P-RGE: A Framework for On-Device LLM Fine-Tuning with Parallelized Randomized Gradient Estimation

Team Bytebots
Srikrishna, Srikanth, Kiran, Vishweshwar

Department of Engineering Science
Indian Institute of Technology Hyderabad

August 25, 2025

Abstract

The personalization of Large Language Models (LLMs) on edge devices represents a new frontier in AI, yet it is fundamentally constrained by the limitations of mobile hardware and software. Traditional backpropagation-based training is infeasible due to prohibitive memory costs, high computational demands, and the inference-only nature of most mobile ML runtimes. This paper introduces a comprehensive framework that enables efficient, privacy-preserving fine-tuning directly on resource-constrained devices. Our method synergistically combines **Parallelized Randomized Gradient Estimation (P-RGE)**, a zeroth-order optimization algorithm that approximates gradients using only forward passes, with **Low-Rank Adaptation with Frozen-A (LoRA-FA)**, a highly parameter-efficient fine-tuning strategy. This architecture allows the entire update logic to be embedded within the model's inference graph, enabling fine-tuning even within inference-only environments such as ExecuTorch. We present the mathematical underpinnings, algorithmic design, experimental validation, and deployment pathway, offering a robust foundation for the next generation of private, personalized, and efficient LLM applications on the edge.

1 Introduction

The advent of Large Language Models (LLMs) has redefined the capabilities of artificial intelligence [2]. The next evolutionary step is to move beyond general-purpose models towards deep personalization, adapting LLMs to individual user data and contexts. While cloud-based fine-tuning is mature, it raises significant privacy concerns and introduces network latency. On-device fine-tuning is the definitive solution, ensuring data remains local and secure. However, this paradigm confronts three fundamental barriers, which we term the "three walls" of on-device training:

1. **The Memory Wall:** Backpropagation requires caching intermediate activations, a process whose memory footprint can exceed 45 GB for a 7B model [1], far beyond the capacity of typical edge devices.

2. **The Compute Wall:** Gradient computation is a computationally intensive workload not well-suited for mobile accelerators (e.g., DSPs, NPUs) that are highly optimized for inference.

3. **The Infrastructure Wall:** Mobile ML frameworks like ExecuTorch [8] and TensorFlow Lite are inference-only engines; they discard the computation graph and lack the primitives to support backpropagation.

We address these limitations by abandoning backpropagation in favor of a zeroth-order (ZO) optimization strategy specifically designed for the constraints of inference-only runtimes.

2 Related Work

Our work builds upon two primary research areas: parameter-efficient fine-tuning and zeroth-order optimization.

Parameter-Efficient Fine-Tuning (PEFT). PEFT methods aim to reduce the cost of fine-tuning by updating only a small subset of a model's parameters. Techniques like adapter tuning [6] and prompt tuning [7] have shown promise. Low-Rank Adaptation (LoRA) [3] has become particularly popular, as it injects trainable low-rank matrices into the model without introducing inference latency. Our work utilizes LoRA-FA, a variant that further reduces trainable parameters by freezing one of the low-rank matrices [5].

Zeroth-Order (ZO) Optimization. ZO methods are gradient-free and thus do not require backpropagation. They are well-suited for black-box optimization problems. Recent work like MeZO [4] successfully applied a ZO method to fine-tune LLMs by introducing a memory-efficient "random seed trick." However, MeZO's sequential nature makes it computationally slow. Our P-RGE algorithm directly addresses this performance bottleneck through parallelization.

3 The P-RGE Framework

Our framework is built on three core principles: a gradient-free optimizer, a highly efficient adaptation method, and a parallelization strategy to unify them.

3.1 Zeroth-Order Optimization via RGE

Let $\theta \in \mathbb{R}^d$ denote the trainable model parameters and $\mathcal{L}(\theta; \mathcal{B})$ be the loss on a mini-batch $\mathcal{B}$. The **Randomized Gradient Estimator (RGE)** approximates the true gradient $\nabla\mathcal{L}(\theta)$ using a finite-difference method along random directions:

$$\hat{\nabla}\mathcal{L}(\theta) = \frac{1}{q}\sum_{i=1}^{q} \frac{\mathcal{L}(\theta + \epsilon \mathbf{z}_i) - \mathcal{L}(\theta - \epsilon \mathbf{z}_i)}{2\epsilon}\mathbf{z}_i \tag{1}$$

where $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_d)$ are random direction vectors, $\epsilon > 0$ is the perturbation magnitude, and q is the *query budget*. The variance of this estimator scales as $O(d/q)$, making a larger q crucial for stable training.

3.2 Parallelized RGE (P-RGE)

A naive implementation of Equation 1 requires $2q$ sequential forward passes. P-RGE mitigates this cost via two levels of parallelization, as detailed in 1.

Algorithm 1 One Step of P-RGE Training

Input: Parameters θ, batch $\mathcal{B}$, budget q, step size η, scale ϵ
Generate q random seeds $\{s_1, \ldots, s_q\}$
▷ *Outer & Inner Loops: Batched Forward Pass*
Construct mega-batch by replicating $\mathcal{B}$ for $2q$ perturbations.
For each query $i \in \{1, \ldots, q\}$:
  Regenerate $\mathbf{z}_i$ from seed s_i.
  Compute $\mathcal{L}_i^+ = \mathcal{L}(\theta + \epsilon\mathbf{z}_i)$ and $\mathcal{L}_i^- = \mathcal{L}(\theta - \epsilon\mathbf{z}_i)$ in parallel.
▷ *Gradient Estimation & Update*
Initialize total update $\Delta\theta = \mathbf{0}$.
For each query $i \in \{1, \ldots, q\}$:
  Regenerate $\mathbf{z}_i$ from seed s_i.
  $g_i = (\mathcal{L}_i^+ - \mathcal{L}_i^-)/(2\epsilon)$.
  $\Delta\theta \leftarrow \Delta\theta + g_i \cdot \mathbf{z}_i$.
$\theta \leftarrow \theta - \eta \cdot (\Delta\theta/q)$.

Outer-Loop Parallelization. We construct a mega-batch of size $E = q \times B$ by replicating the input batch B for each of the q queries. This allows all queries to be processed in a single, large forward pass, fully exploiting hardware parallelism.

Inner-Loop Parallelization. For each direction $\mathbf{z}_i$, both $\mathcal{L}(\theta + \epsilon\mathbf{z}_i)$ and $\mathcal{L}(\theta - \epsilon\mathbf{z}_i)$ are evaluated simultaneously by further doubling the effective batch size. This minimizes redundant memory access to the frozen weights.

3.3 Low-Rank Adaptation with Frozen-A (LoRA-FA)

Applying ZO updates to all d parameters is infeasible. We adopt LoRA-FA, a PEFT method where a frozen weight matrix $\mathbf{W}_0 \in \mathbb{R}^{d_{in} \times d_{out}}$ is augmented with a low-rank update:

$$\mathbf{y} = \mathbf{x}\mathbf{W}_0 + s \cdot (\mathbf{x}\mathbf{A})\mathbf{B} \tag{2}$$

where $\mathbf{A} \in \mathbb{R}^{d_{in} \times r}$ and $\mathbf{B} \in \mathbb{R}^{r \times d_{out}}$. In LoRA-FA, $\mathbf{A}$ is randomly initialized and then frozen. Only the zero-initialized $\mathbf{B}$ matrix is trainable ($\theta = \text{vec}(\mathbf{B})$). This drastically reduces d, making the ZO update feasible.

4 System Architecture and Deployment

The system is designed to bridge Python-based simulation with real-world mobile deployment.

4.1 Repository Layout

The codebase is organized for clarity and modularity:

```
/
 main.py              # Streamlit UI
 train.py             # Training loop
 prge_optimizer.py    # Core P-RGE logic
 lora_fa_layer.py     # Custom LoRA-FA module
 export_to_mobile.py  # ExecuTorch export script
 ...                  # Utilities
```

4.2 Embedding Updates into the Inference Graph

The key to on-device training is to create a self-updating model. We design a 'DualForwardingLoRALayer' whose forward pass incorporates the update logic:

1. The layer's 'forward' method accepts the input activations, a scalar projected gradient estimate g_i, and a random seed s_i.
2. The seed is used to deterministically regenerate the noise vector $\mathbf{z}_i$.
3. The trainable matrix $\mathbf{B}$ is updated in-place using the logic from 1.
4. The model, when exported via 'torch.export', contains this self-updating logic within its static graph.

This design effectively "tricks" an inference engine into performing an optimization step with each forward pass.

5 Experimental Evaluation

We validate our framework by reproducing key results from Gao et al. [1] using the provided codebase.

Setup. We fine-tune TinyLlama-1.1B on the GLUE SST-2 sentiment classification task. The effective batch size is kept constant at $E = 16$. We compare single-query MeZO ($q = 1, B = 16$) with P-RGE ($q = 4, B = 4$).

Accuracy. As shown in Table 1, P-RGE consistently outperforms the single-query baseline. The multi-query approach provides a more stable and accurate gradient estimate, leading to better final model performance with the same computational budget per step.

Table 1: SST-2 Accuracy (%) on TinyLlama-1.1B

Method	Accuracy
MeZO ($q = 1, B = 16$)	87.5
P-RGE ($q = 4, B = 4$)	**89.1**

Performance. The parallelization strategies in P-RGE lead to significant speedups in wall-clock time over sequential implementations. The benefits are most pronounced with quantization, where memory-bound dequantization operations are performed once instead of twice per query, yielding a nearly 2x speedup.

6 Conclusion

We have presented a comprehensive framework that integrates P-RGE and LoRA-FA to make on-device fine-tuning of LLMs feasible under the severe constraints of mobile inference engines. By embedding the update logic directly into the inference graph, our approach achieves a unique combination of personalization, privacy, and efficiency. This work provides a validated and practical foundation for the next generation of edge AI applications that can adapt securely and intelligently to individual users.

References

[1] L. Gao, A. Ziashahabi, Y. Niu, S. Avestimehr, and M. Annavaram, "Enabling efficient on-device fine-tuning of LLMs using only inference engines," *arXiv preprint arXiv:2409.15520*, 2024.

[2] T. Brown et al., "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, 2020.

[3] E. J. Hu et al., "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022.

[4] S. Malladi et al., "MeZO: Fine-tuning language models with just forward passes," in *Advances in Neural Information Processing Systems*, 2023.

[5] Q. Zhang et al., "Adaptive budget allocation for parameter-efficient fine-tuning," in *International Conference on Learning Representations*, 2023.

[6] N. Houlsby et al., "Parameter-efficient transfer learning for NLP," in *International Conference on Machine Learning*, 2019.

[7] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," in *Empirical Methods in Natural Language Processing*, 2021.

[8] PyTorch Team, "ExecuTorch: A new runtime for on-device inference," *PyTorch Blog*, 2023.