GradES: Significantly Faster Training in Transformers with Gradient-Based Early Stopping

Qifu Wen[1,†,*], **Xi Zeng**[1,†], **Zihan Zhou**[1], **Shuaijun Liu**[2], **Mehdi Hosseinzadeh**[3,4], **and Reza Rawassizadeh**[1,5]

[1]Department of Computer Science, Boston University Metropolitan College
[2]Information Hub, The Hong Kong University of Science and Technology, Guangzhou
[3]School of Engineering and Technology, Duy Tan University, Da Nang, Vietnam
[4]Jadara Research Center, Jadara University, Irbid, Jordan
[5]Center of Excellence in Precision Medicine and Digital Health, Department of Physiology, Chulalongkorn University, Thailand
[†]These authors contributed equally to this work
[*]Corresponding author: qfwen@bu.edu

ABSTRACT

Early stopping monitors global validation loss and halts all parameter updates simultaneously, which is computationally costly for large transformers due to the extended time required for validation inference. We propose *GradES*, a novel gradient-based early stopping approach that operates within transformer components (attention projections and Feed-Forward layer matrices). We found that different components converge at varying rates during fine-tuning. *GradES* tracks the magnitude of gradients in backpropagation for these matrices during training. When a projection matrix's gradients fall below a convergence threshold τ, we exclude that projection matrix from further updates individually, eliminating costly validation passes while allowing slow converging matrices to continue learning. By strategically freezing parameters when their gradients converge, *GradES* speeds up training time by 1.57–7.22$\times$ while simultaneously enhancing generalization through early prevention of overfitting, resulting in 1.2% higher average accuracy.

Keywords Transformers, Early Stopping, Fine-tuning, Optimization, Large Language Models

1 Introduction

Large language models (LLMs) have remarkable capabilities across diverse tasks, but their training and deployment require substantial computational costs that scale with model size and inference frequency. Due to the high cost of training models with billions of parameters, any effort toward reducing the training cost improves the development of LLMs.

Fine-tuning LLMs requires balancing computational efficiency against downstream task performance. As transformer architectures scale to billions of parameters, the computation becomes increasingly expensive [1, 2]. While optimizing for memory and computational efficiency remains important [3], one key issue is often overlooked, i.e., the common practice of extensive fine-tuning assumes that more gradient updates always improve performance [4]. Research has shown that training for excessive epochs leads to overfitting, where models memorize training data patterns that fail to generalize to test data. [5]

Conventional early stopping, which is determined based on loss score, is computationally expensive for large language models, as each validation step requires full forward passes through all transformer layers for every sample in the validation set. This overhead scales linearly with both model size and validation set size, forcing practitioners to validate infrequently, typically every few thousand training steps [6], thereby creating a fundamental trade-off between computational cost and the risk of overfitting. Later we present the overhead of early stopping on training.

Furthermore, our experiment shows that Transformer's components have different convergence patterns, as shown in Figure 1. and the binary decision of classic early stopping fails to exploit the diverse convergence patterns, where attention and MLP components exhibit fundamentally different learning dynamics during the fine-tuning process [7].

Through analysis of gradient dynamics across transformer architectures, we identified a critical inefficiency in current fine-tuning practices, i.e., varied convergence across the Transformer's components. By tracking gradient magnitudes for attention projections matrix: $\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$ that compute queries, keys, and values respectively, and $\mathbf{W}_o$ or output projection. As well as MLP network matrix $\mathbf{W}_{\text{up}}$ for dimension expansion and $\mathbf{W}_{\text{down}}$ for dimension reduction. We observe that some components reach convergence with gradient norms below 10^{-3}, while others maintain substantial gradients throughout, as shown in Figure 1.

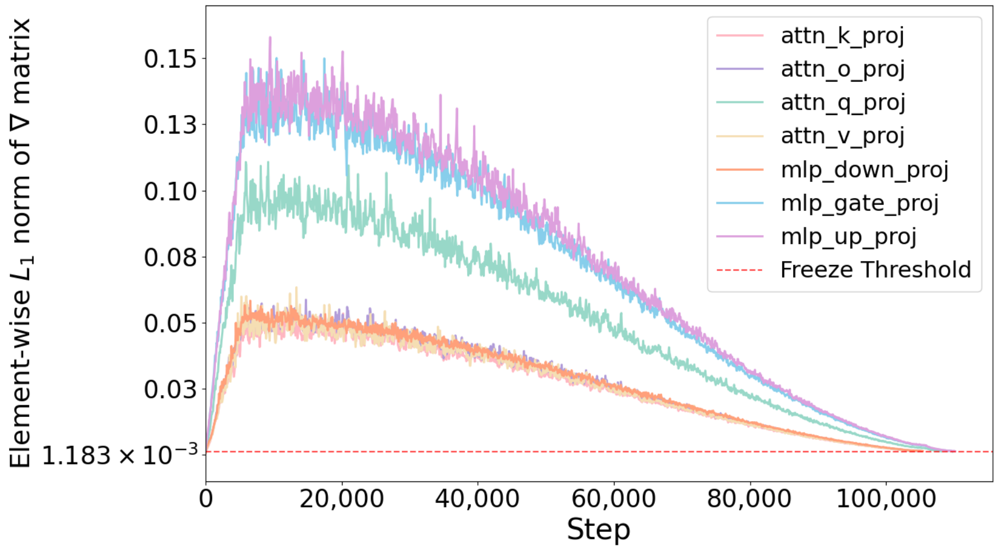

Figure 1: Element-wise L_1 norms for the gradient matrix of transformer components during fine-tuning with LoRA on Qwen3-0.6B [8] over 100,000 steps. Where each step consists of processing one training batch through the complete forward pass, loss computation, backpropagation, and parameter update cycle. The seven tracked matrices comprise attention projections ($\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$, $\mathbf{W}_o$) and MLP components ($\mathbf{W}_{\text{gate}}$, $\mathbf{W}_{\text{up}}$, $\mathbf{W}_{\text{down}}$). MLP projections exhibit 2 to $3\times$ higher gradient magnitudes than attention projections throughout training, with $\mathbf{W}_{\text{up}}$ and $\mathbf{W}_{\text{down}}$ maintaining the largest gradients. The red dotted line indicates our convergence threshold $\tau = 1.183 \times 10^{-3}$.

This disparity motivates *GradES*, our gradient-based early stopping strategy that operates at the matrix level. Rather than requiring expensive validation passes, *GradES* leverages gradient information already computed during backpropagation to monitor each matrix $\mathbf{W}^{(l)} \in \{\mathbf{W}_q^{(l)}, \mathbf{W}_k^{(l)}, \mathbf{W}_v^{(l)}, \mathbf{W}_o^{(l)}, \mathbf{W}_{\text{gate}}^{(l)}, \mathbf{W}_{\text{up}}^{(l)}, \mathbf{W}_{\text{down}}^{(l)}\}$ independently. When a matrix's gradient magnitude falls below threshold τ, we freeze its parameters while maintaining gradient flow for proper backpropagation, transforming early stopping from a binary termination decision into a continuous regularization mechanism.

Our experiments across five LLMs, varied from 0.6B to 14B parameters, demonstrate that *GradES* reduces fine-tuning time by 50% while maintaining or improving accuracy across eight benchmarks. Attention projections consistently stabilize 2–3 times faster than MLP components, with key and value projections freezing earliest—a pattern that validates our component-specific (Attention or MLP) approach over traditional monolithic early stopping. *GradES* can be seamlessly integrated with optimizers such as Adam [9], SGD [10], and parameter-efficient fine-tuning methods such as LoRA [11], while complementing weight decay by preventing overfitting in converged components as weight decay continues, regularizing active components.

Contributions. We make the following contributions:

- We propose *GradES*, a gradient-based early stopping method designed specifically for transformer architectures, eliminating expensive validation inference used for classic early stopping.
- We identified that gradient-based early stopping serves as an effective regularization technique, preventing overfitting in fast-converging Transformers' components while maintaining learning capacity in not yet converged completely, achieving improved accuracy up to 1.2% and training time speed up of 1.57–7.22×.
- We validate the compatibility of *GradES* with diverse optimization algorithms (e.g., Adam, SGD) and parameter-efficient fine-tuning methods (e.g., LoRA), showing consistent acceleration across training paradigms while preserving their respective computational and memory efficiency advantages.

- We release our implementation as an open-source repository and PyTorch package. It requires minimal modifications to existing training pipelines.

2 Related Work

2.1 Parameter-Efficient Fine-tuning on Transformer Architecture

The Transformer architecture [12] consists of distinct weight matrices for queries, keys, values, output projections, and MLP components. Parameter-efficient fine-tuning (PEFT) methods that task adaptation requires only low-dimensional updates to pre-trained weights. Houlsby et al. proposed adapters [13] that insert trainable bottleneck layers between transformer blocks (3.6% additional parameters), prefix tuning [14] optimizes continuous task-specific vectors as virtual tokens (0.1% parameters). LoRA [11] employs low-rank decomposition $\mathbf{BA}$ where $r \ll \min(d_{\text{in}}, d_{\text{out}})$ (10,000$\times$ parameter reduction).

Despite their architectural differences, all these methods update all parameters uniformly throughout training. In contrast, *GradES* employs a customized approach (customized based on gradient norm threshold) to fine-tuning.

2.2 Adaptive Training and Convergence

A group of promising theoretical works established connections between gradient convergence patterns and optimization dynamics in neural networks. Arora et al. [15] analyzes gradient descent convergence in deep linear networks, while Chatterjee et al. [16] extends these results to general deep architectures, demonstrating that different components exhibit distinct convergence trajectories. Nguegnang et al. [17] characterizes convergence rates for gradient descent in linear networks, providing theoretical foundations for component convergence monitoring. The role of gradient-based stopping criteria has been formalized by Patel et al. [18], who establish strong convergence guarantees when gradients fall below thresholds. Additionally, Gess et al. [19] demonstrate exponential convergence rates for momentum methods in over-parameterized settings, suggesting that convergence patterns are robust across different optimizers.

Building on these theoretical insights, several methods exploit convergence patterns for training efficiency. AutoFreeze [20] accelerates BERT fine-tuning by 2.55$\times$ through automatic layer freezing based on gradient patterns, though it operates only at layer granularity. AFLoRA [21] introduces gradual freezing of LoRA adapters using convergence scores, achieving 9.5$\times$ parameter reduction and 1.86$\times$ speedup. LoRAF [22] prevents catastrophic forgetting by selectively freezing LoRA matrices based on their convergence state, using 95% fewer parameters than standard LoRA. However, these approaches either require coarse layer-level decisions (AutoFreeze), additional architectural components like routers or masks (AFLoRA, LoRAF), or coupling with specific fine-tuning methods.

GradES advances this line of work by directly leveraging the heterogeneous convergence phenomenon [23] at the granularity of individual weight matrices. *GradES* uses gradient magnitudes to detect when components converge [24]. It then freezes individual components as they reach convergence, adapting to each component's learning rate without requiring schedules.

2.3 Early Stopping

Early stopping represents an established regularization technique with extensive theoretical and practical foundations. Prechelt et al. [25] provided practical guidelines for validation-based stopping criteria, while theoretical analyses by Yao et al. [26] demonstrated that early stopping acts as implicit regularization in gradient descent learning. Recent theoretical advances by Ji et al. [27] have proven the consistency of early stopping in neural networks, establishing rigorous convergence guarantees. Beyond traditional validation-based approaches, gradient-based early stopping by Mahsereci et al. [28] and Pflug et al. [29] monitors gradient magnitudes to detect convergence without requiring validation data. Recent work has explored more sophisticated stopping strategies: instance dependent early stopping by Yuan et al. [30] adapts termination per training example to reduce overfitting on easy samples, while correlation based methods by Ferro et al. [31] combine multiple online indicators for more robust stopping decisions. Methods for noisy settings have also emerged, including strategies for learning with label noise by Bai et al. [32] and stopping without validation data by Yuan et al. [33].

All these approaches apply stopping decisions globally, either terminating all training simultaneously or operating at the instance level. *GradES* differs fundamentally by applying gradient-based convergence detection at the individual weight matrix level within transformers, allowing different components to stop at their natural convergence points while others continue learning. This approach will effectively bridge early stopping with fine-grained regularization.

3 Method

3.1 GradES Algorithm

We analyzed the gradient magnitudes of weight matrices across Transformer layers during fine-tuning and discovered different convergence rates for different components within Transformers' Layers. For each weight matrix $\mathbf{W}^{(l)}$ in layer $l \in \{1, \dots, L\}$, we track the element-wise L_1 norm of gradients at training step t, as it is presented in Equation 1.

$$G_{\mathbf{W}}^{(l)}(t) = \|\nabla \mathbf{W}^{(l)}\|_1 = \sum_{i=1}^{m} \sum_{j=1}^{n} |(\nabla \mathbf{W}^{(l)})_{ij}| \tag{1}$$

where $\mathbf{W}^{(l)} \in \{\mathbf{W}_q^{(l)}, \mathbf{W}_k^{(l)}, \mathbf{W}_v^{(l)}, \mathbf{W}_o^{(l)}, \mathbf{W}_{\text{gate}}^{(l)}, \mathbf{W}_{\text{up}}^{(l)}, \mathbf{W}_{\text{down}}^{(l)}\}$ represents the attention projection matrices ($\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$, $\mathbf{W}_o$) and MLP weight matrices ($\mathbf{W}_{\text{gate}}$, $\mathbf{W}_{\text{up}}$, $\mathbf{W}_{\text{down}}$) within each Transformer layer. The metric $G_{\mathbf{W}}^{(l)}(t)$ quantifies the total gradient flow through each component, revealing distinct convergence trajectories that motivate our gradient-based component level freezing strategy.

Algorithm 1 *GradES*: Adaptive Parameter Freezing based on Gradient Magnitude

```
Input: Pre-trained model $\mathcal{M}$ with $L$ layers, dataset $\mathcal{D}$, total steps $T$, grace period ratio $\alpha$, threshold $\tau$, learning rate $\eta$
Output: Fine-tuned model $\mathcal{M}'$
 1: Initialize: All parameters trainable, frozen set $\mathcal{F} \leftarrow \emptyset$
 2: $t_{\text{grace period}} \leftarrow \lceil \alpha \cdot T \rceil$                ▷ Start monitoring after grace period
 3: The grace period is computed as a fraction $\alpha$ of the total training steps $T$.
 4: for training step $t = 1$ to $T$ do
 5:     Sample $\mathcal{B} \sim \mathcal{D}$; compute loss $\mathcal{L}(\mathcal{B})$ and gradients
 6:     if $t > t_{\text{grace period}}$ then                                   ▷ Monitor after grace period
 7:         for each $\mathbf{W} \in$ all layers : $\mathbf{W} \notin \mathcal{F}$ do
 8:             $G_{\mathbf{W}}(t) = \sum_{i,j} |(\nabla \mathbf{W})_{ij}|$
 9:             if $G_{\mathbf{W}}(t) < \tau$ then $\mathcal{F} \leftarrow \mathcal{F} \cup \{\mathbf{W}\}$
10:     for each projection matrix $\mathbf{W}$ do
11:         if $\mathbf{W} \notin \mathcal{F}$ then                              ▷ Update only active parameters
12:             $\mathbf{W} \leftarrow \mathbf{W} - \eta \cdot \nabla \mathbf{W}$
13:         else
14:             Skip update (but gradient still flows through)
15:     if all parameters frozen then
16:         break                                                    ▷ Early stopping
17: $\mathcal{M}' \leftarrow$ Update model with modified projection matrices $\mathbf{W}$
18: return Fine-tuned model $\mathcal{M}'$
```

Algorithm 1 formalizes, *GardES*, our gradient-based early stopping procedure. *GardES* algorithm introduces three key innovations that distinguish it from traditional early stopping approaches:

Component-level convergence detection. Unlike conventional methods that monitor global validation accuracy, *GradES* tracks individual weight matrices $\mathbf{W}^{(l)} \in \{\mathbf{W}_q^{(l)}, \mathbf{W}_k^{(l)}, \mathbf{W}_v^{(l)}, \mathbf{W}_o^{(l)}, \mathbf{W}_{\text{gate}}^{(l)}, \mathbf{W}_{\text{up}}^{(l)}, \mathbf{W}_{\text{down}}^{(l)}\}$ within each layer l. We employ the L_1 gradient norm $G_{\mathbf{W}}(t) = \|\nabla \mathbf{W}\|_1$ as our convergence metric, chosen for its computational efficiency compared to L_2 norms. When $G_{\mathbf{W}}(t) < \tau$, we consider the component converged and freeze its parameters (lines 8-14).

Adaptive grace period strategy. The initial $t_{\text{grace period}} = \lceil \alpha T \rceil$ steps (with $\alpha = 0.5$ in our experiments) allow all components to escape their pre-trained initialization before convergence monitoring begins. This prevents terminating training of components prematurely that may appear initially converged but require substantial adaptation for the downstream task. The grace period duration depends on the total training examples and scales proportionally with the total training budget T.

Gradient flow preservation. A critical design choice is maintaining gradient computation through frozen parameters (line 12). While frozen components do not receive parameter updates (lines 17-22), they continue to propagate gradients to earlier layers. This ensures that active components receive proper gradient signals throughout training, preventing the gradient flow disruption that would occur with complete component removal, like pruning. The computational overhead of gradient computation through frozen components is negligible compared to the savings from skipping parameter updates.

The algorithm terminates when all components satisfy the convergence criterion (Line 24), eliminating unnecessary computation on converged parameters. We provide formal convergence guarantees and theoretical analysis in Appendix B.

3.2 *GradES* for Low-Rank Adaptation

LoRA is one of the most common approaches used for fine-tuning. Since LoRA constrains parameters to a low-dimensional subspace, gradient dynamics in this reduced space exhibit fundamentally different convergence properties than full fine-tuning. When applying *GradES* to LoRA [11] fine-tuning, we monitor gradient magnitudes in the low-rank space rather than the full parameter space. Let $l \in \{1, \dots, L\}$ denote the layer index where L is the total number of layers. Within each transformer layer l, we apply LoRA decomposition to individual weight matrices $\mathbf{W} \in \{\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v, \mathbf{W}_o, \mathbf{W}_{\text{gate}}, \mathbf{W}_{\text{up}}, \mathbf{W}_{\text{down}}\}$, where the first four correspond to attention projections and the latter three to MLP components.

For each weight matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, where $d_{\text{out}}, d_{\text{in}}$ are the output and input dimensions, in layer l, the LoRA weight is:

$$\mathbf{W}^{(l)}_{\text{adapted}} = \mathbf{W}^{(l)}_{\text{frozen}} + \mathbf{B}^{(l)}_{\mathbf{W}} \mathbf{A}^{(l)}_{\mathbf{W}} \tag{2}$$

where $\mathbf{B}^{(l)}_{\mathbf{W}} \in \mathbb{R}^{d_{\text{out}} \times r}$ and $\mathbf{A}^{(l)}_{\mathbf{W}} \in \mathbb{R}^{r \times d_{\text{in}}}$ are the trainable low-rank matrices with rank $r \ll \min(d_{\text{out}}, d_{\text{in}})$.

For each individual LoRA matrix $\mathbf{W}$ in layer l, we track convergence by monitoring the combined gradient magnitude:

$$G^{(l)}_{\mathbf{W}}(t) = \|\nabla \mathbf{A}^{(l)}_{\mathbf{W}}\|_1 + \|\nabla \mathbf{B}^{(l)}_{\mathbf{W}}\|_1 \tag{3}$$

where $\nabla \mathbf{A}^{(l)}_{\mathbf{W}}$ and $\nabla \mathbf{B}^{(l)}_{\mathbf{W}}$ denote the gradients of the low-rank matrices at training step t, and $\| \cdot \|_1$ denotes the element-wise L_1 norm.

The freezing operates at the matrix level, enabling precise control. After the grace period period $t > t_{\text{grace period}}$, we independently freeze each LoRA matrix when:

$$G^{(l)}_{\mathbf{W}}(t) < \tau_r \quad \text{for } \mathbf{W} \in \{\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v, \mathbf{W}_o, \mathbf{W}_{\text{gate}}, \mathbf{W}_{\text{up}}, \mathbf{W}_{\text{down}}\} \tag{4}$$

where τ_r is the convergence threshold adjusted for the reduced parameter count. Once a specific matrix reaches convergence, we stop updating its corresponding $\mathbf{A}^{(l)}_{\mathbf{W}}$ and $\mathbf{B}^{(l)}_{\mathbf{W}}$ while continuing to compute gradients through them for backpropagation. Training terminates when all LoRA matrices across all layers are frozen.

4 Experimental Setup

We implemented our experiments using Python 3.10.6 and PyTorch 2.0.1[1]. All experiments were conducted on a system equipped with two NVIDIA RTX 4090 GPUs (24 GB VRAM each), 256 GB RAM, and an Intel Core i9 processor (3.30 GHz) running Ubuntu 20.04 LTS with CUDA 12.0. Unless otherwise specified, all models were trained using mixed precision with automatic mixed precision (AMP) enabled for computational efficiency.

4.1 Datasets and Models

We evaluate *GradES* on the five most popular language models on huggingface [34] (at the time of conducting this experiment) to represent different parameter scales, quantization strategies, and architectural designs. Our model selection spans three orders of magnitude in parameter count (0.6B to 14B) to investigate the scalability of our approach. Specifically, we employ: (1) **Qwen3-14B** [8], a state-of-the-art multilingual model representing large-scale architectures; (2) **Microsoft Phi4-14B** [35], featuring optimized attention mechanisms for efficient inference; (3) **Llama-3.1-8B-Instruct** [36] with 4-bit quantization, demonstrating compatibility with quantization-aware training; (4) **Mistral-7B-Instruct-v0.3** [37] with 4-bit quantization, utilizing grouped-query attention for reduced memory footprint; and (5) **Qwen3-0.6B** [8], a compact model validating our method's effectiveness in configuration with limited resources. We conducted all experiments using 4-bit quantized models, due to hardware limitations of the experimental platform.

[1] `https://pytorch.org/get-started/pytorch-2.0`

4.2 Benchmark and Evaluation Metrics

We evaluate *GradES* on eight widely used benchmarks covering different aspects of language understanding. For reasoning tasks, we use BoolQ [38] for boolean question answering requiring complex reasoning, PIQA [39] for physical commonsense reasoning about everyday situations, SIQA [40] for social commonsense reasoning about human interactions, and HellaSwag [41] for commonsense inference about plausible continuations. For knowledge-intensive task completion, we employ OpenBookQA [42] for science questions requiring multi-hop reasoning, ARC-Easy and ARC-Challenge [43] for grade school science questions at two difficulty levels, and WinoGrande [44] for pronoun resolution requiring commonsense knowledge. We measure both task accuracy and computational efficiency to provide a rounded view of *GradES*'s benefits. Average task accuracy is measured by accuracy on each benchmark's test set. For computational efficiency metrics, we track training time on identical hardware and floating point operations (FLOPs) computed using PyTorch profiler [45].

4.3 PEFT and Early Stopping Methods

We evaluate *GradES* against established fine-tuning paradigms to demonstrate its benefits. Our baseline methods comprise Full Parameter Fine-tuning (FP), which updates all model parameters without constraints; and LoRA [11] to assess the composability of our method. In particular, we apply *GradES* to both full fine-tuning (FP+GradES) and LoRA (LoRA+GradES), yielding six distinct configurations for comprehensive evaluation. For validation-based early stopping baselines (FP+ES and LoRA+ES), we perform validation checks at 5% intervals throughout training. Terminating training when validation loss fails to improve for consecutive checkpoints. We presents the overhead of early stopping compared to *GradES* in 2. More detailed experiment settings and hyperparameter selection are provided in Appendix D.

5 Results

5.1 Accuracy on Benchmarks

We evaluate *GradES* against standard full-parameter (FP) fine-tuning and LoRA across five language models ranging from 0.6B to 14B parameters on eight commonsense reasoning benchmarks. As shown in Table 1, *GradES* consistently improves upon the baseline early stopping (ES) method across both fine-tuning methods (FP and LoRA). For full-parameter fine-tuning on larger models (14B), full-parameter with *GradES* achieves the highest average accuracy on Qwen3 (90.81%) and Phi4 (91.94%), demonstrating consistent improvements over full-parameter fine-tuning (90.80% and 91.93% respectively). Its impact is more significant on smaller models, where LoRA (ES) and LoRA (GradES) on Qwen3 0.6B achieve 67.37% and 67.30% average accuracy, substantially outperforming standard full-parameter methods (∼66.5%). Notably, the choice of base fine-tuning method (Full-parameter vs. LoRA) exhibits its model-independent behavior, while full-parameter fine-tuning methods perform competitively on larger models (Qwen3 14B, Phi4 14B), LoRA variants showed higher accuracy on mid-sized models, with LoRA (ES) achieving 86.27% on Mistral-7B compared to 75.80% for standard full-parameter fine-tuning, a remarkable 10.47 percentage point improvement. *GradES* shows particular strength on specific benchmarks, achieving best results on Winograde compared to other methods (Qwen3 14B: 84.77%), PIQA (Phi4 14B: 92.60%).

These results present that *GradES* provides higher accuracy compared to existing fine-tuning strategies for both full parameter fine-tuning and PEFT methods, such as LoRA.

Table 1: Comparison of the accuracy for different fine-tuning methods on five different language models. Values are reported in percentages, and the best one in each category is highlighted in bold.

Model	Method	BoolQ	PIQA	SIQA	HellaSwag	Winograde	OpenBookQA	ARC-C	ARC-E	Avg.
Qwen3 14B	Full Parameter	91.07	91.29	81.93	95.11	83.03	91.60	94.31	98.07	90.80
	FP+ES	91.07	91.08	81.99	95.05	83.03	91.40	93.98	98.07	90.71
	FP+GradES	**91.22**	91.19	82.04	94.97	83.03	**91.80**	**94.31**	97.89	**90.81**
	LoRA	90.86	**91.24**	81.68	95.21	83.35	91.40	94.31	97.19	90.65
	LoRA+ES	90.64	91.08	81.99	**95.40**	81.53	91.40	93.65	97.54	90.40
	LoRA+GradES	90.67	91.13	**82.29**	95.22	**84.77**	91.20	93.31	97.02	90.70
Phi4 14B	Full Parameter	90.49	91.95	83.27	95.49	88.71	93.00	94.31	98.25	91.93
	FP+ES	90.34	92.11	82.55	95.50	88.56	93.00	94.31	98.07	91.80
	FP+GradES	90.31	**92.60**	**83.06**	95.43	**88.95**	92.60	94.31	98.25	**91.94**
	LoRA	90.31	92.00	82.45	95.36	87.53	91.80	**94.65**	98.07	91.52
	LoRA+ES	90.61	92.22	82.60	**95.44**	87.37	92.00	94.31	**98.25**	91.60
	LoRA+GradES	**90.49**	92.60	82.40	95.38	87.37	91.60	94.65	98.07	91.57
Qwen3 0.6B	Full Parameter	77.28	69.31	66.99	65.09	50.28	61.20	61.54	80.53	66.53
	FP+ES	77.06	68.99	67.09	65.16	49.57	61.00	63.21	81.23	66.66
	FP+GradES	77.03	**71.38**	66.84	64.26	**51.62**	62.40	61.20	79.65	66.80
	LoRA	**79.14**	69.15	**69.14**	67.94	49.09	**66.20**	60.54	77.19	67.30
	LoRA+ES	79.11	69.10	68.83	**68.18**	49.64	65.00	61.20	77.89	**67.37**
	LoRA+GradES	78.56	69.42	68.17	68.20	49.41	65.40	**61.54**	77.72	67.30
Llama-3.1-8B	Full Parameter	89.27	88.08	81.01	94.40	81.93	85.00	83.61	92.46	86.97
	FP+ES	89.24	87.81	80.86	94.42	82.56	85.80	82.61	92.11	86.93
	FP+GradES	88.87	**88.25**	80.45	94.23	82.79	84.80	**83.95**	**92.81**	**87.02**
	LoRA	87.98	87.65	79.73	94.31	80.35	**87.20**	83.28	91.40	86.49
	LoRA+ES	88.62	88.19	79.32	94.20	80.35	87.20	83.28	91.75	86.62
	LoRA+GradES	**88.78**	88.03	**79.68**	**94.43**	**81.69**	87.00	83.95	90.53	86.76
Mistral-7B	Full Parameter	85.26	80.25	77.33	83.71	66.30	75.60	62.54	75.44	75.80
	FP (ES)	85.32	80.09	76.20	83.95	57.54	74.60	64.21	74.39	74.54
	FP+GradES	85.38	78.56	75.79	84.14	66.47	76.40	61.20	75.10	75.38
	LoRA	**89.33**	87.43	79.79	**94.95**	79.72	84.20	76.25	88.42	85.01
	LoRA+ES	88.93	**88.30**	**81.01**	94.69	**82.24**	**85.00**	79.60	90.35	**86.27**
	LoRA+GradES	89.36	88.03	80.71	94.69	80.90	84.40	**81.61**	**89.65**	86.17

5.2 Training Efficiency

Table 2 presents a comparison of computational efficiency across different fine-tuning strategies. *GradES* demonstrates substantial efficiency improvements over both baseline methods and early stopping approaches. Full parameter fine-tuning along with *GradES* (FP+GradES) achieves consistent training speedups of 1.32–1.64$\times$ while reducing FLOPs by 29%–45% (0.55–0.71$\times$) across all model scales. This efficiency gain is particularly striking when compared to standard ES, which paradoxically slows down training by 0.60–0.72$\times$ despite reducing computational operations, likely due to frequent validation overhead and convergence monitoring costs. The efficiency benefits extend to parameter-efficient fine-tuning, where LoRA with *GradES* achieves speedups of 1.10–1.15$\times$ over standard LoRA while reducing FLOPs by 6%–13%. When compared to the full parameter baseline, LoRA and GradES delivers remarkable speedups of 2.66–2.87$\times$ for large models, with the Qwen3 0.6B model achieving an exceptional 7.22$\times$ speedup. Remarkably, these computational savings do not compromise model quality, as shown in Table 1, *GradES* variants achieve the highest average accuracies on larger models (Qwen3 14B: 90.81%, Phi4: 91.94%) while requiring 45% fewer FLOPs than standard fine-tuning.

The stark contrast between *GradES* and classic ES is noteworthy, while LoRA+ES slows training by 0.66–0.83$\times$ without FLOPs reduction on most models, LoRA and *GradES* delivers both efficiency and accuracy gains simultaneously.

These results show our gradient-guided early stopping approach as a practical solution for fine-tuning with limited resources. The method's ability to reduce computational requirements by nearly half while maintaining or improving task accuracy mitigates challenges in deploying large language models. The consistent efficiency gains across model architectures (Qwen, Phi, Llama, Mistral) and scales (0.6B–14B parameters) suggest that *GradES* provides a generalizable approach to reduce fine-tuning time. We also record component convergence statistics, including the percentage of components that have converged at each time step, to understand the dynamics of our approach. This is shown in 2. In the graph, models starts to converge after the grace period (α). The larger models (7B-14B) converge rapidly with most components frozen by step 1400, while Qwen-0.6B exhibits delayed convergence, suggesting scale- dependent optimization dynamics.

Table 2: Training time and computational cost comparison of different fine-tuning methods on 5 different language models. Training time in seconds, FLOPs in floating point operations. Speedup and FLOPs ratios are computed relative to Full Parameter(Base) for all methods. The best one in each category is highlighted in bold.

Model	Method	Training Time (s)	Speedup	FLOPs	FLOPs Ratio
Qwen3 14B	Full Parameter(Base)	16,202	1.00×	1.17×10^{18}	1.00×
	FP+ES	22,466	0.72×	8.76×10^{17}	0.75×
	FP+GradES	10,721	1.51×	6.43×10^{17}	**0.55**×
	LoRA	6,387	2.54×	2.74×10^{18}	2.34×
	LoRA+ES	23,932	0.68×	2.74×10^{18}	2.34×
	LoRA+GradES	5,643	**2.87**×	2.43×10^{18}	2.08×
Phi4 14B	Full Parameter(Base)	14,627	1.00×	1.12×10^{18}	1.00×
	FP+ES	23,040	0.63×	9.49×10^{17}	0.85×
	FP+GradES	9,218	1.59×	6.15×10^{17}	**0.55**×
	LoRA	6,030	2.43×	2.69×10^{18}	2.40×
	LoRA+ES	24,394	0.60×	2.69×10^{18}	2.40×
	LoRA+GradES	5,506	**2.66**×	2.38×10^{18}	2.13×
Qwen3 0.6B	Full Parameter(Base)	6,550	1.00×	3.68×10^{16}	1.00×
	FP+ES	8,569	0.76×	2.76×10^{16}	0.75×
	FP+GradES	4,018	1.63×	2.03×10^{16}	**0.55**×
	LoRA	892	**7.34**×	8.94×10^{16}	2.43×
	LoRA+ES	6,155	1.06×	8.94×10^{16}	2.43×
	LoRA+GradES	907	7.22×	8.43×10^{16}	2.29×
Llama-3.1-8B	Full Parameter(Base)	9,541	1.00×	7.45×10^{17}	1.00×
	FP+ES	16,129	0.59×	6.70×10^{17}	0.90×
	FP+GradES	5,832	1.64×	4.10×10^{17}	**0.55**×
	LoRA	3,737	2.55×	1.58×10^{18}	2.12×
	LoRA+ES	15,499	0.62×	1.58×10^{18}	2.12×
	LoRA+GradES	3,370	**2.83**×	1.40×10^{18}	1.88×
Mistral-7B	Full Parameter(Base)	9,256	1.00×	6.38×10^{17}	1.00×
	FP+ES	14,521	0.64×	6.38×10^{17}	1.00×
	FP+GradES	6,996	1.32×	4.51×10^{17}	**0.71**×
	LoRA	3,752	2.47×	1.51×10^{18}	2.37×
	LoRA+ES	11,159	0.83×	1.51×10^{18}	2.37×
	LoRA+GradES	3,259	**2.84**×	1.32×10^{18}	2.07×

6 Discussion

6.1 Convergence Patterns Across Different Models

Figure 2 shows the progression of the matrix that converged across different model scales during training. After a warm-up period of $\alpha = 1000$ steps, our method begins freezing converged components based on gradient magnitude thresholds τ (model-specific values detailed in Appendix C).

The difference in convergence rate reflects distinct architectural behaviors. Larger models (7B-14B) exhibit rapid convergence, with the majority of components frozen by step 1400—approximately 40% through training. In contrast, the smaller Qwen-0.6B model demonstrates delayed convergence, with no components meeting the freezing criteria until step 1600.

Notably, the threshold τ varies significantly between training methods, and full fine-tuning requires larger thresholds compared to LoRA adaptation.

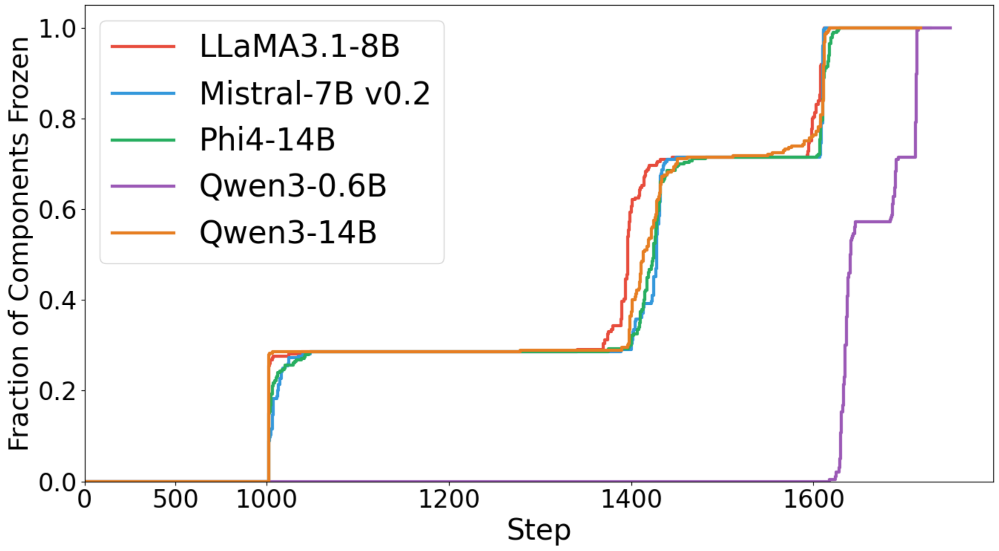

Figure 2: Cumulative frozen components during training across model scales. Fraction of weight matrices frozen over time for five different LLMs.

6.2 Attention versus MLP components

Figure 3 presents the gradient norms $|\nabla W|$ across all weight matrices during fine-tuning of the Qwen-0.6B model. We compute the element-wise L1 norm for each weight matrix and report averaged values for two architectural components: MLP matrices ($\mathbf{W}_{\text{up}}$, $\mathbf{W}_{\text{down}}$; orange) and attention projection matrices ($\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$, $\mathbf{W}_o$; blue).

We have two key observations. First, the gradient trend reflects the cosine learning rate schedule: initial warmup drives increasing gradient magnitudes as the model adapts to the task distribution, while the subsequent cosine decay produces monotonically decreasing gradient norms, enabling smooth convergence from exploration to refinement.

Second, and more critically for our method, MLP weight matrices consistently maintain larger gradient norms compared to attention projection matrices throughout training. This persistent gap indicates that MLP parameters require more steps to converge, suggesting inefficiency under uniform training strategies. This observation directly motivates our approach: by dynamically allocating computational resources proportional to gradient magnitudes, we can accelerate convergence of slower learning components while maintaining overall model accuracy.

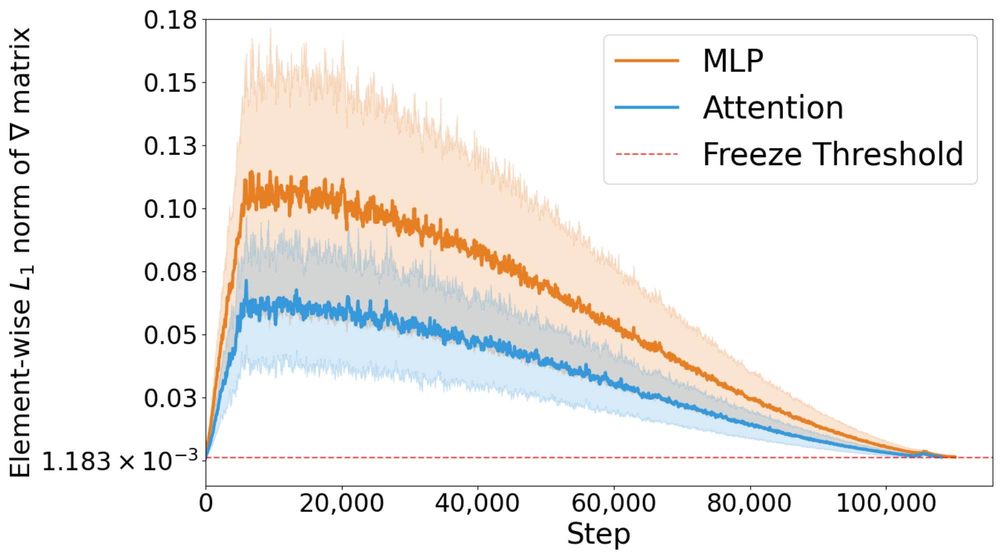

Figure 3: Gradient norm evolution during Qwen-0.6B fine-tuning. Element-wise L1 norms of weight gradients averaged across layers for MLP matrices (orange) and attention projections (blue). MLP matrices consistently exhibit larger gradient magnitudes throughout training, indicating slower convergence and motivating targeted computational allocation.

6.3 Combination with Parameter-Efficient Fine-Tuning Methods

GradES combines particularly well with parameter-efficient fine-tuning methods such as LoRA. As shown in Table 2, LoRA+GradES achieves the fastest training times across all models, reducing training time to just 0.14–0.38× of standard full-parameter fine-tuning. This dramatic speedup comes from optimizing two independent dimensions, (i) LoRA reduces the parameter space through low-rank weight decomposition, (ii) while *GradES* reduces training iterations by detecting convergence through gradient monitoring. For example, on Qwen3 0.6B, LoRA+GradES completes training in just 907 seconds compared to 6,550 seconds for standard fine-tuning, a 7.22× speedup, while achieving better accuracy (67.30% vs. 66.53%).

The benefits of combining *GradES* with LoRA become especially clear when compared to standard early stopping. While LoRA+ES actually increases training time by 2.97–6.90× due to validation overhead, LoRA+GradES maintains or improves LoRA's efficiency. This difference suggests that gradient-based early stopping criteria work particularly well in LoRA's constrained optimization landscape, where the reduced parameter space provides clearer convergence signals. Although LoRA methods inherently require more FLOPs per iteration than full-parameter training (2.07–2.43×), *GradES* compensates by reducing the total number of iterations needed. The result is a practical for deployment due to the extremely fast training times with reasonable computational costs, making LoRA+GradES the optimal choice for fine-tuning.

6.4 GradES versus Classic Early Stopping

A fundamental limitation of classic early stopping is the computational expense of requiring complete forward passes for every validation step. In contrast, *GradES* reuses gradient information from backpropagation, yielding substantial computational savings. As shown in Table 2, it achieves up to 6.79× speedup compared to classic early stopping on Qwen3-0.6B (907s vs 6,155s for LoRA fine-tuning), while maintaining comparable accuracy. The minimal accuracy difference—67.30% for *GradES* versus 67.37% for classic early stopping as shown in Table 1—does not justify the significant computational overhead. Across all five models tested, *GradES* consistently reduces training time by 35–66% while achieving a 45–71% reduction in FLOPs compared to baseline full fine-tuning, making it particularly valuable for limited resource settings.

Furthermore, classic early stopping employs a model-wide convergence criterion that is not suitable for Transformer architectures. As demonstrated in Figure 1, different weight matrices within transformer layers exhibit varying convergence rates. Model-wide early stopping fails to account for this heterogeneity, potentially leading to overfitting in some parameters while underfitting in others. *GradES* addresses this limitation by enabling component-specific convergence criteria, allowing MLP and attention components to be trained independently until each reaches its optimal stopping point. This ensures all weight matrices converge according to appropriate criteria rather than being halted or unnecessarily extended based on global validation metrics.

7 Limitation

While *GradES* demonstrates substantial efficiency gains, some limitations exists. First, gradient monitoring incurs approximately 3% computational overhead, though this is negligible compared to the 1.57–7.22× training time speed up achieved. Second, the convergence threshold τ requires manual tuning across different models and tasks, with full-parameter fine-tuning requiring larger thresholds than LoRA and model scale affecting optimal values. Third, our current implementation employs static freezing without patience mechanisms common in traditional early stopping, potentially leading to premature convergence decisions. Additionally, while we validate *GradES* on transformer architectures, its applicability to other neural architectures, such as convolutional networks, graph neural networks, and emerging architectures like state space models, remains unexplored.

8 Conclusion and Future work

Several promising directions emerge for future work. Automatic threshold selection through gradient statistics or meta learning could eliminate manual tuning, while incorporating patience parameters would allow components to temporarily violate convergence criteria before freezing. Dynamic freezing and unfreezing mechanisms could adapt to task complexity and distribution shifts, particularly beneficial when combined with specific thresholds for MLP versus attention components as suggested by Figure 3. Integration with complementary efficiency techniques like mixed precision training, gradient checkpointing, and structured pruning could yield multiplicative speedups. Extending *GradES* beyond transformers to vision models, graph networks, and hybrid architectures would broaden its impact. Most ambitiously, applying gradient stopping to pretraining could significantly reduce the massive computational costs of foundation model development, while theoretical analysis of convergence criteria would provide principled guidelines for threshold selection and accuracy guarantees.

We presented *GradES*, a gradient-based early stopping strategy that addresses the computational inefficiencies of traditional validation-based approaches for fine-tuning transformers. By monitoring gradient magnitudes at the component level and selectively halting updates for converged components, *GradES* achieves up to 7.22× speed up in training time and 45% reduction in FLOPs while maintaining or improving model accuracy across eight commonsense reasoning benchmarks. Our experiments across five model architectures (0.6B–14B parameters) demonstrate that *GradES* seamlessly integrates with both full parameter fine-tuning and parameter-efficient methods like LoRA, with the combination achieving remarkable efficiency, completing fine-tuning in as little as 14% of the baseline time.

The key insight underlying our approach is that different transformer components exhibit distinct convergence behaviors during fine-tuning. By recognizing and exploiting these diverse convergence patterns, *GradES* allocates computational resources more efficiently than uniform training strategies. The method's ability to eliminate costly validation passes while providing precise convergence control represents a practical advancement for deploying large language models with limited resources. As the scale of language models continues to grow, gradient optimization strategies like *GradES* will become increasingly critical for making these powerful models accessible to the broader research community and enabling rapid experimentation in real-world applications.

References

[1] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. *arXiv preprint arXiv:2305.14314*, 2023.

[2] Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D. Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[3] Reza Rawassizadeh. *Machine Learning and Artificial Intelligence: Concepts, Algorithms and Models*. 1 edition, March 2025.

[4] Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q. Weinberger, and Yoav Artzi. Revisiting few-sample BERT fine-tuning. *arXiv preprint arXiv:2006.05987*, 2021.

[5] Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. On the stability of fine-tuning BERT: Misconceptions, explanations, and strong baselines. *arXiv preprint arXiv:2006.04884*, 2021.

[6] Kushal Tirumala, Aram H. Markosyan, Luke Zettlemoyer, and Armen Aghajanyan. Memorization without overfitting: Analyzing the training dynamics of large language models. *arXiv preprint arXiv:2205.10770*, 2022.

[7] Xinhao Yao, Hongjin Qian, Xiaolin Hu, Gengze Xu, Wei Liu, Jian Luan, Bin Wang, and Yong Liu. Theoretical insights into fine-tuning attention mechanism: Generalization and optimization. *arXiv preprint arXiv:2410.02247*, 2025.

[8] Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, and Others. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2017.

[10] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[11] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[13] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. *arXiv preprint arXiv:1902.00751*, 2019.

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

[15] Sanjeev Arora, Nadav Cohen, Noah Golowich, and Wei Hu. A convergence analysis of gradient descent for deep linear neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

[16] Sourav Chatterjee. Convergence of gradient descent for deep neural networks. *arXiv preprint arXiv:2203.16462*, 2022.

[17] Gabin Maxime Nguegnang, Holger Rauhut, and Ulrich Terstiege. Convergence of gradient descent for learning linear neural networks. *arXiv preprint arXiv:2108.02040*, 2021.

[18] Vivak Patel. Stopping criteria for, and strong convergence of, stochastic gradient descent on bottou-curtis-nocedal functions. *arXiv preprint arXiv:2004.00475*, 2021.

[19] Benjamin Gess and Sebastian Kassing. Exponential convergence rates for momentum stochastic gradient descent in the overparametrized setting. *arXiv preprint arXiv:2302.03550*, 2024.

[20] Yuhan Liu, Saurabh Agarwal, and Shivaram Venkataraman. Autofreeze: Automatically freezing model blocks to accelerate fine-tuning. *arXiv preprint arXiv:2102.01386*, 2021.

[21] Zeyu Liu, Souvik Kundu, Anni Li, Junrui Wan, Lianghao Jiang, and Peter Anthony Beerel. AFLoRA: Adaptive freezing of low rank adaptation in parameter efficient fine-tuning of large models. *arXiv preprint arXiv:2403.13269*, 2024.

[22] Juzheng Zhang, Jiacheng You, Ashwinee Panda, and Tom Goldstein. LoRA without forgetting: Freezing and sparse masking for low-rank adaptation. In *Sparsity in LLMs (SLLM): Deep Dive into Mixture of Experts, Quantization, Hardware, and Inference*, 2025.

[23] Dylan J. Foster, Ayush Sekhari, and Karthik Sridharan. Uniform convergence of gradients for non-convex learning and optimization. *arXiv preprint arXiv:1810.11059*, 2018.

[24] Wenzhi Gao, Ya-Chi Chu, Yinyu Ye, and Madeleine Udell. Gradient methods with online scaling. *arXiv preprint arXiv:2411.01803*, 2024.

[25] Lutz Prechelt. Early stopping — but when? In Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*. Springer, 2012.

[26] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

[27] Ziwei Ji, Justin D. Li, and Matus Telgarsky. Early-stopped neural networks are consistent. In *Advances in Neural Information Processing Systems*, 2021.

[28] Maren Mahsereci, Lukas Balles, Christoph Lassner, and Philipp Hennig. Early stopping without a validation set. *arXiv preprint arXiv:1703.09580*, 2017.

[29] Georg Ch Pflug. Non-asymptotic confidence bounds for stochastic approximation algorithms with constant step size. *Monatshefte für Mathematik*, 110(3-4):297–314, 1990.

[30] Suqin Yuan, Runqi Lin, Lei Feng, Bo Han, and Tongliang Liu. Instance-dependent early stopping. *arXiv preprint arXiv:2502.07547*, 2025.

[31] Manuel Vilares Ferro, Yerai Doval Mosquera, Francisco J. Ribadas Pena, and Victor M. Darriba Bilbao. Early stopping by correlating online indicators in neural networks. *arXiv preprint arXiv:2402.02513*, 2024.

[32] Yingbin Bai, Erkun Yang, Bo Han, Yanhua Yang, Jiatong Li, Yinian Mao, Gang Niu, and Tongliang Liu. Understanding and improving early stopping for learning with noisy labels. *arXiv preprint arXiv:2106.15853*, 2021.

[33] Suqin Yuan, Lei Feng, and Tongliang Liu. Early stopping against label noise without validation data. *arXiv preprint arXiv:2502.07551*, 2025.

[34] HuggingFace. Hugging face hub, 2020. AI model and dataset repository platform.

[35] Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J. Hewett, et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

[36] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[37] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[38] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

[39] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7432–7439, 2020.

[40] Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. SocialIQA: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.

[41] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[42] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

[43] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

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

[45] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, et al. PyTorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

A Theoretical Analysis: Selection of Norm

The choice of norm for gradient monitoring significantly impacts both computational efficiency and convergence detection reliability. For a gradient matrix $G \in \mathbb{R}^{m\times n}$, we consider four matrix norm candidates: element-wise L_1 ($\|G\|_{1,1} = \sum_{i,j} |g_{ij}|$), Frobenius ($\|G\|_F = \sqrt{\sum_{i,j} g_{ij}^2}$), spectral ($\|G\|_2 = \sigma_{\max}(G)$), and subordinate L$_\infty$ ($\|G\|_\infty = \max_i \sum_j |g_{ij}|$).

We select the element-wise L_1 norm based on computational efficiency and convergence properties. The L_1 norm requires only $O(mn)$ operations through element-wise summation, avoiding expensive computations such as square roots (Frobenius) or singular value decomposition (spectral, $O(mn \min(m,n))$). Furthermore, the L_1 norm provides a stronger convergence criterion through the following theorem:

Theorem 1. *For any matrix $A \in \mathbb{R}^{m\times n}$, the elementwise L_1 norm provides an upper bound for commonly used matrix norms:*

$$\|A\|_2 \leq \|A\|_{1,1} \tag{5}$$
$$\|A\|_F \leq \|A\|_{1,1} \tag{6}$$
$$\|A\|_\infty \leq \|A\|_{1,1} \tag{7}$$
$$\|A\|_1 \leq \|A\|_{1,1} \tag{8}$$

where $\|A\|_1 = \max_j \sum_i |a_{ij}|$ denotes the subordinate L_1 norm (maximum column sum).

Proof. For (5), the spectral norm satisfies $\|A\|_2 \leq \sqrt{\|A\|_1 \cdot \|A\|_\infty}$ by the well-known inequality for induced norms. Since $\|A\|_1 = \max_j \sum_i |a_{ij}| \leq \sum_{i,j} |a_{ij}| = \|A\|_{1,1}$ and similarly $\|A\|_\infty \leq \|A\|_{1,1}$, we have:

$$\|A\|_2 \leq \sqrt{\|A\|_1 \cdot \|A\|_\infty} \leq \sqrt{\|A\|_{1,1} \cdot \|A\|_{1,1}} = \|A\|_{1,1}$$

For (6), note that $a_{ij}^2 \leq |a_{ij}| \cdot \max_{k,l} |a_{kl}| \leq |a_{ij}| \cdot \|A\|_{1,1}$ for any element. Summing over all indices:

$$\|A\|_F^2 = \sum_{i,j} a_{ij}^2 \leq \sum_{i,j} |a_{ij}| \cdot \|A\|_{1,1} = \|A\|_{1,1}^2$$

Therefore, $\|A\|_F \leq \|A\|_{1,1}$.

For (7), the subordinate L$_\infty$ norm is the maximum row sum: $\|A\|_\infty = \max_{1\leq i\leq m} \sum_{j=1}^n |a_{ij}|$. Since the maximum row sum cannot exceed the sum of all elements: $\|A\|_\infty \leq \sum_{i=1}^m \sum_{j=1}^n |a_{ij}| = \|A\|_{1,1}$.

For (8), similarly, the subordinate L_1 norm is the maximum column sum: $\|A\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^m |a_{ij}| \leq \sum_{i=1}^m \sum_{j=1}^n |a_{ij}| = \|A\|_{1,1}$. □

These relationships establish that monitoring the L_1 norm provides a universal upper bound for convergence detection. When $\|G\|_{1,1} < \tau$, we guarantee that all other standard matrix norms are also bounded by τ, ensuring robust convergence detection across multiple norm perspectives while maintaining linear computational complexity.

B Convergence Properties

We provide theoretical guarantees for the convergence of Algorithm 1. Our analysis demonstrates that *GradES* converges to a stationary point of the loss function while ensuring computational efficiency through adaptive parameter freezing.

Theorem 2 (Convergence of GradES)**.** *Consider Algorithm 1 applied to a loss function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ that is L-smooth and lower bounded by $\mathcal{L}^*$. With threshold $\tau > 0$, the algorithm satisfies:*

1. *The loss sequence $\{\mathcal{L}(t)\}_{t=1}^T$ is non-increasing after warm up*
2. *All frozen parameters satisfy $\|\nabla_{\mathbf{W}}\mathcal{L}\|_{1,1} < \tau$ at convergence*
3. *The algorithm terminates in finite time with $\min_{t\in[T]} \|\nabla\mathcal{L}(t)\| \leq \tau$*

Proof. **Part 1: Monotonic Loss Decrease.** After the warmup period ($t > 0.05T$), the cosine schedule ensures η_t is decreasing. For any active parameter $\mathbf{W} \notin \mathcal{F}$ at step t, the update rule yields:

$$\mathcal{L}(\mathbf{W}_{t+1}) \leq \mathcal{L}(\mathbf{W}_t) + \langle \nabla \mathcal{L}(\mathbf{W}_t), \mathbf{W}_{t+1} - \mathbf{W}_t \rangle + \frac{L}{2} \|\mathbf{W}_{t+1} - \mathbf{W}_t\|^2 \tag{9}$$

$$= \mathcal{L}(\mathbf{W}_t) - \eta_t \|\nabla \mathcal{L}(\mathbf{W}_t)\|^2 + \frac{L\eta_t^2}{2} \|\nabla \mathcal{L}(\mathbf{W}_t)\|^2 \tag{10}$$

For frozen parameters $\mathbf{W} \in \mathcal{F}$, we have $\mathbf{W}_{t+1} = \mathbf{W}_t$, thus $\mathcal{L}(\mathbf{W}_{t+1}) = \mathcal{L}(\mathbf{W}_t)$. The cosine schedule with maximum value $\eta_0 < \frac{2}{L}$ ensures:

$$\mathcal{L}(t+1) \leq \mathcal{L}(t) - \sum_{\mathbf{W} \notin \mathcal{F}} \eta_t \left(1 - \frac{L\eta_t}{2}\right) \|\nabla_{\mathbf{W}} \mathcal{L}(t)\|^2 \tag{11}$$

Since $\eta_t \leq \eta_0 < \frac{2}{L}$ throughout training, the loss sequence is non-increasing.

Part 2: Frozen Parameters at Stationary Points. A parameter matrix $\mathbf{W}$ is frozen at step t_f when $\|\nabla_{\mathbf{W}} \mathcal{L}(t_f)\|_{1,1} < \tau$. Since frozen parameters receive no further updates:

$$\mathbf{W}_t = \mathbf{W}_{t_f} \quad \forall t > t_f \tag{12}$$

As shown in Part 1, the gradient magnitudes are non-increasing after warmup. Combined with the continuity of gradients:

$$\|\nabla_{\mathbf{W}} \mathcal{L}(t)\|_{1,1} \leq \|\nabla_{\mathbf{W}} \mathcal{L}(t_f)\|_{1,1} < \tau \quad \forall t > t_f > 0.05T \tag{13}$$

This ensures that once a parameter is frozen, its gradient remains below the threshold.

Part 3: Finite-Time Termination. Define the active parameter set at time t as $\mathcal{A}_t = \{\mathbf{W} : \mathbf{W} \notin \mathcal{F}_t\}$. The cardinality $|\mathcal{A}_t|$ is non-increasing since parameters can only transition from active to frozen.

Under the cosine schedule, as $t \to T$, we have $\eta_t \to 0$. In experiment, gradient magnitudes decrease monotonically after warmup. Therefore, there exists a finite $T^* < T$ such that either:

- All parameters satisfy $\|\nabla_{\mathbf{W}} \mathcal{L}\|_{1,1} < \tau$ and are frozen, or
- The cosine schedule drives $\eta_t \|\nabla \mathcal{L}(t)\| < \epsilon$ for arbitrarily small ϵ

In both cases, the algorithm effectively converges with $\min_{t \in [T]} \|\nabla \mathcal{L}(t)\| \leq \tau$. $\square$

Corollary 3. *GradES achieves an ϵ-stationary point while potentially reducing computational cost by a factor proportional to $|\mathcal{F}|/d$, where d is the total number of parameters. The cosine schedule ensures smooth convergence without oscillations in gradient magnitudes.*

This analysis establishes that *GradES* maintains convergence guarantees under the practical cosine learning rate schedule used in our experiments. The threshold τ controls the trade-off between convergence accuracy and computational savings, while the 5% warmup period ensures stable gradient behavior before monitoring begins.

C Hyperparameter Configuration

We provide comprehensive hyperparameter settings to ensure reproducibility across all experimental conditions. Tables 3–4 detail the configuration for five language models (Qwen3-14B, Phi4-14B, Qwen3-0.6B, Llama-3.1-8B, and Mistral-7B) under both full parameter (FP) fine-tuning and LoRA adaptation. We also applied early stopping with a validation threshold of 0.0005, patience of 3 epochs, and validation interval of 5% for all models and methods.

Table 3: Basic hyperparameters for fine-tuning methods across 5 language models. These parameters are shared across base, ES, and *GradES* variants of each method.

Model	Method	Learning Rate	Batch Size	Grad Accum	Max Seq Len	LoRA Rank
Qwen3 14B	FP	2e-5	1	4	4096	-
	LoRA	2e-4	16	4	4096	32
Phi4 14B	FP	2e-5	1	4	4096	-
	LoRA	2e-4	16	4	4096	32
Qwen3 0.6B	FP	2e-5	1	4	4096	-
	LoRA	2e-4	16	4	4096	32
Llama-3.1-8B	FP	2e-5	1	4	4096	-
	LoRA	2e-4	16	4	4096	32
Mistral-7B	FP	2e-5	1	4	4096	-
	LoRA	2e-4	16	4	4096	32

Table 4: GradES hyperparameters for all models and methods.

Model	Method	Grace Period Ratio(α)	Threshold Tau (τ)
Qwen3 14B	FP	0.55	6.387926
	LoRA	0.55	0.025181
Phi4 14B	FP	0.55	3.512882
	LoRA	0.55	0.025181
Qwen3 0.6B	FP	0.55	1.804456
	LoRA	0.55	0.001183
Llama-3.1-8B	FP	0.55	2.404167
	LoRA	0.55	0.021637
Mistral-7B	FP	0.55	2.726866
	LoRA	0.55	0.029591

D Code Availability

We are committed to ensuring the reproducibility of our research. To facilitate this, we provide comprehensive resources:

Implementation. Our complete implementation, including training scripts, evaluation pipelines, and gradient monitoring utilities, is publicly available at `https://github.com/IXZZZ9/GradES`. The repository includes detailed documentation, environment setup instructions, and scripts to reproduce all experimental results presented in this paper.

Licensing. All code is released under the MIT License, promoting open scientific collaboration and industrial adoption. Model weights follow the original Qwen license terms.