RoBERTa-base, RoBERTa-large models (Liu, 2019), and Llama3.2-1B, and we

use the widely recognized GLUE benchmark (Wang et al., 2019).

We simulate real-world scenarios where only limited labeled data is available for finetuning tasks due to the challenging nature of collecting accurate ground truth labels. In addition to the complete GLUE benchmark, we conduct separate experiments on a limited GLUE dataset, limited to a maximum of 10,000 samples. Accordingly, we use the small-scale datasets (COLA, STSB, MRPC, RTE) in their entirety and select the first 10,000 labeled samples from the larger datasets (MNLI, QQP, QNLI, SST2). These experiments aim to determine whether the proposed method can compress base models with limited training data.

Moreover, we evaluate our method for base-model pruning without a specific target task. For these experiments, we apply our method during pre-training the Llama-1B model on the C4 dataset (Raffel et al., 2020). Additionally, our framework is trained with the Llama-2-7B backbone frozen, and once trained, it is evaluated on a zero-shot task. We report the median validation accuracy value for each experiment over five random initialization seeds. The number of trainable parameters (TP) excludes the classifier head, following the same setup as in previous works, e.g., (Hu et al., 2021). In sparsified pre-training (Section 6.8 and general model pruning experiments (Section 6.7), we report perplexity computed on the validation set from C4 dataset. Additional technical details are provided in the Appendix.

6.2 Baselines

We compare our method to full finetuning, in which the model is initialized to the pre-trained weights and biases, and all parameters undergo gradient updates, and several recently proposed efficient finetuning methods: LoRA (Hu et al., 2021) with rank= 4, BitFit (Zaken et al., 2021), VeRA (Kopiczko et al., 2023), LoRA-XS (Bałazy et al., 2024), $\text{RoCoFT}_{\text{1-Row}}$ (Kowsher et al., 2024), APT (Zhao et al., 2024a), and VeLoRA (Miles et al., 2024). We provide additional descriptions for the baseline methods in the Appendix R.

6.3 Accuracy Results

We present the accuracy results in Table 1. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. From Table 1, we see that our model is comparable to LoRA and full finetuning on average when applied to the RoBERTa-Base base model. In addition, our model outperforms full finetuning on average when applied to the RoBERTa-Large base model. Moreover, FineGates outperforms other efficient finetuning methods with both backbones. Table 3 presents a comparison between our method and LoRA using the modern Llama3.2-1B backbone. Despite utilizing fewer trainable parameters, our approach achieves higher accuracy than LoRA on most tasks.

Furthermore, our approach not only reduces the trainable parameter count but also compresses the base model, yielding a $10-20\%$ reduction in its parameter count while incurring an insignificant loss in accuracy (as shown in the last two rows of Table 4). This highlights the efficiency and effectiveness of our method in balancing compression and performance.

6.4 Sparsification Results

We present the sparsification results of FineGates in Figure 4. We report accuracy measurements for each sparsity level. It could be seen that our method can remove up to 20% of parameters without significant loss in Matthews correlation for the CoLA dataset, up to 40% of parameters trained on the SST2 dataset with only a loss of 4% in accuracy, and up to 40% for the STSB dataset with a loss of only 3% in Pearson Correlation metric.

6.5 Comparison to the APT method

To compare our method with the recently proposed APT model, we conducted experiments on the MRPC, STSB, SST2, and RTE datasets. We obtained results for the APT model by running it with LoRA rank values ranging from 2 to 4, using hyperparameters similar to those in our method. We fixed the target sparsity at 40% for both methods. The results are presented in Table 2. FineGates achieves performance comparable to the APT model, without the computationally intensive pruning used by APT. We would also like to emphasize the simplicity of our implementation, which does not require the efficient search tools and additional dependencies needed by the APT method.

6.6 Inference Speedup

We evaluate the inference time of the compressed model at the end of each training epoch. For every epoch, validation wall-clock time is measured 10 times per run. To account for variability, training is repeated with 3 different random seeds, yielding a total of 30 inference time measurements per epoch. All validation measurements are conducted in CPU-only environments to isolate the benefits of structured pruning. Importantly, our reductions are obtained prior to any GPU-specific optimizations or quantization methods, underscoring the intrinsic efficiency gains of our approach independent of hardware accelerators or low-level optimizations. In

Table 1: Evaluation results on the GLUE benchmark demonstrate that our method outperforms other efficient finetuning approaches as well as full finetuning, while also enabling base model **pruning** and **inference speedup**.

Method	TP	CoLA	STS-B	MRPC	RTE	SST2	MNLI	QNLI	QQP	Avg.
RoBERTa-Base										
Full Finetune	125M	63.6	90.9	90.2	80.5	94.8	87.6	92.8	91.9	86.5
Galore (Zhao et al., 2024b)	125M	60.3	90.7	92.2	79.4	94.0	87.0	92.2	91.1	85.9
LoRA($r = 4$) (Hu et al., 2021)	0.7M	64.0	90.9	89.7	83.4	94.4	87.6	92.7	91.0	86.6
BitFit (Zaken et al., 2021)	0.11M	61.8	90.8	**92.0**	77.8	93.7	85.2	91.3	84.5	84.6
VeRA (Kopiczko et al., 2023)	0.04M	65.6	90.7	89.5	78.7	94.6	-	91.8	-	85.2
$\text{RoCoFT}_{\text{1-Row}}$ (Kowsher et al., 2024)	0.08M	60.2	90.7	87.7	76.6	94.1	85.2	90.7	88.5	84.2
VeLoRA (Miles et al., 2024)	0.16M	64.6	90.8	91.3	78.0	94.4	**86.3**	92.1	**89.9**	85.9
FineGates	0.17M	**65.7**	**91.0**	90.2	**83.4**	**94.7**	85.8	**92.3**	89.2	**86.6**
RoBERTa-Large										
Full Finetune	355M	68.0	92.3	90.9	86.6	96.4	90.2	94.7	92.2	88.9
LoRA($r = 4$) (Hu et al., 2021)	1.8M	71.0	92.3	90.7	89.5	96.4	90.4	94.8	91.7	89.3
LoRA-XS (Bałazy et al., 2024)	0.06K	68.5	92.2	**91.2**	89.5	96.3	-	94.3	-	88.7
VeRA (Kopiczko et al., 2023)	0.06M	68.0	91.7	90.9	85.9	96.1	-	**94.4**	-	87.8
$\text{RoCoFT}_{\text{1-Row}}$ (Kowsher et al., 2024)	0.22M	65.7	91.8	90.0	85.3	**96.6**	**90.7**	94.2	**90.2**	88.1
FineGates	0.4M	**71.4**	**92.3**	**91.2**	**90.2**	96.0	89.1	94.1	89.4	**89.2**

Table 2: This table compares FineGates with the APT method employing a RoBERTa-Base backbone. In our model, we set the rank to 1, as we are training a single column and a single row of gate vectors.

Method	Min r	Max r	TP	Sparsity	MRPC	SST2	STSB	RTE	CoLA
APT	2	4	0.17M	**40**%	**86.0**	90.6	**89.4**	**67.5**	**45.9**
FineGates	1	1	0.17M	**40**%	84.8	**91.2**	88.1	**67.5**	40.6

Table 3: Performance of our method vs. LoRA on GLUE tasks using Llama3.2-1B backbone.

Model	TP	**CoLA**	**STS-B**	**MRPC**	**RTE**	**SST-2**
LoRA($r = 4$)	2.2M	66.1	90.1	87.7	83.7	95.9
FineGates	0.9M	63.6	91.2	88.5	84.5	93.5

this practical setting, FineGates reduces the inference time of the adopted Llama3.2-1B model on the MRPC task by up to 25%, with only a 4% loss in accuracy. More details are in Appendix F.

6.7 General Model Sparsification

Inspired by the recently proposed method for training unstructured and semi-structured pruning masks for the base model, denoted as MaskLLM (Fang et al., 2024), we conduct an additional experiment to evaluate FineGates in this scenario. We train our method using the Llama-2-7B backbone with a limited number of training tokens (TT) and evaluate its performance on the C4 validation set. Notably, our method requires only **2M** trainable parameters (TP), whereas MaskLLM requires an additional trainable mask matrix for each base model weight matrix. Moreover, MaskLLM proposes a semi-structured sparsity pattern, which is beneficial for GPU execution, whereas our method achieves structured sparsity, which is more suitable for CPU inference. In this experiment, we train FineGates without bias and freeze the parameters of the original base model. The perplexity results are shown in Table 5. The evaluation of all methods is done with a maximal sequence length of 256, which is different from the length of 4096, reported by Fang et al. (2024).

6.8 Sparse Model Pre-training

We also evaluate our method in the pretraining scenario, in which the base model is trained from scratch with the gates. We follow the same setup as suggested by Zhao et al. (2024b) and train the Llama-1B base model with 1,340M parameters on the C4 dataset. We conduct a full finetuning of the base model (FFT) and its sparsified version (FineGates + FFT) for a limited number of training tokens (TT) and present the perplexity (PPL), number of inference parameters (IP), and number of added training parameters (ATP) in Table 6. By adding only 0.09% parameters to the base model during pretraining, our method enables faster convergence of sparsified weights and structured pruning of 44% (584M parameters) of the model's weights.

7 CONCLUSION

In this work, we propose *FineGates*, a sparsification-based finetuning method for foundation language models (LMs). By learning binary stochastic gating vectors on weight matrices, FineGates introduces structured sparsity that can remove up to 40% of parameters in

Table 4: Finetuning accuracy on limited GLUE benchmark datasets. We present the accuracy results achieved by FineGates with various sparsity constraints: $s \geq 0\%$, $s \geq 10\%$, and $s \geq 20\%$. The number of removed parameters is shown in olive, and the relative change in accuracy is depicted in gray compared to full finetuning.

		Small				Limited Large			
Method	TP ↓	CoLA	STS-B	MRPC	RTE	SST2	MNLI	QNLI	QQP
RoBERTa-Base									
Full Finetune	125M	63.6	90.9	90.2	80.5	92.8	81.4	87.7	85.2
FineGates, $s \geq 0\%$	0.17M	65.7	91.0	90.2	83.4	94.0	81.3	89.1	84.9
FineGates, $s \geq 10\%$ (-12M)	0.17M	65.2 (+1.6%)	90.7(-0.2%)	89.2 (-1%)	81.3 (+0.8%)	94.0 (+1.2%)	80.6 (-0.8%)	88.8 (+1.1%)	84.3 (-0.9%)
FineGates, $s \geq 20\%$ (-25M)	0.17M	61.4 (-2.2%)	90.5 (-0.4%)	87.7 (-2.5%)	78.0 (-2.5%)	93.8 (+1%)	79.4 (-2%)	87.6 (-0.1%)	83.7 (-1.5%)
RoBERTa-Large									
Full Finetune	355M	68.0	92.3	90.9	86.6	93.5	85.9	92.4	85.8
FineGates, $s \geq 0\%$	0.4M	71.4	92.3	91.2	90.2	96.2	86.2	92.4	86.1
FineGates, $s \geq 10\%$ (-35M)	0.4M	69.5 (+1.5%)	91.8 (-0.5%)	90.4 (-0.5%)	89.2 (+2.6%)	96.1 (+2.6%)	85.0 (-0.9%)	92.1 (-0.3%)	86.1 (+0.3%)
FineGates, $s \geq 20\%$ (-70M)	0.4M	68.1 (+0.1%)	91.3 (-1%)	90.2 (-0.7%)	87.0 (+0.4%)	95.3 (+1.8%)	85.0 (-0.9%)	92.0 (-0.4%)	85.2 (-0.6%)

Table 5: Validation perplexity of the sparsified Llama-2-7B.

Model	C4	TP	TT	Sparsity
Pretrained	7.76	6,738M	2T	N/A
MaskLLM	11.15	6,738M	2B	2:4
FineGates	11.75	**2M**	**82M**	**21%**

Table 6: Pre-training Llama-1B base model from scratch with FineGates under limited training budget.

Model	PPL	TT (B)	IP (M)	ATP (M)
FFT	342.75	1.9	1,340	0
FineGates + FFT	**24.87**	1.9	**756**	1.21
FineGates + FFT	21.41	4	690	1.21

attention layers while preserving accuracy. This approach offers a more efficient alternative to conventional low-rank adaptation methods, reducing inference costs without sacrificing expressiveness. Experiments on the GLUE benchmark show that FineGates matches or outperforms existing finetuning approaches, and we further provide a theoretical analysis of its convergence and optimization landscape. Beyond task-specific adaptation, we also demonstrate the effectiveness of FineGates as a pruning strategy during pre-training, enabling faster convergence and improved inference efficiency. Overall, our results highlight structured sparsification as a powerful mechanism for efficient and scalable LM adaptation.

ACKNOWLEDGMENTS

OL was supported by the MOST grant No. 0007341. This work was conducted as part of the Ph.D. studies of Jonathan Svirsky under the supervision of Dr. Ofir Lindenbaum at Bar-Ilan University. The authors thank the anonymous reviewers for their constructive feedback and insightful discussions, which significantly contributed to improving the manuscript.

References

Bałazy, K., Banaei, M., Aberer, K., and Tabor, J. (2024). Lora-xs: Low-rank adaptation with extremely small number of parameters. *arXiv preprint arXiv:2405.17604*.

Bondarenko, Y., Del Chiaro, R., and Nagel, M. (2024). Low-rank quantization-aware training for llms. *arXiv preprint arXiv:2406.06385*.

Ceritli, T., Ozkan, S., Min, J., Noh, E., Min, C., and Ozay, M. (2024). A study of parameter efficient finetuning by learning to efficiently fine-tune. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 15819–15836.

Chavan, A., Liu, Z., Gupta, D., Xing, E., and Shen, Z. (2023). One-for-all: Generalized lora for parameter-efficient fine-tuning. *arXiv preprint arXiv:2306.07967*.

DeCarlo, L. T. (1997). On the meaning and use of kurtosis. *Psychological methods*, 2(3):292.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. (2022). Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332.

Fang, G., Yin, H., Muralidharan, S., Heinrich, G., Pool, J., Kautz, J., Molchanov, P., and Wang, X. (2024). Maskllm: Learnable semi-structured sparsity for large language models. *Advances in Neural Information Processing Systems*, 37:7736–7758.

Figurnov, M., Mohamed, S., and Mnih, A. (2018). Implicit reparameterization gradients. *Advances in neural information processing systems*, 31.

Frantar, E. and Alistarh, D. (2023). Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Gordon, M. A., Duh, K., and Andrews, N. (2020). Compressing bert: Studying the effects of weight pruning on transfer learning. *arXiv preprint arXiv:2002.08307*.

He, S., Ding, L., Dong, D., Zhang, M., and Tao, D. (2022). Sparseadapter: An easy approach for improving the parameter-efficiency of adapters. *arXiv preprint arXiv:2210.04284*.

Hinton, G. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2021). Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Jana, S., Li, H., Yamada, Y., and Lindenbaum, O. (2023). Support recovery with projected stochastic gates: Theory and application for linear models. *Signal Processing*, 213:109193.

Kopiczko, D. J., Blankevoort, T., and Asano, Y. M. (2023). Vera: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*.

Kowsher, M., Esmaeilbeig, T., Yu, C.-N., Soltanalian, M., and Yousefi, N. (2024). Rocoft: Efficient finetuning of large language models with row-column updates. *arXiv preprint arXiv:2410.10075*.

Lei, Z., Qian, D., and Cheung, W. (2024). Fast randomized low-rank adaptation of pre-trained language models with pac regularization. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 5236–5249.

Lester, B., Al-Rfou, R., and Constant, N. (2021). The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Li, X. L. and Liang, P. (2021). Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Li, Y., Luo, F., Tan, C., Wang, M., Huang, S., Li, S., and Bai, J. (2022). Parameter-efficient sparsity for large language models fine-tuning. *arXiv preprint arXiv:2205.11005*.

Lin, C., Li, L., Li, D., Zou, J., Luo, W., Xue, W., and Guo, Y. (2024a). Nora: Nested low-rank adaptation for efficient fine-tuning large models. *arXiv preprint arXiv:2408.10280*.

Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., and Han, S. (2024b). Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100.

Lindenbaum, O., Salhov, M., Averbuch, A., and Kluger, Y. (2021). L0-sparse canonical correlation analysis. In *International Conference on Learning Representations*.

Liu, Y. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Loshchilov, I. and Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Ma, X., Fang, G., and Wang, X. (2023). Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720.

Miles, R., Reddy, P., Elezi, I., and Deng, J. (2024). Velora: Memory efficient training using rank-1 subtoken projections. *Advances in Neural Information Processing Systems*, 37:42292–42310.

Miller, A., Foti, N., D'Amour, A., and Adams, R. P. (2017). Reducing reparameterization gradient variance. *Advances in Neural Information Processing Systems*, 30.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. (2018). Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Pfeiffer, J., Kamath, A., Rücklé, A., Cho, K., and Gurevych, I. (2020). Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Refael, Y., Arbel, I., and Huleihel, W. (2024a). Learning k-level structured sparse neural networks using group envelope regularization.

Refael, Y., Arbel, I., Lindenbaum, O., and Tirer, T. (2025a). Lorenza: Enhancing generalization in low-rank gradient llm training via efficient zeroth-order adaptive sam.

Refael, Y., Smorodinsky, G., Tirer, T., and Lindenbaum, O. (2025b). Sumo: Subspace-aware moment-orthogonalization for accelerating memory-efficient llm training.

Refael, Y., Svirsky, J., Shustin, B., Huleihel, W., and Lindenbaum, O. (2024b). Adarankgrad: Adaptive gradient-rank and moments for memory-efficient

llms training and fine-tuning. *arXiv preprint arXiv:2410.17881*.

Rozner, A., Battash, B., Wolf, L., and Lindenbaum, O. (2024). Knowledge editing in language models via adapted direct preference optimization. *arXiv preprint arXiv:2406.09920*.

Salas, H. N. (1999). Gershgorin's theorem for matrices of operators. *Linear algebra and its applications*, 291(1-3):15–36.

Sanh, V., Wolf, T., and Rush, A. (2020). Movement pruning: Adaptive sparsity by fine-tuning. *Advances in neural information processing systems*, 33:20378–20389.

Vaswani, A. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*.

Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. (2019). Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.

Wang, S., Yu, L., and Li, J. (2024). Lora-ga: Low-rank adaptation with gradient approximation. *Advances in Neural Information Processing Systems*, 37:54905–54931.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. (2016). Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29.

Xia, M., Zhong, Z., and Chen, D. (2022). Structured pruning learns compact and accurate models. *arXiv preprint arXiv:2204.00408*.

Xu, Y., Xie, L., Gu, X., Chen, X., Chang, H., Zhang, H., Chen, Z., Zhang, X., and Tian, Q. (2023). Qa-lora: Quantization-aware low-rank adaptation of large language models. *arXiv preprint arXiv:2309.14717*.

Yamada, Y., Lindenbaum, O., Negahban, S., and Kluger, Y. (2020). Feature selection using stochastic gates. In *International Conference on Machine Learning*, pages 10648–10659. PMLR.

Yang, J., Lindenbaum, O., Kluger, Y., and Jaffe, A. (2023). Multi-modal differentiable unsupervised feature selection. In *Uncertainty in Artificial Intelligence*, pages 2400–2410. PMLR.

Zaken, E. B., Ravfogel, S., and Goldberg, Y. (2021). Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

Zhang, J.-C., Xiong, Y.-J., Xia, C.-M., Zhu, D.-H., and Qiu, X.-H. (2025). Parameter-efficient fine-tuning of large language models via deconvolution in subspace. In Rambow, O., Wanner, L., Apidianaki, M., Al-Khalifa, H., Eugenio, B. D., and Schockaert, S., editors, *Proceedings of the 31st International Conference on Computational Linguistics*, pages 3924–3935, Abu Dhabi, UAE. Association for Computational Linguistics.

Zhang, L., Zhang, L., Shi, S., Chu, X., and Li, B. (2023a). Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*.

Zhang, Q., Chen, M., Bukharin, A., Karampatziakis, N., He, P., Cheng, Y., Chen, W., and Zhao, T. (2023b). Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*.

Zhao, B., Hajishirzi, H., and Cao, Q. (2024a). Apt: Adaptive pruning and tuning pretrained language models for efficient training and inference. *arXiv preprint arXiv:2401.12200*.

Zhao, J., Zhang, Z., Chen, B., Wang, Z., Anandkumar, A., and Tian, Y. (2024b). Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [No]
2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]
3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [No]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]
5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Train Less, Infer Faster: Efficient Model Finetuning and Compression via Structured Sparsity: Supplementary Materials

A Expected ℓ_0 norm

$$\mathbb{E}||\boldsymbol{\omega}||_0 = \frac{1}{d}\sum_j \mathbb{P}(\omega_j > 0) = \frac{1}{d}\sum_j \mathbb{P}(\mu_j + 0.5 + \epsilon_j > 0) =$$

$$= \frac{1}{d}\sum_j (1 - \mathbb{P}(\mu_j + 0.5 + \epsilon_j \leq 0)) =$$

$$= \frac{1}{d}\sum_j \left(1 - \Phi\left(\frac{-\mu_j - 0.5}{\sigma}\right)\right) =$$

$$= \frac{1}{d}\sum_j 1 - \frac{1}{2}\left(1 + \operatorname{erf}\left(-\frac{\mu_j + 0.5}{\sqrt{2}\sigma}\right)\right) =$$

$$= \frac{1}{d}\sum_j \left(\frac{1}{2} + \frac{1}{2}\operatorname{erf}\left(\frac{\mu_j + 0.5}{\sqrt{2}\sigma}\right)\right).$$

B Proof of Proposition 5.2

Setup and notation. Let $L : \mathbb{R}^{m\times n} \to \mathbb{R}_+$ be differentiable and satisfy the Polyak–Łojasiewicz (PL) inequality on a set $\mathcal{W} \subseteq \mathbb{R}^{m\times n}$ with parameter $\beta > 0$:

$$\frac{1}{2}\,\|\nabla_{\mathbf{W}} L(\mathbf{W})\|_F^2 \;\geq\; \beta\big(L(\mathbf{W}) - L^\star\big) \qquad \forall\, \mathbf{W} \in \mathcal{W}, \tag{9}$$

where $L^\star = \inf_{\mathbf{W}\in\mathcal{W}} L(\mathbf{W})$. Write $\mathrm{vec}(\cdot)$ for vectorization and use the Euclidean/Frobenius norms throughout.

Two-sided gates parametrization. Fix a base matrix $\mathbf{W}_0 \in \mathbb{R}^{m\times n}$. For row/column gates $\boldsymbol{\omega}_r \in \mathbb{R}^m$, $\boldsymbol{\omega}_c \in \mathbb{R}^n$, define

$$W(\boldsymbol{\omega}_r, \boldsymbol{\omega}_c) \;=\; \mathrm{Diag}(\boldsymbol{\omega}_r)\,\mathbf{W}_0\,\mathrm{Diag}(\boldsymbol{\omega}_c),$$

i.e., $\mathbf{W}_{ij} = \boldsymbol{\omega}_{r,i}\,\mathbf{W}_{0,ij}\,\boldsymbol{\omega}_{c,j}$. Consider the feasible set

$$\Omega \;=\; \Big\{(\boldsymbol{\omega}_r, \boldsymbol{\omega}_c) \in \{0,1\}^m \times \{0,1\}^n :\; \|\boldsymbol{\omega}_r\|_0 \geq 1 - s,\; \|\boldsymbol{\omega}_c\|_0 \geq 1 - s\Big\},$$

where s is the sparsity level and let $\mathcal{W}_\Omega := \{W(\boldsymbol{\omega}_r, \boldsymbol{\omega}_c) : (\boldsymbol{\omega}_r, \boldsymbol{\omega}_c) \in \Omega\}$.

We also consider its natural continuous relaxation $\widehat{\Omega}$:

$$\widehat{\Omega} \;=\; \Big\{(\boldsymbol{\omega}_r, \boldsymbol{\omega}_c) \in [\alpha, 1]^m \times [\alpha, 1]^n :\; \|\boldsymbol{\omega}_r\|_0 \geq 1 - s,\; \|\boldsymbol{\omega}_c\|_0 \geq 1 - s\Big\},$$

with any fixed $\alpha \in (0, 1]$ (so no active row/column is exactly zero during the first iteration of training).

Assumptions.

1. There exist active index sets $S_r \subseteq [m]$, $S_c \subseteq [n]$ with $|S_r| \geq 1-s$, $|S_c| \geq 1-s$ such that on $S_r \times S_c$ the base weights are bounded away from zero:
$$\min_{i \in S_r,\, j \in S_c} |\mathbf{W}_{0,ij}| \;\geq\; w_0 \;>\; 0.$$
(Equivalently, $\mathbf{W}_0$ has no vanishing entries on any row/column that the constraint allows to be active.)
2. We optimize over the relaxed feasible set $\widehat{\Omega}$ (piecewise-smooth; the discrete case is covered as a limit).

Lemma B.1 (Chain-rule lower bound via Jacobian)**.** *Let $F(\boldsymbol{\theta}) := L\big(W(\boldsymbol{\theta})\big)$ where $\boldsymbol{\theta} := (\boldsymbol{\omega}_r, \boldsymbol{\omega}_c) \in \widehat{\Omega}$. If, on $\widehat{\Omega}$,*
$$\sigma_{\min}\big(J(\boldsymbol{\theta})\big) \;\geq\; \gamma \;>\; 0 \qquad \text{where } J(\boldsymbol{\theta}) := \frac{\partial \operatorname{vec}(W(\boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} \in \mathbb{R}^{mn \times (m+n)},$$
then F satisfies the PL inequality on $\widehat{\Omega}$ with parameter $\beta\gamma^2$:
$$\frac{1}{2}\,\|\nabla_{\boldsymbol{\theta}} F(\boldsymbol{\theta})\|_2^2 \;\geq\; \beta\,\gamma^2\,\big(F(\boldsymbol{\theta}) - L^\star\big), \qquad \forall\, \boldsymbol{\theta} \in \widehat{\Omega}.$$

Proof. *By the chain rule, $\nabla_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) = J(\boldsymbol{\theta})^\top \nabla_{\operatorname{vec}(W)} L(W(\boldsymbol{\theta}))$. Hence*
$$\|\nabla_{\boldsymbol{\theta}} F(\boldsymbol{\theta})\|_2 \;=\; \|J(\boldsymbol{\theta})^\top \nabla_{\operatorname{vec}(W)} L\|_2 \;\geq\; \sigma_{\min}(J(\boldsymbol{\theta}))\,\|\nabla_{\operatorname{vec}(W)} L\|_2.$$
Squaring, using $\sigma_{\min}(J) \geq \gamma$, the Frobenius/Euclidean equivalence, and equation 9 gives the claim. □

Lemma B.2 (Uniform lower bound on the Jacobian for gates)**.** *Under 1–2, there exists $\gamma > 0$ depending only on $(s, \underline{\alpha}, \underline{w}_0)$ such that $\sigma_{\min}(J(\boldsymbol{\theta})) \geq \gamma$ for all $\boldsymbol{\theta} \in \widehat{\Omega}$.*

Proof. Write $\mathbf{W}_{ij} = \boldsymbol{\omega}_{r,i}\,\mathbf{W}_{0,ij}\,\boldsymbol{\omega}_{c,j}$. The partials are
$$\frac{\partial \mathbf{W}_{ij}}{\partial \boldsymbol{\omega}_{r,i}} = \mathbf{W}_{0,ij}\,\boldsymbol{\omega}_{c,j}, \qquad \frac{\partial \mathbf{W}_{ij}}{\partial \boldsymbol{\omega}_{c,j}} = \boldsymbol{\omega}_{r,i}\,\mathbf{W}_{0,ij}, \qquad \text{others } 0.$$
Thus $J(\boldsymbol{\theta})$ has two block-columns corresponding to $(\boldsymbol{\omega}_r, \boldsymbol{\omega}_c)$ whose nonzeros on $S_r \times S_c$ are bounded below in magnitude by $\underline{w}_0\,\underline{\alpha}$. Moreover, the $mn \times (m+n)$ Gram matrix $G(\boldsymbol{\theta}) := J(\boldsymbol{\theta})^\top J(\boldsymbol{\theta})$ is block-structured:
$$G(\boldsymbol{\theta}) = \begin{bmatrix} \operatorname{Diag}\big(g_r(\boldsymbol{\omega}_c)\big) & H(\boldsymbol{\theta}) \\ H(\boldsymbol{\theta})^\top & \operatorname{Diag}\big(g_c(\boldsymbol{\omega}_r)\big) \end{bmatrix},$$
where $g_r(\boldsymbol{\omega}_c)_i = \sum_j (W_{0,ij}\boldsymbol{\omega}_{c,j})^2$ and $g_c(\boldsymbol{\omega}_r)_j = \sum_i (W_{0,ij}\boldsymbol{\omega}_{r,i})^2$. On the active sets, $g_r(\boldsymbol{\omega}_c)_i \geq s\,\underline{w}_0^2\,\underline{\alpha}^2$ and $g_c(\boldsymbol{\omega}_r)_j \geq s\,\underline{w}_0^2\,\underline{\alpha}^2$. By Gershgorin theorem (Salas, 1999), the minimal eigenvalue of $G(\boldsymbol{\theta})$ is bounded below by a constant $c(s, \underline{\alpha}, \underline{w}_0) > 0$ independent of $\boldsymbol{\theta} \in \widehat{\Omega}$ (dependent only on the diaogonal values). Hence $\sigma_{\min}(J(\boldsymbol{\theta})) = \sqrt{\lambda_{\min}(G(\boldsymbol{\theta}))} \geq \gamma := \sqrt{c} > 0$. □

Proposition B.3 (PL is preserved under two-sided gates)**.** *Since L satisfies PL condition, the composed objective*
$$F(\boldsymbol{\omega}_r, \boldsymbol{\omega}_c) \;:=\; L\big(\operatorname{Diag}(\boldsymbol{\omega}_r)\,\mathbf{W}_0\,\operatorname{Diag}(\boldsymbol{\omega}_c)\big)$$
satisfies the PL inequality on $\widehat{\Omega}$ with parameter $\beta\,\gamma^2$, where $\gamma > 0$ is the uniform Jacobian lower bound from Lemma B.2. Proof. *Combine Lemmas B.1 and B.2.* □

Why the LoRA parametrization does *not* preserve PL. Consider the same base loss L but the LoRA map $W(\boldsymbol{A}, \boldsymbol{B}) = \mathbf{W}_0 + \boldsymbol{A}\boldsymbol{B}$ with $\boldsymbol{A} \in \mathbb{R}^{m \times r}$, $\boldsymbol{B} \in \mathbb{R}^{r \times n}$. Even if L is PL in W on $\mathbb{R}^{m \times n}$, the composed
$$F_{\text{LoRA}}(\boldsymbol{A}, \boldsymbol{B}) \;:=\; L(\mathbf{W}_0 + \boldsymbol{A}\boldsymbol{B})$$
fails the PL inequality in general.

Proposition B.4 (Counterexample for LoRA)**.** *Let $L(\mathbf{W}) = \frac{1}{2}\|\mathbf{W} - \mathbf{W}^\star\|_F^2$, which satisfies the PL inequality on $\mathbb{R}^{m \times n}$ with parameter $\beta = 1$. If $\mathbf{W}^\star \neq \mathbf{W}_0$, then $F_{\text{LoRA}}(\boldsymbol{A}, \boldsymbol{B}) = \frac{1}{2}\|\mathbf{W}_0 + \boldsymbol{A}\boldsymbol{B} - \mathbf{W}^\star\|_F^2$ does not satisfy the PL inequality on $\mathbb{R}^{m \times r} \times \mathbb{R}^{r \times n}$.* Proof. *At $(\boldsymbol{A}, \boldsymbol{B}) = (0, 0)$ we have $\nabla_A F_{\text{LoRA}} = (\mathbf{W}_0 + \boldsymbol{A}\boldsymbol{B} - \mathbf{W}^\star)\boldsymbol{B}^\top = 0$ and $\nabla_B F_{\text{LoRA}} = \boldsymbol{A}^\top(\mathbf{W}_0 + \boldsymbol{A}\boldsymbol{B} - \mathbf{W}^\star) = 0$, so $\|\nabla F_{\text{LoRA}}(0,0)\|_F = 0$. But $F_{\text{LoRA}}(0,0) = \frac{1}{2}\|\mathbf{W}_0 - \mathbf{W}^\star\|_F^2 > 0$ by $\mathbf{W}^\star \neq \mathbf{W}_0$. The PL inequality would force $\|\nabla F\|^2 \geq 2\beta(F - F^\star)$, which is violated here since the left-hand side is 0 while the right-hand side is positive (note $F^\star \geq 0$). Hence F_{LoRA} is not PL.* □

C Sparsification Results

Sparsification-Accuracy trade-off measured on CoLA, SST2 (full), STSB, and MRPC datasets with RoBERTa-Base, RoBERTa-Large, and Llama3.2-1B backbones is shown in Figure 4.

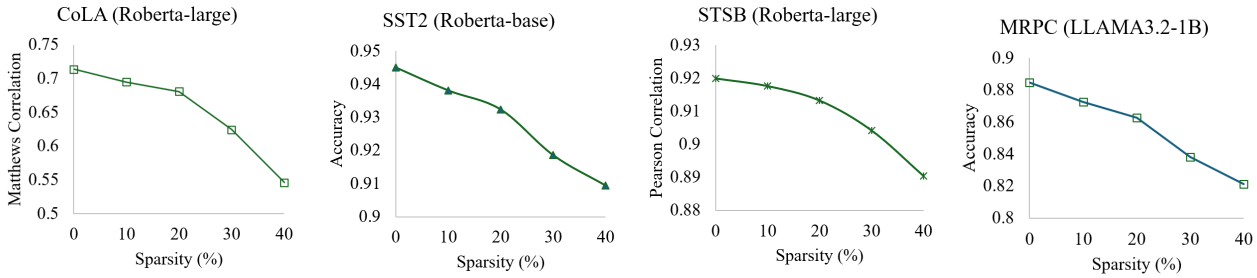

Figure 4: Sparsification-Accuracy trade-off measured on CoLA, SST2 (full), STSB, and MRPC datasets with RoBERTa-Base, RoBERTa-Large, and Llama3.2-1B backbones. Our model provides $> \mathbf{40}\%$ of structured sparsity while sacrificing only $\mathbf{4}\%$ of accuracy compared to the model without sparsification on the SST2 dataset, where we train $\boldsymbol{\omega}_r, \boldsymbol{\omega}_c$ with total $166K$ parameters. On CoLA, the method reduces up to 20% of parameters without significant loss in accuracy, and 40% on the STSB dataset with only 3% drop in accuracy. The method removes up to $470M$ parameters from the Llama3.2-1B base model with only 6% accuracy loss.

D Convergence Experiment

We train both FineGates and LoRA on MRPC dataset for 50 epochs with a batch size of 16, learning rate $1 - e10^{-3}$. In Figure 3 we show validation accuracy. We repeat the experiment 10 times and compute mean±std values after every epoch.

E FineGates Modifications

We conduct additional experiments to examine whether adapting fewer projection matrices in the transformer layers significantly affects the performance of our approach. Furthermore, we investigate the effect of extending our model with additional low-rank weights $\mathbf{W}_A$ and $\mathbf{W}_B$ for each adapted layer, following the design proposed by (Hu et al., 2021). Table 7 reports results for three model variants on RoBERTa-Base and RoBERTa-Large: FineGates w/o $\mathbf{W}_{mlp}$ that does not adapt intermediate and output projections for RoBERTa layers, FineGates w/ $\mathbf{W}_B\mathbf{W}_A$ adds low-rank matrices $\mathbf{W}_B, \mathbf{W}_A$ with $r = 8$ and FineGates w $\mathbf{W}_B\mathbf{W}_A$ but w/o $\mathbf{W}_{mlp}$ which combines the two modifications. From this experiment, we draw three main conclusions: (1) the base FineGates model, where only gates are trained, achieves accuracy comparable to the other variants; (2) the number of trainable parameters can be reduced by up to 4 times without significantly compromising accuracy (FineGates w/o $\mathbf{W}_{mlp}$); and (3) introducing low-rank parameter matrices yields additional accuracy gains.

F Inference Time Measurements

We evaluate the inference time of the compressed model after each training epoch. For each epoch, the wall-clock time of the validation phase is measured 10 times per run. To account for variability, we train with three different random seeds, resulting in a total of 30 inference time measurements per epoch. All validation-time measurements are performed on CPU only. Our experiments use the MRPC dataset with the LLaMA-3.2-1B backbone. To assess inference efficiency, we compute the average validation time over a 5-epoch window and report the relative reduction in inference time using the first epoch as a reference. Specifically, the reduction is defined as $(t_0 - t_i)/t_0$, where t_i denotes the average validation time after epoch i, aggregated over 30 measurements (3 seeds $\times$ 10 repetitions per seed). We report timing results alongside validation accuracy (averaged over three seeds) in Figure 5, and in relation to the number of pruned parameters in Figure 6.

Table 7: Modifications of FineGates. The evaluation is done for the next versions of FineGates: (1) training without gates on feed-forward layers (w/o $\mathbf{W}_{mlp}$), (2) training with additional low-rank parameters (w/ $\mathbf{W}_B\mathbf{W}_A$), (3) training with low-rank parameters (w/ $\mathbf{W}_B\mathbf{W}_A$) but excluding feed-forward layers (w/o $\mathbf{W}_{mlp}$) matrices from adaptation.

		Small				Sub-sampled Large			
Method	TP ↓	CoLA	STS-B	MRPC	RTE	SST2	MNLI	QNLI	QQP
Roberta-Base									
FineGates	0.17M	65.7	91.0	90.0	83.4	94.0	81.3	89.1	84.9
FineGates w/o $\mathbf{W}_{mlp}$	0.04M	65.1	90.9	**90.4**	80.5	94.2	81.0	89.2	84.7
FineGates w/ $\mathbf{W}_B\mathbf{W}_A$	1.4M	**66.5**	**91.2**	90.2	**83.8**	**94.3**	81.4	89.4	84.8
FineGates w/ $\mathbf{W}_B\mathbf{W}_A$ w/o $\mathbf{W}_{mlp}$	0.6M	65.8	90.7	89.7	82.3	**94.3**	**81.7**	89.7	**85.2**
Roberta-Large									
FineGates	0.4M	**71.4**	92.3	91.2	90.2	**96.2**	86.2	92.4	86.1
FineGates w/o $\mathbf{W}_{mlp}$	0.1M	70.5	92.0	91.4	**90.6**	96.1	86.6	92.3	86.4
FineGates w/ $\mathbf{W}_B\mathbf{W}_A$	3.8M	70.1	92.2	90.6	88.1	95.4	86.1	92.1	85.5
FineGates w/ $\mathbf{W}_B\mathbf{W}_A$ w/o $\mathbf{W}_{mlp}$	1.7M	69.9	**92.6**	**91.9**	88.5	**96.2**	**87.2**	**92.7**	**86.7**

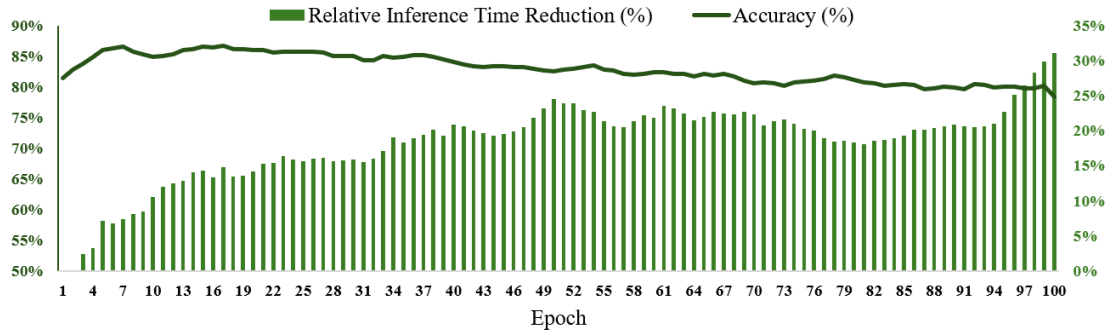

Figure 5: CPU Inference time measurements along with validation accuracy averaged across 3 seeds for a single validation epoch of MRPC dataset.

G Wall-clock Times measurements for APT and FineGates

In Table 8 we show the total training time measured for APT, FineGates and fully finetuned model. APT was trained without distillation and FineGates was trained without kurtosis score.

Table 8: To compare wall-clock times, we trained both models on the SST-2 data for 50 epochs, measuring the total training time using the RoBERTa-base backbone.

Model	**Time (seconds)**	**Peak GPU Memory (MB)**
Full Finetune	11,521	3,257
APT	21,542	3,291
FineGates	19,284	2,477

H Proof of Lemma 5.4

Proof. Write $\mathcal{L}(\mathbf{W}, \boldsymbol{\omega}_r, \boldsymbol{\omega}_c) := f\big(\operatorname{Diag}(\boldsymbol{\omega}_r)\mathbf{W}\operatorname{Diag}(\boldsymbol{\omega}_c)\big)$ and let $G := \nabla f(\widetilde{\mathbf{W}}) \in \mathbb{R}^{k\times d}$ evaluated at $\widetilde{\mathbf{W}} = \operatorname{Diag}(\boldsymbol{\omega}_r)\mathbf{W}\operatorname{Diag}(\boldsymbol{\omega}_c)$. By the chain rule,

$$\nabla_{\mathbf{W}} F \;=\; \operatorname{Diag}(\boldsymbol{\omega}_r)\, G\, \operatorname{Diag}(\boldsymbol{\omega}_c) \;=\; G \odot (\boldsymbol{\omega}_r\, \boldsymbol{\omega}_c^{\top}).$$

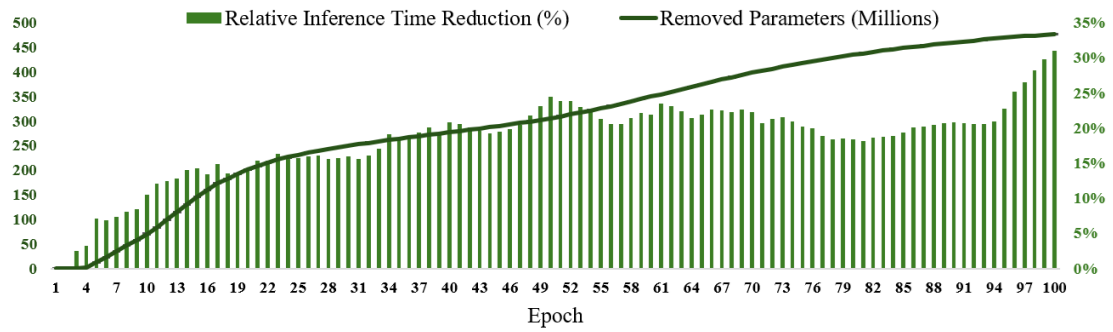

Figure 6: CPU Inference time measurements along with number of removed parameters averaged across 3 seeds for a single validation epoch of the MRPC dataset.

For the gate variables, using $\partial\widetilde{\mathbf{W}}/\partial(\boldsymbol{\omega}_r)_i = e_i e_i^\top \mathbf{W}\,\mathrm{Diag}(\boldsymbol{\omega}_c)$ and $\partial\widetilde{\mathbf{W}}/\partial(\boldsymbol{\omega}_c)_j = \mathrm{Diag}(\boldsymbol{\omega}_r)\mathbf{W}e_j e_j^\top$, we obtain

$$\big(\nabla_{\boldsymbol{\omega}_r} F\big)_i = \big\langle G,\, e_i e_i^\top \mathbf{W}\,\mathrm{Diag}(\boldsymbol{\omega}_c)\big\rangle = \sum_{j=1}^{d} G_{ij}\,\mathbf{W}_{ij}\,(\boldsymbol{\omega}_c)_j = \Big(\big(G \odot \mathbf{W}\big)\,\boldsymbol{\omega}_c\Big)_i,$$

$$\big(\nabla_{\boldsymbol{\omega}_c} F\big)_j = \big\langle G,\, \mathrm{Diag}(\boldsymbol{\omega}_r)\mathbf{W}e_j e_j^\top\big\rangle = \sum_{i=1}^{k} G_{ij}\,\mathbf{W}_{ij}\,(\boldsymbol{\omega}_r)_i = \Big(\big(G \odot \mathbf{W}\big)^\top \boldsymbol{\omega}_r\Big)_j.$$

Because ∇f is L_f-Lipschitz, the map $(\mathbf{W}, \boldsymbol{\omega}_r, \boldsymbol{\omega}_c) \mapsto G$ is Lipschitz on any bounded set. Multiplication by $\mathrm{Diag}(\boldsymbol{\omega}_r)$ and $\mathrm{Diag}(\boldsymbol{\omega}_c)$ (with $\boldsymbol{\omega}_r, \boldsymbol{\omega}_c \in [0,1]$) is linear and non-expansive, hence the composite maps to $\nabla_{\mathbf{W}} F$, $\nabla_{\boldsymbol{\omega}_r} F$, and $\nabla_{\boldsymbol{\omega}_c} F$ are Lipschitz on bounded sets. Since h_r, h_c are smooth on $[0,1]$, their gradients are Lipschitz there; therefore,

$$\nabla L \;=\; \nabla F \;+\; \lambda\big(0, \nabla h_r, \nabla h_c\big)$$

is L_L-Lipschitz on a bounded set containing the iterates.

Let $\mathbf{z}_t := (\mathbf{W}_t, \boldsymbol{\omega}_{r,t}, \boldsymbol{\omega}_{c,t})$ and take one gradient step $\mathbf{z}_{t+1} = \mathbf{z}_t - \eta \nabla L(\mathbf{z}_t)$ with $\eta \in (0, 2/L_L)$. By the descent lemma,

$$L(\mathbf{z}_{t+1}) \;\le\; L(\mathbf{z}_t) - \eta\Big(1 - \tfrac{\eta L_L}{2}\Big)\,\|\nabla L(\mathbf{z}_t)\|^2,$$

so $L(\mathbf{z}_t)$ is monotonically non-increasing. Since f is bounded below and h_r, h_c are bounded below on $[0,1]$, L is bounded below; summing the inequality yields $\sum_t \|\nabla L(\mathbf{z}_t)\|^2 < \infty$, hence $\|\nabla L(\mathbf{z}_t)\| \to 0$. Any accumulation point therefore satisfies the first-order optimality condition. □

Remark H.1 (Bias gating and one-sided special cases). If a bias b is gated as $\widetilde{b} = \boldsymbol{\omega}_r \odot b$ (or left ungated), the proof extends verbatim via standard column-augmentation. Setting $\boldsymbol{\omega}_r = \mathbf{1}$ or $\boldsymbol{\omega}_c = \mathbf{1}$ recovers the one-sided gating case.

I Kurtosis Score

Denote by $\mathbf{W} \in \mathbb{R}^{k\times d}$ a weight matrix selected from $\{\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v, \mathbf{W}_o, \mathbf{W}^i_{mlp}, \mathbf{W}^o_{mlp}\}$ in attention layer. For a given hidden representation $\boldsymbol{X} \in \mathbb{R}^{b\times t\times d}$, we compute the activation matrix $\boldsymbol{O} \in \mathbb{R}^{k\times d}$ by first averaging the tensor $\boldsymbol{X}$ in batch size and sequence length dimensions to obtain $\mathbf{x}' \in \mathbb{R}^{1\times d}$ by:

$$\mathbf{x}' = \frac{1}{b \times t} \sum_{i=1}^{b} \sum_{j=1}^{t} \boldsymbol{X}_{i,j} \tag{10}$$

and then compute the activation matrix by:

$$\boldsymbol{O} = [\boldsymbol{\omega}_r \odot \boldsymbol{W} \odot \boldsymbol{\omega}_c] \odot \mathbf{x}' \tag{11}$$

where $\odot$ denotes element-wise multiplication and $\boldsymbol{\omega}_r, \boldsymbol{\omega}_c$ are gating vectors.

We then compute the kurtosis vectors $\boldsymbol{k}_{\text{cols}} = \text{kurt}(\boldsymbol{O}) \in \mathbb{R}^{1\times d}$ and $\boldsymbol{k}_{\text{rows}} = \text{kurt}(\boldsymbol{O}^T) \in \mathbb{R}^{1\times k}$, where $\text{kurt}(\cdot)$ denotes Pearson's kurtosis (DeCarlo, 1997).

Note that aggregating tensor $\boldsymbol{X}$ by averaging over the batch and sequence dimensions is a simplification. A more accurate approach would compute kurtosis vectors for each batch and sequence element individually, following by averaging the vectors $\boldsymbol{k}_{\text{cols}}, \boldsymbol{k}_{\text{rows}}$. However, in practice, this computation is considerably more expensive and requires substantially more GPU memory.

J Statistics of the learned gates across different Transformer layers

We analyze the statistics of the learned gates across different Transformer layers (Figure 7), and observe that the final layers tend to become sparse earlier in training. This behavior is consistent with prior findings, such as Sanh et al. (2020) and Gordon et al. (2020), which report that later layers are more amenable to pruning without significantly harming performance.

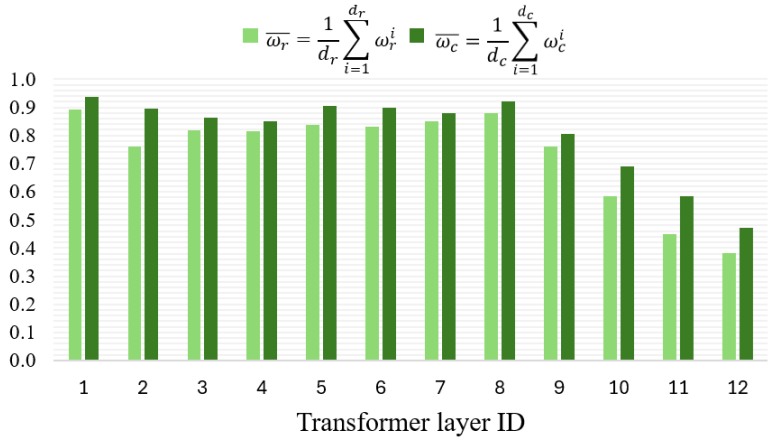

Figure 7: For each Transformer layer in Roberta-Base model we present the gates values averaged for each gates row vector, $\frac{1}{d_r}\sum_{i=1}^{d_r} \boldsymbol{\omega}_r^i$, and gates column vector $\frac{1}{d_c}\sum_{i=1}^{d_c} \boldsymbol{\omega}_c^i$, where d_r, d_c are the dimensions of gates vectors $\boldsymbol{\omega}_r, \boldsymbol{\omega}_c$ correspondingly.

K LLAMA-1B Model

Table 9: Model architecture for the pretraining experiment.

Params	Hidden	Intermediate	Heads	Layers	Steps	Data Amount
1B	2,048	5,461	24	32	20K	2B

We set the maximum sequence length to 256 for all models, with a batch size of 512. We apply learning-rate warmup for the first 10,000 training steps, followed by a cosine annealing schedule that decays to 10% of the initial learning rate. Training is stopped after processing 2B tokens from the training dataset. The model architecture is summarized in Table 9, and we use the T5-base tokenizer for this experiment.

L MaskLLM Evaluation

We evaluate the pre-trained model using the original implementation provided by the authors[2] . The masking model we tested was trained by the authors on a subset of the C4 dataset. While the original results were reported with a maximum sequence length of 4096, due to our limited computational budget we evaluate it with

[2]https://github.com/NVlabs/MaskLLM

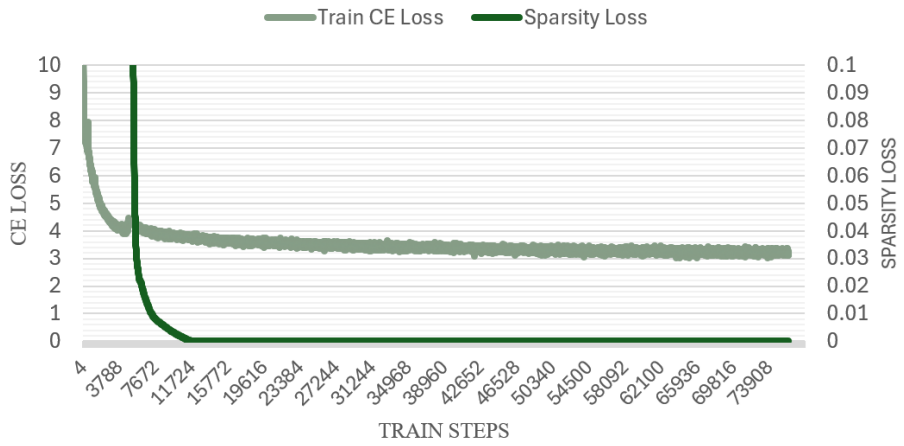

Figure 8: Train loss convergence of FineGates + FFT method in Section 5.7

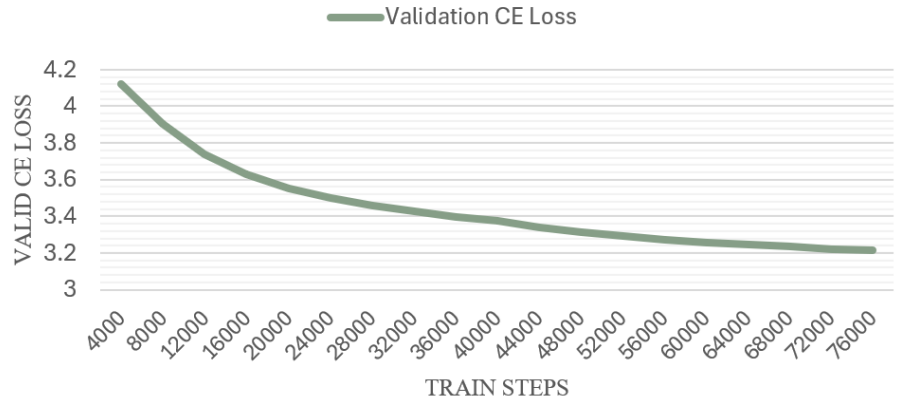

Figure 9: Validation loss convergence of FineGates + FFT method in Section 5.7

a maximum sequence length of 256. For a fair comparison, we train FineGates on the C4 dataset using the same sequence length.

M GLUE Benchmark statistics

Presented in Table 10.

Table 10: The GLUE benchmark datasets statistics

Dataset	MNLI	QQP	QNLI	SST2	COLA	STSB	MRPC	RTE
Samples	392,702	363,846	104,743	67,349	8,551	5,749	3,668	2,490

N Hyperparameters for training

We present the hyperparameters for RoBERTa models in Table 11, for comparisons with the APT model in Table 12, and for LLaMA experiments in Table 13. Each dataset includes its own validation set, which is used for evaluation. In target-task fine-tuning experiments, we train all models with the Adam optimizer and decoupled weight decay regularization (Loshchilov and Hutter, 2017), optimizing $\mathbf{\Omega}$ with fixed learning rates across tasks. For sparsified base-model pretraining and post-training pruning, we restrict the training budget to 2B tokens and apply learning-rate warmup followed by a cosine scheduler.

Table 11: Hyperparameters for Table 1.

Hyperparameter / Dataset	CoLA	STS-B	MRPC	RTE	SST-2	MNLI	QNLI	QQP
RoBERTa-Base								
lr bias + head	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4
lr gates	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3
lr schedule	constant	constant	constant	constant	constant	constant	constant	constant
epochs	100	100	100	100	50	50	50	50
lambda	50	10	50	50	0.1	0.01	0.01	0.01
max seq length	512	512	512	512	512	512	512	512
RoBERTa-Large								
lr bias + head	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4
lr gates	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3
lr schedule	constant	constant	constant	constant	constant	constant	constant	constant
epochs	50	50	50	50	100	100	100	100
lambda	10	10	10	10	0.1	0.1	0.1	0.1
max seq length	512	512	512	512	512	512	512	512

Table 12: Hyperparameters for Table 2.

Hyperparameter / Dataset	CoLA	STS-B	MRPC	RTE	SST-2
RoBERTa-Base					
lr bias + head	1e-4	1e-4	1e-4	1e-4	1e-4
lr gates	1e-3	1e-3	1e-3	1e-3	1e-3
lr schedule	constant	constant	constant	constant	constant
epochs	100	100	100	100	50
lambda	100	100	100	100	100
max seq length	512	512	512	512	512

O Empirical Loss Convergence Plots

To support our theoretical claim regarding model convergence, we present the loss curves of the pre-training LLaMA-1B base model with FineGates in Figures 8 and 9.

P Hardware used for experiments.

We conducted training and evaluation of the compressed LLaMA-3.2-1B backbone model on a single NVIDIA RTX 6000 GPU paired with an AMD EPYC 9334 32-core processor. While training was performed on the GPU, evaluation wall-clock time measurements were obtained on the CPU. For all other experiments, we used a single NVIDIA H200 GPU with an Intel® Xeon® Platinum 8568Y+ CPU for both training and evaluation.

Q Limitations

While our method demonstrates strong performance and efficiency gains, it is important to acknowledge certain limitations. Our experiments are restricted to the GLUE benchmark. This reflects the practical constraints faced by smaller academic labs, which often lack access to large-scale computing resources. Although we could not evaluate larger models such as LLaMA-2 13B or GPT-style architectures, we contend that results from smaller-scale models remain valuable. They provide rigorous, reproducible insights and can inform innovations that may later scale to larger systems. Moreover, GLUE offers a standardized benchmark for comparison with prior work, making it a meaningful testbed for early-stage progress, and is still used by many recent works, i.e. Zhang et al. (2025),Lei et al. (2024),Wang et al. (2024),Refael et al. (2024b),Bałazy et al. (2024),Ceritli et al. (2024), Miles et al. (2024).

R Baselines Description

We compare our method against full fine-tuning—where the model is initialized with pre-trained weights and all parameters are updated during training—as well as several recently proposed efficient fine-tuning methods:

- LoRA (Hu et al., 2021) with rank= 4 applied additionally to the weight matrices $\widetilde{W}_{\text{mlp}}$ as in our method,

Table 13: Hyperparameters for Table 3.

Hyperparameter/ Dataset	CoLA	STS-B	MRPC	RTE	SST-2
			LLAMA-3.2-1B		
lr bias + head	1e-4	1e-4	1e-4	1e-4	1e-4
lr gates	1e-3	1e-3	1e-3	1e-3	1e-3
lr schedule	constant	constant	constant	constant	constant
epochs	100	100	100	100	50
lambda	0.01	0.01	0.01	0.01	0.01
max seq length	512	512	512	512	512

which results in total 0.7M trainable parameters for Roberta-Base model and 1.8M for Roberta-Large base model.

- BitFit (Zaken et al., 2021) - an efficient finetuning method where only biases are trained for each adapted layer in the base model.
- VeRA (Kopiczko et al., 2023) - vector-based random matrix adaptation where the number of trainable parameters is significantly lower than in LoRA without pruning or compression of the base model.
- LoRA-XS (Bałazy et al., 2024) further reduces the number of trainable parameters by learning a small matrix positioned between two frozen low-rank matrices.
- $\text{RoCoFT}_{\text{1-Row}}$ (Kowsher et al., 2024) - the method proposed to directly train a single row or column in the base model without adding adapter weights.
- APT (Zhao et al., 2024a) - a recently proposed finetuning method that allows compression of the base model while training adapter weights. Our approach differs from APT in two main points: (1) we learn the gates for the base model weights jointly with the finetuning task objective, and (2) our model obtains comparable results without training additional low-rank matrices $\mathbf{W}_A, \mathbf{W}_B$ resulting in optimization of less trainable parameters.
- VeLoRA (Miles et al., 2024) an adapter-based method similarly to LoRA which is trained with low gradients projections.
- MaskLLM (Fang et al., 2024) is a method that prunes the LLM models in a semi-structured (or "N:M") way, aimed at reducing computational overhead during inference. Instead of developing a new importance criterion, MaskLLM explicitly models N:M patterns as a learnable distribution through Gumbel Softmax sampling.

S GPU-based Inference Time

We train and evaluate our model with a Llama-3.2-1B backbone on an NVIDIA A100 (80GB). We present results in Table 14, measured after each epoch (in total 100 epochs) and averaged over five epochs. The base time measurement is obtained by taking the average inference time per epoch across the first 5 epochs (with 0 parameters removed). All measurements are repeated 10 times.

Table 14: Training progression showing evaluation time reduction, removed parameters, and achieved sparsity across epochs.

Epochs	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
Eval Epoch Time Reduction (%)	1.5	4.9	8.1	9.4	14.1	15.1	16.1	17.3	18.5	19.8	21.3	22.9	24.6	25.9	27.0	27.9	28.5	28.9	29.3
Removed Parameters (Millions)	45	109	181	223	244	259	271	284	299	318	341	367	391	410	427	441	454	464	473
Sparsity (%)	3.62	8.86	14.75	18.15	19.81	21.04	22.07	23.13	24.34	25.86	27.76	29.84	31.81	33.33	34.70	35.88	36.92	37.73	38.46