

UC Santa Barbara

UC Santa Barbara Electronic Theses and Dissertations

Title

Doing More with Less: Efficiency Algorithms for Adaption, Inference, and Agentic Flow in Large Language Models

Permalink

<https://escholarship.org/uc/item/3064r35g>

ISBN

9798186381532

Author

YANG, YIFAN

Publication Date

2026-06-09

Peer reviewed|Thesis/dissertation

University of California
Santa Barbara

Doing More with Less: Efficiency Algorithms for Adaption, Inference, and Agentic Flow in Large Language Models

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Yifan Yang

Committee in charge:

Professor Zheng Zhang, Chair
Professor Haewon Jeong
Professor Yao Qin

June 2026

The Dissertation of Yifan Yang is approved.

Professor Haewon Jeong

Professor Yao Qin

Professor Zheng Zhang, Committee Chair

May 2026

Doing More with Less: Efficiency Algorithms for Adaption, Inference, and Agentic Flow
in Large Language Models

Copyright © 2026

by

Yifan Yang

Acknowledgements

I would first like to express my deepest gratitude to my advisor, Professor Zheng Zhang, for his guidance and unwavering support throughout my Ph.D. journey at UCSB. Your encouragement helped me through the most difficult period of my Ph.D., and your mentorship guided me onto the right path in research. I am truly grateful for your patience, insight, and continuous support.

I would also like to thank my committee members, Professor Qin Yao and Professor Haewon Jeon, for their valuable feedback, support, and encouragement throughout my Ph.D. proposal and defense. I am also deeply grateful to all of my mentors and collaborators Chang Liu, Alec Koppel, Jiajun Zhou, Ngai Wong, Zhuotong Chen, Zihu Wang, Kai Zhen, Ershad Banijamal, Denis Filimonov, Athanasios Mouchtaris, Sajjad Ghiasvand, Zhiyu Xue, Mahnoosh Alizadeh, Ramtin Pedarsani, Zhen Zhang, Nathan Susanj, Siegfried Kunzmann, Ziyue Liu, Yequan Zhao, Bhavana Ganesh, Markus Müller, Rupak Vignesh Swaminathan, Ryan Solgi, Jinming Lu, Jing Liu, whose advice, discussions, and contributions have helped shape my growth as a researcher. In addition, I would like to thank my mentors and managers during my internships: Kai Zhen, Markus Müller, Athanasios Mouchtaris, Pengkai Zhu, Jiawei Zhao, Ankan Bansal, and Shabnam Ghadar. Their guidance introduced me to industry research and product development, and helped me grow professionally.

I would like to give special thanks to my girlfriend, Jiayi Tian, who has been my best friend and constant source of support. Thank you for celebrating my achievements with me and for standing by me during difficult times. I am also deeply thankful to my parents and grandparents for their unconditional love and support. They have always been my strongest support, giving me the strength and courage to move forward.

Curriculum Vitæ

Yifan Yang

Education

2026	Ph.D. in Computer Science (Expected), University of California, Santa Barbara.
2025	M.S. in Computer Science, University of California, Santa Barbara.
2021	B.E. in Electronic and Information Engineering, Huazhong University of Science and Technology, China.

Publications and Articles

1. Liyan Tan, Yequan Zhao, Yifan Yang, Ruijie Zhang, Xinling Yu, Zheng Zhang. “GRZO: Group-Relative Zeroth-Order Optimization for Large Language Model Fine-Tuning.” *Arxiv* 2026.
2. Yifan Yang, Zhen Zhang, Jiayi Tian, Liyan Tan, Zheng Zhang. “IAPO: Input Attribution-Aware Policy Optimization for Tool Use in Small Multimodal Agents.” *Arxiv* 2026.
3. Jiayi Tian, Ryan Solgi, Jinming Lu, Yifan Yang, Hai Li, and Zheng Zhang.. “FLAT-LLM: Fine-grained Low-rank Activation Space Transformation for Large Language Model Compression.” *Findings of the European Chapter of the Association for Computational Linguistics (EACL)*, Rabat, Morocco, 2026.
4. Yifan Yang, Zhen Zhang, Rupak Vignesh Swaminathan, Jing Liu, Nathan Susanj, Zheng Zhang. “SharpZO: Hybrid Sharpness-Aware Vision Language Model Prompt Tuning via Forward-Only Passes.” *Proceedings of the 39th Annual Conference on Neural Information Processing Systems (NeurIPS)*, San Diego, USA, 2025.
5. Jiajun Zhou, Yifan Yang, Kai Zhen, Ziyue Liu, Yequan Zhao, Ershad Banijamali, Athanasios Mouchtaris, Ngai Wong, and Zheng Zhang. “QuZO: Quantized Zeroth-Order Fine-Tuning for Large Language Models.” *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Suzhou, China, 2025.
6. Zhen Zhang, Yifan Yang, Kai Zhen, Nathan Susanj, Athanasios Mouchtaris, Siegfried Kunzmann, and Zheng Zhang. “MaZO: Masked Zeroth-Order Optimization for Multi-Task Fine-Tuning of Large Language Models.” *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Suzhou, China, 2025.
7. Yifan Yang, Kai Zhen, Bhavana Ganesh, Aram Galstyan, Goeric Huybrechts, Markus Müller, Jonas M. Kübler, Rupak Vignesh Swaminathan, Athanasios Mouchtaris, Sravan Babu Bodapati, Nathan Susanj, Zheng Zhang, Jack FitzGerald,

- Abhishek Kumar. “Wanda++: Pruning Large Language Models via Regional Gradients.” *Findings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, Vienna, Austria, 2025.
8. Sajjad Ghiasvand, Yifan Yang, Zhiyu Xue, Mahnoosh Alizadeh, Zheng Zhang, Ramtin Pedarsani. “Communication-Efficient and Tensorized Federated Fine-Tuning of Large Language Models.” *Findings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, Vienna, Austria, 2025.
 9. Yifan Yang, Kai Zhen, Ershad Banijamali, Athanasios Mouchtaris, Zheng Zhang. “AdaZeta: Adaptive Zeroth-Order Tensor-Train Adaptation for Memory-Efficient Large Language Models Fine-Tuning.” *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Miami, USA, 2024.
 10. Zhuotong Chen, Zihu Wang, Yifan Yang, Qianxiao Li, Zheng Zhang. “PID Control-Based Self-Healing to Improve the Robustness of Large Language Models.” *Transactions on Machine Learning Research (TMLR)*, 2024.
 11. Yifan Yang, Jiajun Zhou, Ngai Wong, Zheng Zhang. “LoRETTA: Low-Rank Economic Tensor-Train Adaptation for Ultra-Low-Parameter Fine-Tuning of Large Language Models.” *Proceedings of the 2024 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, Mexico City, Mexico, 2024.
 12. Yifan Yang, Alec Koppel, Zheng Zhang. “A Gradient-based Approach for Online Robust Deep Neural Network Training with Noisy Labels.” *Arxiv* 2023.
 13. Yifan Yang, Chang Liu, Zheng Zhang. “Particle-based Online Bayesian Sampling.” *Arxiv* 2023.
 14. Yifan Yang, Lin Chen, Pan Zhou, Xiaofeng Ding. “Vflh: A Following-the-Leader-History Based Algorithm for Adaptive Online Convex Optimization with Stochastic Constraints.” *Proceedings of the 35th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, Atlanta, USA, 2023.
 15. Yifan Yang, Jie Xu, Zichuan Xu, Pan Zhou, Tie Qiu. “Quantile Context-Aware Social IoT Big Data Recommendation with D2D Communication.” *IEEE Internet of Things Journal*, 7(6): 5533–5548, 2020.

Abstract

Doing More with Less: Efficiency Algorithms for Adaption, Inference, and Agentic Flow
in Large Language Models

by

Yifan Yang

Large language models (LLMs) have demonstrated strong performance across a wide range of tasks, yet these gains come at the cost of substantial memory consumption and inference latency. This dissertation investigates methods for improving the training and inference efficiency of LLMs across three parts. The first part of this dissertation presents several novel methods for improving zeroth-order (ZO) optimization for LLM fine-tuning. To mitigate the high estimation variance inherent in ZO methods, we propose reducing the number of trainable parameters. Specifically, we introduce LoRETTA and AdaZeta, which achieve competitive performance compared to first-order and ZO fine-tuning methods while using an ultra-low parameter budget. In addition, we propose SharpZO, a method that significantly advances ZO fine-tuning for vision-language models by smoothing the loss landscape with a novel sharpness-aware strategy. The second part focuses on accelerating LLM inference through improved pruning techniques. We introduce the concept of regional gradients in our Wanda++ framework, which can be efficiently computed within each decoder block to construct a principled regional pruning score alongside a corresponding optimization procedure. The third part addresses the tool-calling capabilities of small agent models, where we propose IAPO, a teacher-guided training method that improves tool-use correctness. Throughout the dissertation, we provide theoretical analysis and extensive experimental results to validate our proposed methods.

Contents

Curriculum Vitae	v
1 Introduction	1
1.1 Existing Challenges in LLM Efficiency	1
1.2 Related Works	4
1.3 Overview of the Dissertation	10
2 Background	13
2.1 Background on Parameter Efficient Fine-tuning	13
2.2 Background on Zeroth-Order Optimization	14
2.3 Background on Reinforcement Learning	17
3 Efficient LLM Fine-tuning via Zeroth-Order Optimization	20
3.1 Introduction	20
3.2 Background on Tensor Decomposition	22
3.3 Parameter-Efficient Fine-Tuning through Tensor Decomposition	25
3.4 Zeroth-Order Tensor-Train Adaptation	31
3.5 Theoretical Analysis	34
3.6 Experimental Results	36
3.7 Conclusion	40
4 Sharpness-Aware Zeroth-Order Optimization	42
4.1 Introduction	42
4.2 SharpZO: A Hybrid Backpropagation-Free Optimization Framework	45
4.3 Experimental Results	52
4.4 Conclusion	55
5 Accelerating LLM Inference via Pruning	57
5.1 Introduction	57
5.2 Regional Gradient-Based Scoring	60
5.3 Regional Optimization	63
5.4 Experimental Results	65

5.5	Conclusion	72
6	Improving Tool Use in Small LLM through Reinforcement Learning	74
6.1	Introduction	74
6.2	Background on Visual Tool Use	76
6.3	Input-Attribution-Aware Policy Optimization	79
6.4	Experimental Results	83
6.5	Tool-calling Behavior	87
6.6	Conclusion	87
7	Conclusion	89
7.1	Summary of Contributions	89
7.2	Unifying Themes	93
7.3	Limitations and Future Directions	94
7.4	Concluding Remarks	96
	Bibliography	98

Chapter 1

Introduction

1.1 Existing Challenges in LLM Efficiency

Large language models (LLMs) have fundamentally transformed the landscape of natural language processing and artificial intelligence. Models such as BERT [1], the GPT family [2], and the LLaMA family [3] have demonstrated remarkable generalization capabilities across a wide range of tasks, from question answering and summarization to code generation and dialogue. The key to this success lies in training on vast corpora at scale, which instills rich linguistic and world knowledge into the model parameters. Yet this very strength comes with a heavy cost: as model capacity grows, so too does the computational burden of training, adapting, and serving these systems.

The challenge of scale manifests in several distinct but interrelated bottlenecks. During fine-tuning, the standard paradigm requires computing gradients through the entire model via backpropagation and maintaining optimizer states (e.g., first and second moments in Adam) alongside the model parameters. For a 7-billion parameter model, this alone can demand over 60 GB of GPU memory, far exceeding the capacity of a single consumer-grade GPU. The situation worsens at 70 billion parameters, where even

inference in FP16 format requires approximately 140 GB, necessitating multi-GPU deployments with the time to first token (TTFT) exceeding 100 milliseconds [4, 5]. These figures underscore that the memory and compute requirements of modern LLMs have outpaced what most research groups and practitioners can readily afford.

To address the training-side challenge, the field has converged on parameter-efficient fine-tuning (PEFT) as a practical compromise. PEFT methods such as Adapters [6], LoRA [7], prefix tuning [8], and prompt tuning [9] adapt LLMs by updating only a small fraction of the total parameters, substantially reducing optimizer memory. However, even with PEFT, the backpropagation graph must be maintained throughout the forward pass, keeping memory proportional to sequence length and model depth. Zeroth-order (ZO) optimization offers a more radical alternative: by estimating gradients from pairs of forward passes alone, it eliminates the backward graph entirely and can reduce training memory by over $8\times$ [10]. Yet ZO methods introduce their own difficulties, including a significant performance gap relative to first-order optimizers and a heightened risk of divergence in large-scale settings. Chapters 3 and 4 of this dissertation address these challenges.

On the inference side, even after training is complete, deploying LLMs at scale is expensive. Practical deployment requires either high-end hardware clusters or aggressive model compression. Among compression strategies, pruning—which directly removes individual weights or structured blocks of weights from a trained network—is attractive because it can reduce memory footprint and improve throughput without requiring retraining from scratch. However, existing post-training pruning methods such as Wanda [11] and SparseGPT [12] still fall short of the near-lossless compression ratios achieved by quantization methods [13, 14]. Incorporating gradient information into the pruning score improves accuracy but typically demands full-model backpropagation, reintroducing the very memory costs that pruning aims to eliminate. Chapter 5 of this dissertation proposes

a regional gradient strategy that resolves this tension.

A third and increasingly important dimension of efficiency concerns the deployment of LLMs as autonomous agents. The vision of an agent that reasons over long horizons, interacts with external tools, and responds to multimodal inputs has driven significant research interest. Reinforcement learning (RL) methods, particularly Proximal Policy Optimization (PPO) [15] and Group Relative Policy Optimization (GRPO) [16], have proven effective at training LLMs to perform complex reasoning and tool use. However, the inference cost of large-scale agent models creates a compelling case for small language model (SLM) agents: systems that are compact enough to run on consumer hardware or edge devices, yet capable enough to solve real tasks. This raises a new efficiency challenge: how to improve the decision-making quality of SLMs in multimodal tool-use settings, where standard RL reward signals are too sparse and rigid to guide learning effectively. Chapter 6 of this dissertation addresses this challenge through a novel attribution-aware RL formulation.

This dissertation investigates methods for improving the training and inference efficiency of LLMs along these three interconnected axes: (1) reducing training memory through zeroth-order optimization combined with tensorized PEFT adapters; (2) enhancing zeroth-order vision-language model fine-tuning through smoothness-aware training; (3) accelerating inference through lightweight regional pruning; and (4) enhancing the tool-use capability of small language model agents through attribution-aware reinforcement learning. Taken together, the contributions presented here advance the state of the art at each stage of the LLM lifecycle—adaptation, deployment, and agent interaction—offering practical paths toward making large language models accessible and efficient.

1.2 Related Works

1.2.1 Parameter-Efficient Fine-Tuning

Full fine-tuning of LLMs is prohibitively expensive in both memory and computation as model size grows. This observation has motivated a rich line of research in parameter-efficient fine-tuning (PEFT), which adapts large pre-trained models to downstream tasks by updating only a small subset of parameters. A key empirical finding underlies this research direction: pre-trained models possess a low intrinsic dimensionality, meaning that nearly full fine-tuning performance can be recovered by optimizing within a low-dimensional subspace [17, 18]. This finding suggests that millions of parameters are unnecessary for task adaptation, and motivates the search for principled compression of the trainable parameter space.

The earliest and most direct approach to PEFT is the adapter paradigm introduced by Houlsby et al. [6], which inserts small bottleneck modules after the attention and feed-forward layers of each Transformer block. During fine-tuning, only the adapter parameters are updated while the pre-trained weights remain frozen. Subsequent work extended this idea with AdapterFusion [19], which combines adapters for multiple tasks, and AdapterDrop [20], which removes adapters at inference time for speed. These methods achieve strong performance but still require inserting and storing non-trivial additional modules.

A complementary line of work, pioneered by LoRA [7], constrains the weight update ΔW to a low-rank factorization $\Delta W = AB$, where $A \in \mathbb{R}^{d \times r}$ and $B \in \mathbb{R}^{r \times k}$ with $r \ll \min(d, k)$. This reparameterization introduces no additional inference latency (since BA can be merged back into W after training) and reduces the number of trainable parameters dramatically. AdaLoRA [21] further improves on LoRA by adaptively allocating the rank budget across layers based on singular value importance, and DoRA [22] decomposes the weight update into magnitude and direction components for more stable

training. Despite these advances, LoRA-family methods still require millions of trainable parameters for large models: for LLaMA-2-70B, standard LoRA training involves over 16 million parameters.

A distinct family of PEFT methods operates at the input level rather than the weight level. Prefix tuning [8] prepends learnable continuous vectors to the key and value sequences at every Transformer layer, while prompt tuning [9] appends trainable tokens only at the input embedding layer. These methods are highly parameter-efficient but can underperform adapter-based methods in low-resource settings [23]. P-tuning v2 [24] partially addresses this by adding prompt parameters at all layers, recovering much of the lost performance.

More recently, researchers have explored tensor decomposition as a means to push parameter efficiency to an extreme. Methods such as FACT [25] and tensor-train adapters decompose the adapter weight matrices into compact multi-linear factors, achieving compression ratios far beyond what LoRA-family methods can offer. The ability to represent large matrices with a product of small tensor cores opens a path to sub-megabyte trainable parameter counts even for billion-parameter base models.

1.2.2 Zeroth-Order Optimization

Even with PEFT, fine-tuning LLMs with backpropagation requires maintaining an activation graph proportional to model depth and sequence length, consuming significant GPU memory. Zeroth-order (ZO) optimization offers an elegant alternative: gradient estimates are computed from pairs of forward evaluations, completely eliminating the backward pass and its associated memory cost.

The theoretical foundations of ZO optimization trace back to simultaneous perturbation stochastic approximation (SPSA) [26] and the randomized gradient estimator (RGE)

of Nesterov and Spokoiny [27]. Given a loss function $\ell(\mathbf{w})$ for a model with parameter $\mathbf{w}$, the RGE approximates the gradient as

$$\hat{\nabla}\ell(\mathbf{w}) = \frac{\ell(\mathbf{w} + \epsilon\mathbf{z}) - \ell(\mathbf{w} - \epsilon\mathbf{z})}{2\epsilon}\mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}),$$

where $\mathbf{z}$ is the random perturbation added on weights parameters during estimation and ϵ is a constant hyper-parameter to control the perturbation scale. The whole process requires only two forward passes with perturbed parameters. The key limitation of this estimator is its high variance, which scales with the dimension d of the parameter space, leading to slow convergence in high-dimensional settings.

The MeZO method [10] adapts the SPSA estimator to in-place parameter perturbation for LLMs, enabling memory-efficient ZO fine-tuning that requires only as much memory as inference. MeZO demonstrated over $8\times$ memory reduction compared to Adam-based fine-tuning, making it possible to fine-tune a 30B-parameter model on a single A100 GPU. However, a substantial gap remains between ZO and first-order methods in terms of task performance, and large models are prone to divergence under ZO training due to noisy gradient estimates.

Subsequent work has explored several strategies to close this performance gap. Sparse-MeZO [28] limits perturbations to a sparse subset of parameters, reducing the effective dimensionality of the ZO estimation problem. MeZO-SVRG [29] adapts the variance-reduction technique of SVRG to the ZO setting, achieving more stable convergence at the cost of additional memory for snapshot parameters. ZO-AdaMU [30] incorporates adaptive momentum into the ZO update but introduces additional optimizer state overhead that partially offsets the memory savings. Despite these efforts, no prior method simultaneously addresses the performance gap, the divergence risk, and the need for extreme parameter efficiency.

In the vision-language model (VLM) setting, ZO-based prompt tuning has emerged as an attractive approach for adapting CLIP-style models [31] on memory-constrained edge devices. Black-box prompt tuning methods such as BBT [32], BlackVIP [33], and ZIP [34] replace backpropagation with evolutionary strategies (ES) or ZO-SGD to optimize trainable prompt parameters using only forward passes. While ES methods such as CMA-ES [35] enable effective global exploration, they exhibit high variance and are prone to stagnation in the rugged loss landscapes of VLM fine-tuning. Sharpness-aware minimization (SAM) [36] has been shown to improve generalization in first-order settings by seeking flatter loss minima, but its integration with ZO methods—where gradient access is restricted—remains unexplored.

1.2.3 Inference Acceleration

Model pruning aims to reduce the computational and memory footprint of a trained network by removing unnecessary weights, either individually (unstructured pruning) or in structured groups (structured or semi-structured pruning). The foundational work of Han et al. [37] established magnitude-based pruning as a simple and effective baseline: weights with small absolute values are removed under the assumption that they contribute little to the model’s output. The lottery ticket hypothesis [38] later provided a theoretical lens for understanding why such sparse subnetworks can match the performance of the original dense network when properly initialized.

For LLMs, the scale and cost of retraining necessitate a shift toward post-training pruning methods that operate without full fine-tuning. SparseGPT [12] pioneered this direction by solving a layer-wise reconstruction problem using approximate second-order information, achieving high-quality pruning at 50% sparsity without any gradient descent steps. Wanda [11] proposed a simpler and faster alternative: a pruning score that

multiplies weight magnitude by the ℓ_2 norm of the corresponding input activation, capturing the intuition that large activations amplify small weights. Both methods have been shown to work well for unstructured sparsity, but they fall short of the near-lossless quality achieved by post-training quantization methods such as AWQ [13] at equivalent compression ratios.

The performance gap between post-training pruning and quantization can be partially explained by the absence of gradient information in pruning score computation. GBLM [39] and Pruner-Zero [40] demonstrated that incorporating first-order gradient information into the pruning score can substantially improve accuracy. Specifically, GBLM adds a gradient term to the Wanda score using gradients obtained through full-model backpropagation on the cross-entropy loss. While effective, this approach reintroduces the full memory cost of backpropagation, limiting scalability: computing full-model gradients for LLaMA-2-70B requires the same GPU resources as first-order training.

A parallel line of work focuses on recovering performance after pruning through sparsity-aware fine-tuning. Wanda [11] and SliceGPT [41] both demonstrate that applying LoRA fine-tuning after pruning can recover a significant fraction of the performance loss, at the cost of additional time and memory. Distillation-based recovery methods [42] provide another option but require a teacher model and training data.

1.2.4 Small and multi-modal LLM Agents

The vision of language model agents—systems that reason over long horizons and interact with external tools to solve complex tasks—has generated substantial research momentum. The ReAct framework [43] established a foundational paradigm in which LLMs interleave reasoning traces with action steps (tool invocations), enabling iterative problem solving. Toolformer [44] extended this idea by training LLMs to decide when and

how to call APIs in a self-supervised manner. More recently, large-scale RL training has proven highly effective at improving agent behavior: the DeepSeek-R1 model [45] demonstrated that GRPO-based RL can dramatically improve complex reasoning capabilities, even surpassing models trained with far more supervision.

As the inference cost of large models becomes a practical bottleneck, there is growing interest in deploying smaller models as agents. SLM agents [46, 47] are compact models—typically under 10 billion parameters—that are intended to run on consumer hardware or edge devices while still performing agentic tasks. Prior work has primarily trained SLM agents via supervised fine-tuning (SFT) on trajectories generated by stronger teacher models, which is simple but often suboptimal: SFT trains the model to imitate specific trajectories rather than to optimize for task success, limiting generalization to novel situations.

Reinforcement learning offers a more principled approach to SLM agent training by directly optimizing the policy for task performance. PPO [15] and its variant GRPO [16] have been successfully applied to train LLMs for reasoning and tool use. However, applying vanilla GRPO to multimodal SLMs faces specific challenges: the standard outcome-based binary reward provides extremely sparse feedback, particularly for small models that are more susceptible to reward signal noise.

The multimodal setting introduces additional complexity beyond the text-only case. Early inference-time methods such as Visual Sketchpad [48] and Refocus [49] showed that incorporating visual reasoning steps—such as cropping, highlighting, or masking regions of an image—can improve VLM performance on visual question answering. More recently, RL-based training approaches such as VTool-R1 [50] and OpenThinkIMG [51] have extended this idea to train VLMs to invoke visual tools during reasoning. However, these methods rely on vanilla GRPO with binary outcome rewards, which we show in this dissertation continues to produce substantial tool misuse even in the later stages of

training.

To address the reward sparsity problem, several recent works have proposed more fine-grained reward signals for tool-calling. ToRL [52] combines answer correctness with execution-based penalties to penalize malformed tool calls. TRM [53] trains a dedicated process reward model to evaluate tool invocation quality. However, these approaches rely on rigid format checks or exact matching against ground-truth tool trajectories, which is problematic for multimodal tasks where multiple tool-use paths may be equally valid. Furthermore, they have been evaluated primarily on large-scale models and text-only settings, leaving the specific challenges of multimodal SLM training largely unaddressed.

1.3 Overview of the Dissertation

This dissertation is organized around the theme of improving the efficiency of large language model systems across training, adaptation, inference, and agentic decision making.

Chapter 2 introduces the technical foundations that support the later chapters. It reviews existing parameter-efficient fine-tuning methods, zeroth-order optimization, and reinforcement learning for language model training. This chapter is intended to provide the mathematical and algorithmic background needed to understand the methods developed in the remainder of the dissertation.

Chapter 3 focuses on memory-efficient fine-tuning of large language models. It presents **LoRETTA** [54], a parameter-efficient fine-tuning framework based on tensor-train decomposition, which uses compact tensorized representations to reduce the number of trainable parameters required for adaptation. The chapter then introduces **AdaZeta** [55], a zeroth-order fine-tuning method built on tensorized adapters. By combining low-dimensional parameterization with adaptive query scheduling, AdaZeta

enables backpropagation free fine-tuning while improving optimization stability and efficiency. Together, these two works show how tensorized representations can support both parameter-efficient adaptation and scalable zeroth-order training.

Chapter 4 extends the study of zeroth-order optimization to vision-language models. It presents **SharpZO** [56], a two-stage framework for zeroth-order prompt tuning under memory-constrained settings. The method combines sharpness-aware global exploration with sparse local zeroth-order updates, addressing the instability and variance issues that often limit forward-only optimization. This chapter broadens the scope of the dissertation from language-only models to multimodal settings, showing that zeroth-order methods can also be effective for vision-language adaptation when combined with appropriate optimization design.

Chapter 5 turns from training-time efficiency to inference-time efficiency. It presents **Wanda++** [57], a structured pruning framework for large language models that introduces the notion of regional gradients. Instead of relying on full-model backpropagation, Wanda++ computes lightweight block-level gradient signals using local surrogate objectives, enabling more practical pruning of large models. The chapter shows how localized optimization can bridge the gap between training-free pruning efficiency and gradient-informed pruning quality.

Chapter 6 considers efficiency from a different perspective: improving the capability of smaller multimodal models used as agents. It presents **IAPO** [58], an input attribution-aware reinforcement learning method for multimodal tool use. The key idea is to enrich standard outcome-based rewards with attribution-based supervision derived from a stronger teacher model, allowing the student model to receive more informative feedback about its tool-selection behavior. This chapter illustrates how reinforcement learning objectives can be redesigned to better suit the limitations of small multimodal agents.

Finally, Chapter 7 summarizes the main contributions of the dissertation and dis-

cusses promising directions for future research. Taken together, the chapters present a unified view of efficient language model systems, spanning parameter-efficient adaptation, backpropagation-free optimization, inference-time compression, and reinforcement learning for small-model agents. Across these settings, the dissertation emphasizes that efficiency is not achieved by a single technique, but by designing algorithms that align with the computational constraints and functional requirements of modern language model deployment.

Chapter 2

Background

2.1 Background on Parameter Efficient Fine-tuning

As discussed earlier, parameter-efficient fine-tuning (PEFT) methods address this limitation by freezing most or all of the pre-trained model parameters and introducing only a small number of trainable, task-specific parameters. In this section, we provide a more detailed introduction to several widely used PEFT methods

One early and influential class of PEFT methods is adapter-based tuning. As we show in Fig. 2.1 (a), Adapter methods [6] insert small neural modules, often bottleneck feed-forward networks, into each Transformer layer while keeping the original model parameters frozen. During fine-tuning, only the adapter parameters are updated. This design allows different tasks to share the same frozen backbone while maintaining separate lightweight task-specific modules.

Another family of PEFT methods adapts models through trainable prompts rather than modifying internal weight matrices. For example, prefix-tuning [8] propose the idea of learning continuous prefix vectors that are injected into the Transformer’s attention mechanism, typically as additional key-value states at each layer. We show an example

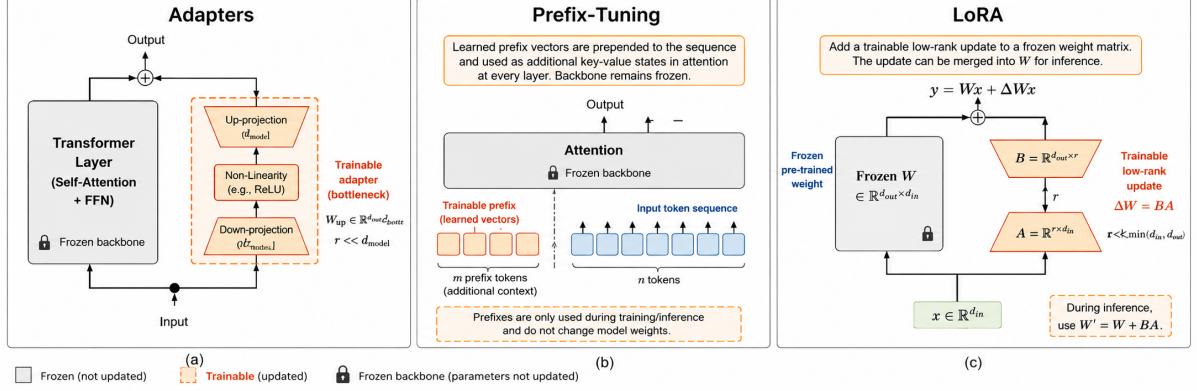


Figure 2.1: Illustration for the Adapters, Prefix-tuning and LoRA methods.

of inserting the prefix into the Attention block in Fig. 2.1 (b).

Low-rank adaptation methods have become one of the most widely used PEFT approaches for large language models. LoRA [7] freezes the pre-trained weights and injects trainable low-rank matrices into selected linear layers, such as the query and value projections in self-attention. Instead of directly updating a full weight matrix W , LoRA represents the task-specific update as a low-rank decomposition $\Delta W = BA$ as we can observe in Fig. 2.1 (c), where the rank is much smaller than the original matrix dimension. This design greatly reduces the number of trainable parameters and optimizer states, while the learned low-rank update can be merged into the original weight matrix at inference time, avoiding additional latency.

2.2 Background on Zeroth-Order Optimization

2.2.1 Coordinate-wise and Randomized Gradient Estimation

Zeroth-order (ZO) optimization methods estimate gradients using only evaluations of the objective function, without any access to analytic gradients or backpropagation. This

property makes them attractive for settings where storing the full computational graph is prohibitively expensive—for instance, when fine-tuning a large language model on a single GPU. Given a model with trainable parameters $\mathbf{w} \in \mathbb{R}^d$ and a training loss $\mathcal{L}(\mathbf{w})$, the gradient $\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w})$ is approximated via finite differences without any backward pass. The two dominant families of ZO gradient estimators are Randomized Gradient Estimation (RGE) and Coordinate-wise Gradient Estimation (CGE) [56], defined respectively as:

$$\begin{aligned} \text{(RGE)} \quad \hat{\nabla}\mathcal{L}(\mathbf{w}) &= \frac{1}{q} \sum_{i=1}^q \left[\frac{\mathcal{L}(\mathbf{w} + \mu \mathbf{u}_i) - \mathcal{L}(\mathbf{w} - \mu \mathbf{u}_i)}{2\mu} \mathbf{u}_i \right], \\ \text{(CGE)} \quad \hat{\nabla}\mathcal{L}(\mathbf{w}) &= \sum_{i=1}^d \left[\frac{\mathcal{L}(\mathbf{w} + \mu \mathbf{e}_i) - \mathcal{L}(\mathbf{w} - \mu \mathbf{e}_i)}{2\mu} \mathbf{e}_i \right], \end{aligned} \tag{2.1}$$

where $\mu > 0$ is a smoothing parameter, $\mathbf{u}_i \in \mathbb{R}^d$ is a random perturbation vector that perturbs all parameters simultaneously (typically drawn from $\mathcal{N}(0, I)$), and $\mathbf{e}_i = [0, \dots, 1, \dots, 0]^\top$ is the i -th standard basis vector. In RGE, q denotes the number of random query perturbations used to reduce estimation variance.

The two estimators differ fundamentally in their query complexity and estimation accuracy. RGE requires only $\mathcal{O}(q)$ forward passes per update step, making it far more scalable when $q \ll d$ —a regime that is essential for large language models where d can reach billions. CGE, by contrast, requires $\mathcal{O}(d)$ forward passes since it perturbs each coordinate independently; this is computationally infeasible for modern LLMs. Despite its higher query cost, CGE produces a more accurate gradient estimate: it directly approximates the true gradient via finite differences along individual coordinate axes, without introducing the smoothed surrogate objective $\mathcal{L}_\mu(\mathbf{w}) = \mathbb{E}_{\mathbf{u}}[\mathcal{L}(\mathbf{w} + \mu \mathbf{u})]$ that RGE implicitly targets [59]. In other words, RGE provides an unbiased estimate of the gradient of the smoothed loss, whereas CGE provides a more faithful approximation

of the original loss gradient. In terms of estimation variance, RGE estimators exhibit variance of order $\mathcal{O}(d/q)$, while CGE estimators enjoy a much lower variance of $\mathcal{O}(1)$ owing to their coordinate-aligned perturbations.

2.2.2 Memory-Efficient Zeroth-Order Fine-Tuning

Standard first-order fine-tuning of LLMs requires storing the full computational graph of activations across all layers during the forward pass so that backpropagation can compute exact parameter gradients. For a model with d parameters processed over a sequence of length L , this incurs memory proportional to $d \times L$, which is the dominant bottleneck when fine-tuning models with tens of billions of parameters on limited hardware.

MeZO [10] eliminates this bottleneck entirely by adapting Simultaneous Perturbation Stochastic Approximation (SPSA) [26] to an in-place perturbation scheme tailored for LLMs. The key insight is that RGE requires only two scalar loss evaluations per gradient estimate, and the perturbation direction $\mathbf{z}$ can be reproduced deterministically from a stored random seed rather than ever materializing the full d -dimensional vector. Concretely, MeZO proceeds as follows at each step: (1) sample a random seed and use it to perturb all model parameters in-place by $+\mu\mathbf{z}$; (2) perform a forward pass to evaluate $\mathcal{L}(\mathbf{w} + \mu\mathbf{z})$; (3) restore the parameters and re-perturb by $-\mu\mathbf{z}$; (4) perform a second forward pass to evaluate $\mathcal{L}(\mathbf{w} - \mu\mathbf{z})$; and (5) compute the scalar projected gradient estimate $\hat{g} = [\mathcal{L}(\mathbf{w} + \mu\mathbf{z}) - \mathcal{L}(\mathbf{w} - \mu\mathbf{z})]/(2\mu)$ and update all parameters in-place by replaying the same random seed. Because neither the perturbation vector nor the activation graph is ever held in memory in its entirety, MeZO reduces peak memory usage to $\mathcal{O}(1)$ beyond the model parameters themselves—achieving roughly an $8\times$

reduction compared with Adam-based fine-tuning [10].

Despite this memory advantage, ZO optimization via RGE converges significantly more slowly than first-order methods, and the performance gap widens on complex tasks. This limitation motivates the work in Chapter 3, where we combine tensorized adapters with an adaptive ZO query schedule to substantially close the performance gap while preserving the memory benefits of backpropagation-free training.

2.3 Background on Reinforcement Learning

Reinforcement learning (RL) offers an attractive training paradigm for large language models when the desired output quality can be assessed by a reward signal but high-quality labeled demonstrations are scarce or costly to obtain. Rather than minimizing a supervised cross-entropy loss, RL-based methods train a policy π_θ —parameterized by the model weights θ —to maximize the expected cumulative reward by exploring the output space through sampling. Early applications of RL to language model alignment, such as RLHF with Proximal Policy Optimization (PPO) [15], trained a separate value network to estimate per-token baselines for variance reduction, but the additional critic model doubles memory requirements and complicates training stability.

Group Relative Policy Optimization (GRPO) [16] simplifies the RL objective by replacing the learned value baseline with a group-level statistical baseline computed directly from sampled responses. Concretely, for each training input consisting of an image $\mathbf{I}$ and a prompt $\mathbf{x}$, GRPO samples a group of G responses $\{\mathbf{y}_1, \dots, \mathbf{y}_G\}$ from the current policy $\pi_{\text{old}}(\cdot \mid \mathbf{I}, \mathbf{x})$ and assigns each response a scalar reward R_i^{out} . In the visual tool-calling setting considered in Chapter 6, this reward is binary: $R_i^{\text{out}} = 1$ if the

final answer in response $\mathbf{y}_i$ is correct, and $R_i^{\text{out}} = 0$ otherwise [50]. The rollout follows a two-turn structure: in the first turn, the policy generates tool calls $\mathbf{y}' \sim \pi_\theta(\cdot \mid \mathbf{x}, \mathbf{I})$; a code executor processes the original image $\mathbf{I}$ according to those calls to produce a modified image $\mathbf{I}'$; and in the second turn, the policy reasons over the combined context $(\mathbf{x}, \mathbf{y}', \mathbf{I}, \mathbf{I}')$ to produce the final answer $\mathbf{y}$.

The group-relative advantage of the i -th response normalizes its reward against the within-group statistics:

$$\hat{A}_i = \frac{R_i^{\text{out}} - \text{mean}(\{R_j^{\text{out}}\}_{j=1}^G)}{\text{std}(\{R_j^{\text{out}}\}_{j=1}^G)}. \quad (2.2)$$

Using this group-relative advantage, the GRPO policy gradient objective is defined as:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E} \left[\frac{1}{G} \sum_{i=1}^G \frac{1}{|\mathbf{y}_i|} \sum_{t=1}^{|\mathbf{y}_i|} \min(r_{i,t}(\theta) \hat{A}_i, \text{clip}(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon) \hat{A}_i) - \beta D_{\text{KL}}[\pi_\theta \parallel \pi_{\text{ref}}] \right] \quad (2.3)$$

where $r_{i,t}(\theta) = \pi_\theta(y_{i,t} \mid \mathbf{x}, y_{i,<t}) / \pi_{\text{old}}(y_{i,t} \mid \mathbf{x}, y_{i,<t})$ is the token-level importance ratio, ϵ is the clipping threshold that limits the policy update magnitude, β is the coefficient for KL regularization toward the reference policy π_{ref} , and the same advantage $\hat{A}_i$ is shared across all tokens in response $\mathbf{y}_i$ since the reward is defined at the response level.

Although GRPO successfully improves overall task performance, its vanilla objective provides no direct supervision over the intermediate decisions made during tool selection and invocation. A model may invoke an incorrect or irrelevant tool yet still receive a positive reward if the final answer happens to be correct—a severe credit assignment problem. This issue is especially acute for small language models (SLMs),

whose weaker grounding and reasoning capabilities make them more susceptible to learning spurious correlations rather than principled tool-selection policies. Designing GRPO-compatible rewards that provide dense, intermediate-step guidance for multimodal tool-calling is the central challenge addressed in [Chapter 6](#).

Chapter 3

Efficient LLM Fine-tuning via Zeroth-Order Optimization

3.1 Introduction

Fine-tuning pre-trained large language models on downstream tasks has become the dominant paradigm for deploying LLMs in practice. However, as model scale continues to grow, the cost of fine-tuning grows in two independent dimensions. The first is parameter efficiency: even with LoRA [7], fine-tuning LLaMA-2-70B requires updating more than 16 million parameters, a count that exceeds the total parameter count of many BERT-scale models. The second is memory efficiency: standard first-order fine-tuning must store a full computational graph of intermediate activations across all layers for backpropagation, which can consume tens to hundreds of gigabytes beyond the model itself. While these two challenges are often treated independently in the literature, this chapter argues that they are deeply coupled and can be addressed simultaneously through the lens of tensor-train decomposition.

This chapter presents two complementary contributions. The first, **LoRETTA** [54]

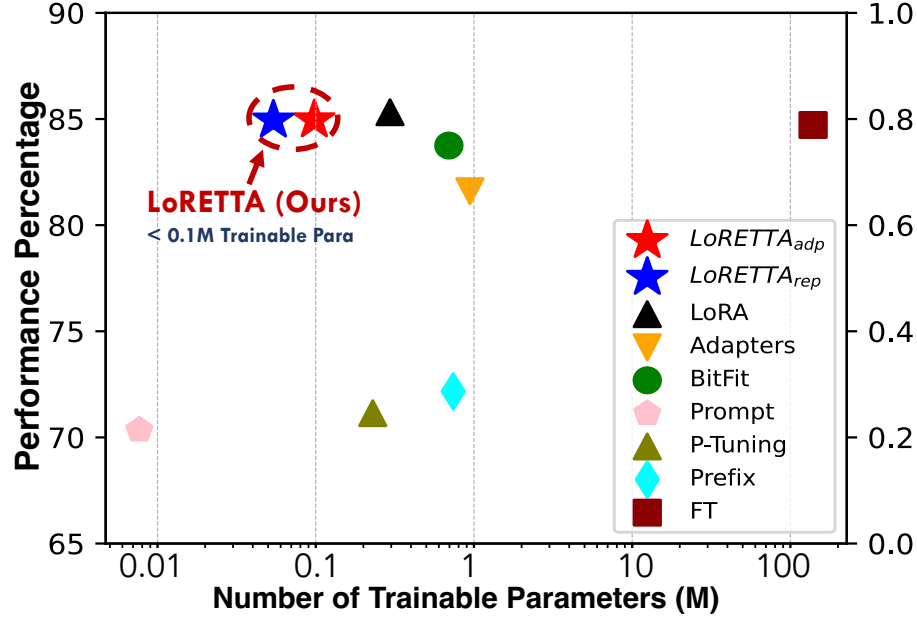


Figure 3.1: Performance vs. trainable parameters on DeBERTa-Base across GLUE tasks. LoRETTA_{adp} and LoRETTA_{rep} achieve competitive or superior performance compared to other PEFT methods while using substantially fewer parameters. Figure reproduced from [54].

(Low-rank Economic Tensor-Train Adaptation), targets parameter efficiency. LoRETTA leverages the tensor-train (TT) format introduced in Chapter 2 to compress the trainable modules injected during fine-tuning. By replacing the weight matrices in adapter blocks and reparameterization layers with their TT representations, LoRETTA achieves parameter counts that are 80–100× smaller than LoRA-scale methods, while maintaining competitive or superior downstream accuracy. On LLaMA-2-7B, LoRETTA’s adapter variant requires only 0.88M trainable parameters—fitting comfortably within 1 MB of storage—and outperforms LoRA on 5 of 7 SuperGLUE tasks.

The second contribution, Adaptive Zeroth-Order Tensor-Train Adaptation (AdaZeta) [55], targets memory efficiency by eliminating backpropagation entirely. Building on the MeZO framework [10] reviewed in Section 2.2.2, AdaZeta combines the ultra-low-dimensional TT adapters from LoRETTA with two new algorithmic contributions: a fast-forward parallel contraction strategy that reduces the per-step forward pass cost,

and an adaptive query number schedule that stabilizes zeroth-order (ZO) gradient estimation for large-scale models. Together, these innovations resolve the divergence issues observed with prior ZO fine-tuning methods such as MeZO-LoRA, and yield superior convergence speed and final accuracy across a wide range of tasks. On LLaMA-2-7B, AdaZeta outperforms all ZO baselines on 8 of 10 benchmarks while requiring only 14 GB of peak GPU memory—an $8\times$ reduction relative to full fine-tuning with AdamW.

The main contributions of this chapter are as follows:

- **LoRETTA**: a TT-based PEFT framework with two variants—tensorized adapters (LoRETTA_{adp}) and TT reparameterization (LoRETTA_{rep})—achieving up to $100\times$ fewer trainable parameters than LoRA on LLaMA-2-scale models, with demonstrated benefits in anti-overfitting and multi-task learning.
- **AdaZeta**: a backpropagation-free fine-tuning framework combining TT adapters with a sublinear adaptive query schedule; outperforms MeZO, MeZO-LoRA, and Sparse-MeZO [28] across diverse tasks while converging up to $6\times$ faster.
- **Theoretical guarantee**: a non-convex convergence analysis showing that the bound improves multiplicatively from both the dimensionality reduction (smaller d) and the growing query count (Q_k), providing a principled explanation for AdaZeta’s empirical advantage.

3.2 Background on Tensor Decomposition

Tensor decomposition provides a compact representation of high-dimensional arrays by factorizing them into structured collections of smaller tensors. The Tensor-Train (TT) decomposition [60], also known as matrix product states in physics, is particularly well-suited for compressing the weight matrices of neural networks because it scales

logarithmically in the number of tensor factors rather than polynomially in the matrix dimensions.

Consider a standard fully connected layer whose forward pass is $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$, where $\mathbf{W} \in \mathbb{R}^{M \times N}$ is the weight matrix, $\mathbf{x} \in \mathbb{R}^N$ is the input vector, and $\mathbf{b}$ is the bias. The TT decomposition begins by reshaping $\mathbf{W}$ into a d -way tensor $\mathcal{W} \in \mathbb{R}^{k_1 \times \dots \times k_d}$, where $\prod_{i=1}^d k_i = M \times N$. This reshaped tensor is then expressed as a contracted product of d small tensor cores $\mathcal{G}_1, \dots, \mathcal{G}_d$, where each core $\mathcal{G}_i \in \mathbb{R}^{r_{i-1} \times k_i \times r_i}$ has three modes: a left bond dimension r_{i-1} , a physical dimension k_i , and a right bond dimension r_i . The boundary conditions are set as $r_0 = r_d = 1$ so that the full contraction yields a scalar result for each element. Any element of $\mathcal{W}$ indexed by $(a_1, \dots, a_d)$ can then be computed as a chain of matrix multiplications among the corresponding matrix slices of each core:

$$\mathcal{W}(a_1, \dots, a_d) = \mathbf{G}_1^{a_1} \cdots \mathbf{G}_i^{a_i} \cdots \mathbf{G}_d^{a_d}, \quad (3.1)$$

where $\mathbf{G}_i^{a_i} := \mathcal{G}_i(:, a_i, :) \in \mathbb{R}^{r_{i-1} \times r_i}$ denotes the a_i -th frontal slice of core $\mathcal{G}_i$. Stacking these slices back into the full tensor yields the TT representation of the weight matrix:

$$\text{TT}(\mathbf{W}) := \prod_{i=1}^d \mathcal{G}_i[r_{i-1}, k_i, r_i]. \quad (3.2)$$

The total number of parameters in the TT representation is $\sum_{i=1}^d r_{i-1} k_i r_i$, which is dramatically smaller than the original $M \times N$ parameters when the TT ranks r_i are small. For simplicity, a common practice is to fix all interior ranks to a single constant value r , after which the compression ratio scales as $\mathcal{O}(r^2 dk)$ versus $\mathcal{O}(M \cdot N)$.

In a tensorized linear layer, one replaces the original weight matrix entirely with

its TT factors. During training, only the small tensor cores $\mathcal{G}_1, \dots, \mathcal{G}_d$ are stored and optimized, while the conceptually large matrix $\mathbf{W}$ is never materialized. To illustrate the degree of compression, consider a typical transformer hidden size of 768. If the weight matrix is reshaped into a tensor of shape $[8, 8, 8, 8, 8, 8]$ with TT rank 5, the number of trainable parameters for a single adapter weight drops from roughly 98K to just 1.2K—a compression ratio exceeding $80\times$ [54]. The TT ranks can also be adjusted adaptively during training to further balance compression and accuracy [61].

A complementary use of TT decomposition is to reparameterize the weight update matrix rather than the base weight itself. This mirrors the intuition behind LoRA [7], which decomposes the update $\Delta\mathbf{W}$ into a product of two low-rank matrices $B \in \mathbb{R}^{h \times r}$ and $A \in \mathbb{R}^{r \times l}$ and writes the layer output as $\mathbf{y} = \mathbf{W}_0\mathbf{x} + B\mathbf{A}\mathbf{x}$. Instead of a single matrix factorization, the update $\Delta\mathbf{W}$ can be represented through two tensorized layers—each expressed in TT format with tensor factors $(\mathcal{G}_1, \dots, \mathcal{G}_d)$ and $(\mathcal{Q}_1, \dots, \mathcal{Q}_d)$ respectively—yielding:

$$\mathbf{y} = \mathbf{W}_0\mathbf{x} + \text{TT}(\Delta\mathbf{W}_{\text{up}}) \cdot \text{TT}(\Delta\mathbf{W}_{\text{down}})\mathbf{x} = \mathbf{W}_0\mathbf{x} + \prod_{i=1}^d \mathcal{G}_i \prod_{i=1}^d \mathcal{Q}_i \mathbf{x}. \quad (3.3)$$

This TT reparameterization reduces the update-matrix parameter count from 12K to 1K compared with LoRA at rank 8 for a hidden size of 768 and TT rank 5. One subtle challenge is initialization: LoRA sets $B = 0$ so that $\Delta\mathbf{W} = B\mathbf{A} = 0$ at the start of training, ensuring fine-tuning begins from the pre-trained checkpoint. With multiple tensor cores, zeroing out a single core more readily causes zero-gradient pathologies because the gradient with respect to any one factor depends on the product of all others. A tensor reconstruction initialization [62]—which converts the randomly initialized factors back into matrix form, measures the noise introduced by this initialization, and subtracts it

from the pre-trained weight—resolves this issue and provides a stable starting point for optimization.

3.3 Parameter-Efficient Fine-Tuning through Tensor Decomposition

3.3.1 Motivation

Research on the intrinsic dimensionality of language model fine-tuning [17] shows that fine-tuning within a randomly projected subspace of only 200–800 dimensions can retain approximately 90% of the performance achieved by full fine-tuning on a RoBERTa model. This finding implies that the true degrees of freedom needed to adapt a pre-trained model to a downstream task are far smaller than the model’s parameter count would suggest. Yet current PEFT methods are far from reaching this theoretical lower bound. Standard Adapters with bottleneck size $r = 8$ require roughly 1 M parameters on a DeBERTa-Base model; LoRA with rank 8 needs 0.30 M. On LLaMA-2-70B, LoRA with rank 8 requires over 16 M trainable parameters—more than the total parameter count of some BERT-scale models [54]. The gap between the intrinsic dimension (~ 800) and the practical PEFT parameter count ($\sim 10^6$ – 10^7) motivates the search for more aggressive compression strategies.

Tensor-train decomposition offers a systematic path to closing this gap. As established in Section 3.2, the TT format factorizes a weight matrix of size $M \times N$ into d small cores $\mathcal{G}_i \in \mathbb{R}^{r_{i-1} \times k_i \times r_i}$, reducing the total parameter count from $M \times N$ to $\sum_{i=1}^d r_{i-1} k_i r_i$. Crucially, this compression ratio scales with the TT rank r and the number of factors d , not with the model’s hidden dimension h . This means that as models grow larger, the TT-based compression ratio increases rather than decreases, making TT decomposition

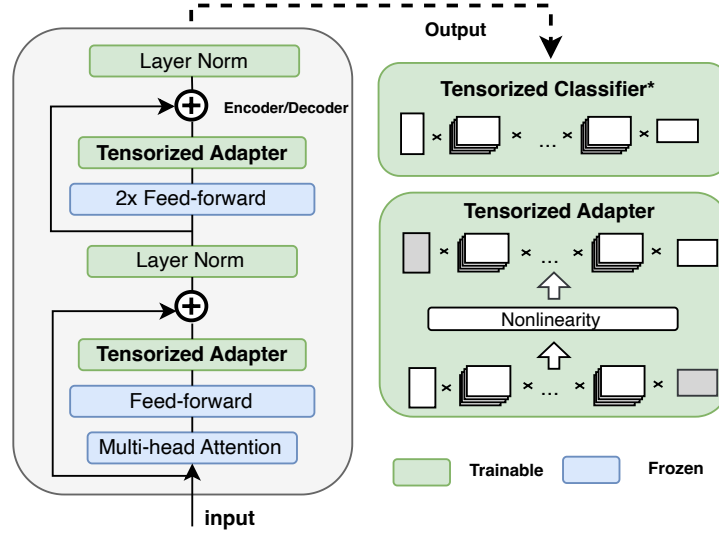


Figure 3.2: Architecture of $\text{LoRETTA}_{\text{adp}}$ for the transformer encoders or decoders. * the tensorized classifier is optional for different tasks. For classification tasks, we set this part to be trainable and we freeze this part during language modeling tasks.

a particularly attractive strategy for the large-model regime.

3.3.2 The LoRETTA Architecture

LoRETTA introduces two complementary variants that apply TT-based compression to the two dominant PEFT paradigms: adapter insertion and weight reparameterization. Both variants are built based on the tensorized linear layer introduced in Section 3.2

LoRETTA_{adp}: Tensorized Adapters. As shown in Fig. 3.2, the $\text{LoRETTA}_{\text{adp}}$ method adapt from a standard adapter block inserts two linear projections—a down-projection $\mathbf{W}_{\text{down}} \in \mathbb{R}^{r \times h}$ and an up-projection $\mathbf{W}_{\text{up}} \in \mathbb{R}^{h \times r}$ —with a nonlinear activation in between. With $h = 768$ and $r = 64$, a single adapter requires approximately $2 \times 768 \times 64 \approx 98\text{K}$ parameters for the weight matrices alone. $\text{LoRETTA}_{\text{adp}}$ replaces both projections with tensorized TT layers (Section 3.2). Setting the tensor shape to $[8, 8, 8, 8, 8, 8]$ with TT rank 5 reduces the parameter count to $\sum_{i=1}^6 (5^2 \cdot 8) = 1.2\text{K}$ —a compression of more than $80\times$. A pair of such tensorized adapters (one after

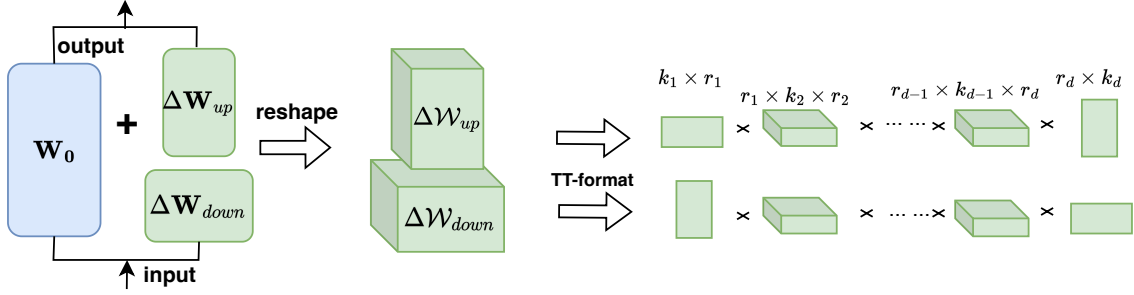


Figure 3.3: Architecture of the LoRETTA_{rep} method for a single transformer encoder.

the attention sub-layer, one after the feed-forward sub-layer) is inserted into each transformer block, following the standard adapter placement strategy [6]. For classification tasks, the layer normalization and the final classifier layer are also made trainable; for generation tasks, the layer normalization alone is fine-tuned and the language model head is kept frozen. On LLaMA-2-7B, the full LoRETTA_{adp} configuration requires only 0.88M trainable parameters, compared to 4.19M for LoRA at rank 8.

LoRETTA_{rep}: TT Reparameterization. The second variant applies TT compression to the weight update matrix rather than the base weights, following the spirit of LoRA. The main idea is illustrated in Fig. 3.2. Whereas LoRA decomposes $\Delta \mathbf{W}$ into a product of two low-rank matrices BA , LoRETTA_{rep} represents $\Delta \mathbf{W}$ through two TT layers as derived in Eq. (3.3):

$$\mathbf{y} = \mathbf{W}_0 \mathbf{x} + \prod_{i=1}^d \mathcal{G}_i \prod_{i=1}^d \mathcal{Q}_i \mathbf{x}.$$

For a hidden size of 768 and TT rank 5, this reduces the update-matrix parameter count from 12K (LoRA, rank 8) to 1K—a 12× compression at the per-layer level. On LLaMA-2-7B, LoRETTA_{rep} requires only 0.51M trainable parameters, fitting the entire

fine-tuning state within 1 MB of storage.

Initialization. As discussed in Section 3.2, naively zeroing one tensor core to match LoRA’s zero-initialization strategy creates zero-gradient pathologies for all remaining cores. LoRETTA instead uses tensor reconstruction initialization: the TT cores are first initialized randomly from a Gaussian distribution, then the resulting matrix is reconstructed, the mean of the reconstruction noise is computed, and this noise is subtracted from the pre-trained weight $\mathbf{W}_0$. This ensures the model output at the start of training is identical to the pre-trained checkpoint while avoiding gradient degeneracy.

3.3.3 Experiments with First-Order Optimizers

Setup. All LoRETTA experiments use the AdamW optimizer [63] on NVIDIA V100-16GB and A100-40GB GPUs. Baselines include full fine-tuning (FT), Adapters [6], LoRA [7], Prefix-tuning [8], Prompt-tuning [9], P-tuning [64], BitFit [65], and IA³ [66], all implemented via the HuggingFace PEFT library. For BERT-family models, evaluation uses the GLUE benchmark [67]; for large-scale models, a low-data SuperGLUE [68] setup with 1,000 training examples is used to evaluate LLaMA-2-7B, -13B, and -70B [3].

Results on BERT-Family Models. Table 3.1 summarizes results on DeBERTa-Base and RoBERTa-Base. On DeBERTa-Base [69], LoRETTA_{adp} (0.10M parameters) achieves an average GLUE score of 84.96, outperforming BitFit at equal parameter count (83.75) and matching or exceeding LoRA _{$r=4$} (84.54) which uses 50% more trainable parameters. LoRETTA_{rep} (0.05M parameters) achieves an average of 84.95 with

$6\times$ fewer parameters than $\text{LoRA}_{r=8}$ (0.30M) and a performance gap of less than 0.6%. On RoBERTa-Base, $\text{LoRETTA}_{\text{adp}}$ (0.10M) scores 85.26, outperforming $\text{LoRA}_{r=8}$ (84.69) while using $6\times$ fewer parameters.

Model & Method	# Train. Param.	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
DeBERTa-Base (FT)	139.19M	88.67	94.61	91.98	59.32	93.04	91.42	68.23	91.10	84.79
DeBERTa-Base (Adapters $_{r=8}$)	0.94M	87.69	94.72	88.88	54.19	92.95	85.52	59.20	89.68	81.60
DeBERTa-Base (LoRA $_{r=8}$)	0.30M	87.30	94.95	92.84	60.56	93.35	85.19	80.14	90.13	85.56
DeBERTa-Base (P-Tuning)	0.23M	56.25	91.39	79.93	43.31	86.30	78.43	55.95	78.38	71.24
DeBERTa-Base (LoRA $_{r=4}$)	0.15M	87.69	94.49	91.10	62.57	92.60	87.30	69.67	91.12	84.54
DeBERTa-Base (Prompt)	0.01M	77.63	92.43	81.90	32.99	80.30	78.15	62.81	56.71	70.36
DeBERTa-Base (Prefix)	0.15M	60.32	88.87	81.22	45.82	83.28	82.22	59.57	84.99	73.28
DeBERTa-Base (BitFit)	0.10M	84.63	95.41	91.42	64.06	93.20	84.15	66.79	90.23	83.75
DeBERTa-Base (LoRETTA$_{\text{adp}}$)	0.10M	85.93	95.30	93.53	60.84	92.99	84.08	75.50	91.32	84.96
DeBERTa-Base (LoRETTA$_{\text{rep}}$)	0.05M	86.80	95.53	88.73	59.69	93.25	89.20	75.81	90.66	84.95
RoBERTa-Base (BitFit)*	0.10M	85.30	94.80	92.33	62.70	91.30	68.10	73.60	88.50	82.08
RoBERTa-Base (LoRA $_{r=8}$)*	0.63M	86.82	94.01	91.48	62.08	92.39	85.71	74.51	90.48	84.69
RoBERTa-Base (LoRETTA$_{\text{adp}}$)	0.10M	85.61	94.38	91.08	62.70	92.12	87.22	78.70	90.26	85.26
RoBERTa-Base (LoRETTA$_{\text{rep}}$)	0.07M	84.40	94.28	90.63	61.72	92.40	85.23	74.42	89.24	84.04

Table 3.1: Comparative analysis of various PEFT methods on the BERT family models (including RoBERTa-base and DeBERTa-base models). The best results are bolded for the PEFT method with trainable parameters lower than 0.2M. * represents results shown in previous works [70, 65]. Different from the LoRA paper [7], we use the F1 score for the MRPC and QQP tasks.

Results on LLaMA-2-7B. As shown in Table 3.2, under the low-data SuperGLUE setting, $\text{LoRETTA}_{\text{adp}}$ (0.88M parameters) outperforms $\text{LoRA}_{r=8}$ (4.19M parameters) on 5 of 7 tasks, including BoolQ (87.0 vs. 84.8), WSC (63.46 vs. 62.50), COPA (87 vs. 81), and DROP (51.60 vs. 45.96). $\text{LoRETTA}_{\text{rep}}$ (0.51M parameters) achieves competitive performance on multiple-choice and generation tasks while fitting its entire fine-tuning state within 1 MB. These results demonstrate that TT-based compression does not sacrifice quality even at billion-parameter scale.

Results on LLaMA-2-13B and LLaMA-2-70B. Table 3.3 reports results on the larger LLaMA-2 variants. $\text{LoRETTA}_{\text{adp}}$ continues to deliver competitive performance against LoRA and Adapters while using substantially fewer trainable parameters, achieving the best DROP score on both 13B (59.41) and 70B (74.50) and the best SQuAD score on 70B (94.33).

Model & Method	Train. Param.	Classification			Multiple Choice		Generation	
		CB	BoolQ	WSC	COPA	ReCoRD	SQuAD	DROP
LLaMA2-7B (FT)	6738.42M	66.07	84.6	63.46	86	81.1	90.71	51.38
LLaMA2-7B (Adapter)	50.33M	66.07	71.8	62.50	84	78.8	88.45	49.14
LLaMA2-7B (LoRA _{r=8})	4.19M	67.86	84.8	62.50	81	79.4	90.56	45.96
LLaMA2-7B (Prefix)	1.31M	51.78	78.6	61.53	83	81.0	90.56	45.95
LLaMA2-7B (IA3)	0.60M	64.29	72.3	36.53	80	81.5	89.41	39.37
LLaMA2-7B (LoRETТА_{rep})	0.51M	55.35	78.1	57.61	86	80.3	88.47	42.71
LLaMA2-7B (LoRETТА_{adp})	0.88M	66.07	87.0	63.46	87	80.0	90.17	51.60

Table 3.2: Performance Comparison on LLaMA-2-7B with low data resource setting (1000 examples). LoRETТА_{adp} outperforms other widely used PEFT methods among most tasks.

Model & Method	LLaMA-2-13B					LLaMA-2-70B		
	Param.	COPA	ReCoRD	SQuAD	DROP	Param.	SQuAD	DROP
Adapters	79.05M	90	83.8	93.37	57.41	252.97M	93.37	68.12
LoRA _{r=8}	6.55M	90	83.4	92.71	59.13	16.38M	93.78	72.99
IA3	0.96M	85	84.2	91.81	51.48	2.45M	92.85	71.48
LoRETТА_{rep}	0.77M	86	84.4	90.87	53.19	1.99M	90.18	68.83
LoRETТА_{adp}	1.67M	90	83.9	92.67	59.41	4.79M	94.33	74.50

Table 3.3: Performance Comparison on LLaMA-2-13B and LLaMA-2-70B. We compare our proposed method with LoRA, which is one of the most widely used high-performance PEFT methods.

Memory Efficiency. As shown in Table 3.4, LoRETТА significantly reduces the memory footprint of the stored fine-tuning state, training FLOPs (due to fewer adapter parameters), and gradient communication overhead—making it particularly attractive for federated or distributed fine-tuning scenarios [71].

Model & Method	Memcpy (μ s)	FLOPs (Reduction)
LLaMA2-7B (Adapter)	10590	6.18×10^{15} (Baseline)
LLaMA2-7B (LoRA)	45674	6.145×10^{15} (-4.2×10^{13})
LLaMA2-7B (LoRETТА _{adp})	9879	6.141×10^{15} (-4.6×10^{13})

Table 3.4: Memory profiling and FLOPs analysis.

Anti-Overfitting and Multi-Task Benefits. The extremely small parameter space of TT adapters acts as a powerful implicit regularizer. In settings with limited training data or many tasks trained simultaneously, the reduced parameter count prevents the model from memorizing task-specific spurious patterns, leading to better generalization. As shown in Figure 3.4, LoRETТА exhibits significantly lower overfitting rates

than Adapters and LoRA under identical data budgets, and retains previously learned task knowledge more effectively in multi-task learning scenarios.

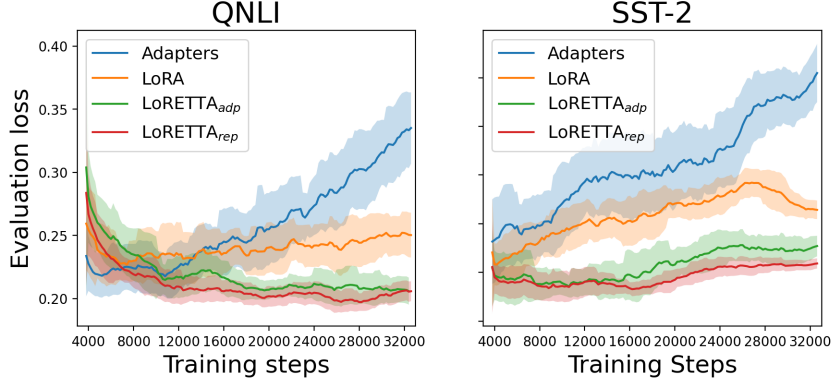


Figure 3.4: Evaluation loss comparison across various PEFT methods on the DeBERTa-base model. The loss is smoothed with a window size of 20 and the shallow means the standard deviation boundaries.

3.4 Zeroth-Order Tensor-Train Adaptation

3.4.1 Limitations of Existing Zeroth-Order Methods

MeZO (Section 2.2.2) eliminates the backpropagation graph entirely by performing in-place parameter perturbation, achieving roughly an $8\times$ reduction in peak memory usage compared to Adam-based fine-tuning. Despite this advantage, MeZO’s reliance on randomized gradient estimation (RGE) introduces a fundamental tension: the variance of the RGE estimator scales as $\mathcal{O}(d/q)$, where d is the number of trainable parameters and q is the number of random queries per step. For full-model MeZO applied to LLaMA-2-7B ($d \approx 7 \times 10^9$), this variance is enormous even with $q = 1$, leading to slow and noisy convergence.

A natural mitigation is to reduce d by applying MeZO only to PEFT adapters rather than the full model, as in MeZO-LoRA. However, empirical observation reveals that

MeZO-LoRA still suffers from divergence at LLaMA-2-7B scale, particularly on tasks such as SST2, RTE, and BoolQ. The instability arises because the gradient noise, though reduced, remains high enough to prevent reliable descent when the query count is fixed at $q = 1$. Increasing q to a fixed large value (e.g., $q = 10$) resolves divergence but multiplies training time by $10\times$, negating the computational advantage of ZO methods. This motivates a design that reduces d more aggressively than LoRA—using TT adapters—and adapts q dynamically rather than fixing it.

3.4.2 Algorithm Design

The AdaZeta framework combines two components to simultaneously address the accuracy and stability problems of ZO fine-tuning.

Tensorized Adapters for Dimensionality Reduction. AdaZeta injects LoRETTA_{adp}-style TT adapters into the model, reducing the number of trainable parameters d from billions (full model) or millions (LoRA) to thousands (TT adapters). The layer normalization is kept frozen during ZO fine-tuning, as the ZO gradient estimate of the normalization scaling factor is highly noisy and degrades performance when updated. As shown by the convergence analysis in Section 3.5, this dimensionality reduction directly shrinks the constant $C(d, \epsilon)$ in the convergence bound, improving the rate at which the gradient norm decreases.

The standard forward pass of a TT layer evaluates the product of d tensor cores sequentially: at each step, the current output tensor is contracted with the next core, producing an intermediate result whose size grows with the TT rank. This sequential process creates a bottleneck for ZO fine-tuning, which requires two forward passes per gradient estimate. Any slowdown in the forward pass is therefore doubled relative to

first-order methods.

AdaZeta addresses this through a parallel contraction strategy that eliminates the sequential dependency between cores. Instead of contracting from left to right, the tensor cores are divided into two (or more) groups, each group is contracted independently in parallel, and the partial results are combined at the end. In the bipartite case for a layer with $2o$ tensor cores, the weight reconstruction becomes:

$$\mathbf{W} = \text{R} \left(\prod_{i=1}^o \mathcal{G}_i \cdot \prod_{j=o+1}^{2o} \mathcal{G}_j \right), \quad (3.4)$$

where $\text{R}(\cdot)$ denotes the reshape operation and the two products are computed independently. Because both halves involve only o small cores each, the intermediate tensors never grow to the size they would reach in a single sequential pass. For larger models, a tripartite or quadripartite split can be used, and all groups can be computed in parallel across different compute units. In practice, parallel contraction achieves approximately 20% speedup in forward pass latency compared with sequential contraction on LLaMA-2-7B [55].

Adaptive Query Number Schedule. To resolve the divergence-versus-speed trade-off, AdaZeta introduces a sublinear query schedule that starts with a small query count and increases it gradually over training. The query number at training step k , which falls within epoch $e_k = \lfloor k / \lceil D/B \rceil \rfloor$ (where D is dataset size and B is batch size), is set as:

$$Q_k := \min \left(\alpha e_k^\beta, Q_{\max} \right), \quad (3.5)$$

with scaling factor $\alpha \in (0, 1)$, sublinear growth exponent $\beta \in (0, 1)$, and a maximum query cap $Q_{\max}$. The query number is held constant within each epoch and updated at

Algorithm 1 AdaZeta Algorithm

Input: Parameters $\mathbf{w}$, loss $\ell(\cdot)$, random seeds $\{s_q\}$, perturbation scale ϵ , schedule constants α, β , max queries $Q_{\max}$, learning rate η .

- 1: **for** step $k = 1, \dots, K$ **do**
- 2: Compute query count: $Q_k \leftarrow \min(\alpha e_k^\beta, Q_{\max})$
- 3: **for** query $q = 1, \dots, Q_k$ **do**
- 4: $\mathbf{w} \leftarrow \mathbf{w} + \epsilon \mathbf{z}_q$, $\mathbf{z}_q \sim \mathcal{N}(0, \mathbb{I}_d, s_q)$
- 5: $\ell_+^q \leftarrow \ell(\mathbf{w}, \mathcal{B})$
- 6: $\mathbf{w} \leftarrow \mathbf{w} - 2\epsilon \mathbf{z}_q$, $\mathbf{z}_q \sim \mathcal{N}(0, \mathbb{I}_d, s_q)$
- 7: $\ell_-^q \leftarrow \ell(\mathbf{w}, \mathcal{B})$
- 8: $\mathbf{w} \leftarrow \mathbf{w} + \epsilon \mathbf{z}_q$ (restore)
- 9: **end for**
- 10: $\hat{\nabla} \ell(\mathbf{w}) \leftarrow \frac{1}{Q_k} \sum_{q=1}^{Q_k} \frac{\ell_+^q - \ell_-^q}{2\epsilon} \mathbf{z}_q$
- 11: $\mathbf{w} \leftarrow \mathbf{w} - \eta \hat{\nabla} \ell(\mathbf{w})$
- 12: **end for**

epoch boundaries. In early training, Q_k is small (often 1), enabling fast gradient steps. As training progresses and the model approaches the loss basin, Q_k grows, reducing variance and enabling more precise convergence. Crucially, because the query count grows sublinearly, the total number of forward passes grows as $\mathcal{O}(K^{1+\beta})$ rather than $\mathcal{O}(K \cdot Q_{\max})$, keeping computational cost manageable.

The full AdaZeta optimization procedure is summarized in Algorithm 1.

3.5 Theoretical Analysis

We analyze the convergence behavior of the AdaZeta algorithm in a non-convex optimization setting, focusing on the interplay between trainable parameter dimensionality and the query schedule. Let $\mathbf{w} \in \mathbb{R}^d$ denote the TT adapter parameters (with $d \ll$ model size), $\ell(\mathbf{w}; \mathcal{B})$ the mini-batch loss, and $\hat{\nabla} \ell(\mathbf{w}_k)$ the RGE gradient estimate with Q_k queries at step k . We require two standard assumptions:

Assumption 3.1 (Lipschitz Gradient). *The loss ℓ has an L -Lipschitz continuous gradient: $\|\nabla\ell(\mathbf{w}_i) - \nabla\ell(\mathbf{w}_j)\| \leq L\|\mathbf{w}_i - \mathbf{w}_j\|$ for all $\mathbf{w}_i, \mathbf{w}_j$.*

Assumption 3.2 (Bounded Gradient). *At each step k , the gradient satisfies $\|\nabla\ell(\mathbf{w}_k)\| \leq \delta$.*

Theorem 3.1 (AdaZeta Convergence). *Under Assumptions 1 and 2, selecting $\mathbf{w}_T$ uniformly at random from the history $\{\mathbf{w}_1, \dots, \mathbf{w}_K\}$, the convergence of AdaZeta satisfies:*

$$\mathbb{E}[\|\nabla\ell(\mathbf{w}_T)\|^2] \leq \mathcal{O}\left(\frac{R + \epsilon^2 L + C(d, \epsilon) \sum_{k=1}^K \frac{1}{Q_k}}{K\epsilon}\right), \quad (3.6)$$

where $R = \ell(\mathbf{w}_1) - \ell^*$ is the initial suboptimality, ϵ is the perturbation scale, and $C(d, \epsilon)$ is a constant that grows with the trainable parameter dimension d .

Proof. Details can be found in Appendix 7.4. □

The bound in Eq. (3.6) reveals three key insights that theoretically justify the AdaZeta design choices.

Dimension reduction helps. The constant $C(d, \epsilon)$ is directly proportional to d . By reducing d from millions (LoRA) to thousands (TT adapters), AdaZeta shrinks this constant by two to three orders of magnitude, dramatically tightening the convergence bound. This is the theoretical justification for using TT adapters over LoRA in the ZO setting: beyond the obvious variance reduction from smaller d/q , the constant term in the bound itself scales down.

Adaptive queries guarantee convergence. With a fixed query count $Q = 1$, the sum $\sum_k 1/Q_k = K$ grows linearly, and the $C(d, \epsilon) \cdot K$ term in the numerator cancels the K in the denominator, preventing the bound from vanishing. With the sublinear schedule $Q_k = \min(\alpha e_k^{0.5}, Q_{\max})$, the sum satisfies $\sum_{k=1}^K 1/Q_k \leq 2\lceil D/B \rceil \sqrt{K/\lceil D/B \rceil}$,

which grows as $\mathcal{O}(\sqrt{K})$. Dividing by K , the problematic term decays as $\mathcal{O}(1/\sqrt{K}) \rightarrow 0$, guaranteeing that the gradient norm converges to zero as $K \rightarrow \infty$.

Joint benefit is multiplicative. Reducing d scales down the coefficient $C(d, \epsilon)$, while growing Q_k reduces the sum $\sum_k 1/Q_k$. These two effects multiply, meaning the benefit of combining TT adapters with adaptive queries is strictly greater than the sum of their individual contributions.

3.6 Experimental Results

3.6.1 Experimental Setup

Models. Two model scales are evaluated: RoBERTa-Large (350M parameters) [72] for the medium-scale ZO experiments, and LLaMA-2-7B [3] for the large-scale experiments.

Datasets. For RoBERTa-Large, experiments follow the 16-shot protocol from MeZO [10], covering SST-2, SST-5, QNLI, MNLI, SNLI, RTE, and Movie Reviews (MR). For LLaMA-2-7B, the low-data SuperGLUE setup is used (1,000 training examples per task) covering classification (RTE, CB, BoolQ, WSC, WiC), multiple-choice (COPA), and generation (MultiRC, ReCoRD, SQuAD, DROP).

Baselines. ZO baselines include MeZO [10], MeZO-LoRA, and Sparse-MeZO [28]. Inference-only baselines include Zero-Shot, In-Context Learning (ICL), and Linear Probing (LP). First-order fine-tuning (FT) and LoRA are included as reference upper bounds.

Hardware and implementation. All AdaZeta experiments use NVIDIA A100-

40GB GPUs. The adaptive query schedule uses $\alpha = \beta = 0.5$ and $Q_{\max} = 3$ for LLaMA-2-7B; query count is fixed at 1 for RoBERTa-Large experiments, since medium-scale gradient noise is manageable without adaptation.

3.6.2 Results on Zeroth-Order Fine-Tuning

RoBERTa-Large (16-shot). The results are shown in Table. 3.5. With the query count fixed at 1, AdaZeta (tensorized adapters only, without the adaptive schedule) already outperforms MeZO-LoRA on 5 of 7 tasks at batch size 16, with a particularly large margin on RTE (66.8 vs. 52.7). This demonstrates that dimensionality reduction from TT adapters is the primary driver of ZO accuracy improvement—before any query adaptation is introduced. Importantly, AdaZeta shows stable convergence at batch size 16 (where MeZO-LoRA degrades due to noise), confirming that the lower parameter count mitigates the gradient instability observed with larger PEFT modules.

Methods	RTE	SST-2	SST-5	QNLI	MNLI	SNLI	MR
FT	66.4	91.9	47.5	63.4	70.0	77.5	88.2
Zero-Shot	51.4	79.0	35.5	50.9	48.8	50.2	80.2
LP	59.4	76.0	40.3	57.6	56.5	66.0	86.6
BS=16							
MeZO	52.7	90.5	31.1	59.9	60.5	63.5	85.5
MeZO-LoRA	52.7	84.2	44.8	60.3	58.5	65.6	85.7
AdaZeta	66.8	91.4	48.3	61.3	58.1	69.1	87.0
BS=64							
MeZO	64.0	90.5	45.5	60.5	58.7	68.5	85.0
MeZO-LoRA	63.9	91.3	43.0	59.0	64.0	69.7	87.4
AdaZeta	64.3	91.5	49.6	60.7	68.1	68.7	86.5

Table 3.5: Comparative analysis of various ZO fine-tuning methods on the Roberta-Large models.

LLaMA-2-7B. Enabling the full AdaZeta framework—TT adapters plus adaptive query schedule—yields the strongest results, which are shown in Table. 3.6. AdaZeta outperforms all three ZO baselines (MeZO, MeZO-LoRA, Sparse-MeZO) on 8 of 10 tasks. Notably, AdaZeta surpasses first-order fine-tuning with AdamW on three tasks: RTE (74.0 vs. 61.7), CB (75.0 vs. 66.1), and COPA (94.0 vs. 86.0). This counter-intuitive

Methods	RTE	CB	BoolQ	WSC	WIC	SST2	MultiRC	COPA	ReCoRD	SQuAD
FT	61.7	66.1	84.6	63.4	65.9	94.0	45.4	86.0	81.1	90.7
LoRA	85.5	67.8	84.8	62.5	73.9	94.8	85.0	81.0	79.4	90.5
Zero-Shot	49.5	32.1	65.1	36.5	50.6	79.7	55.8	59.7	80.9	54.7
ICL	54.5	58.9	67.4	65.4	52.7	81.2	58.7	84.4	80.1	67.1
MeZO	54.6	73.0	68.6	52.8	57.8	85.8	62.6	86.0	70.8	72.5
MeZO-LoRA	59.6	74.0	71.6	53.0	55.2	86.8	67.2	89.0	72.0	80.0
Sparse-MeZO	58.6	76.0	67.8	53.0	56.8	85.2	61.2	86.0	70.6	64.4
AdaZeta	74.0	75.0	79.4	52.2	58.0	91.0	68.2	94.0	71.2	80.0

Table 3.6: Comparative analysis of various ZO fine-tuning methods on the Llama-2-7B model.

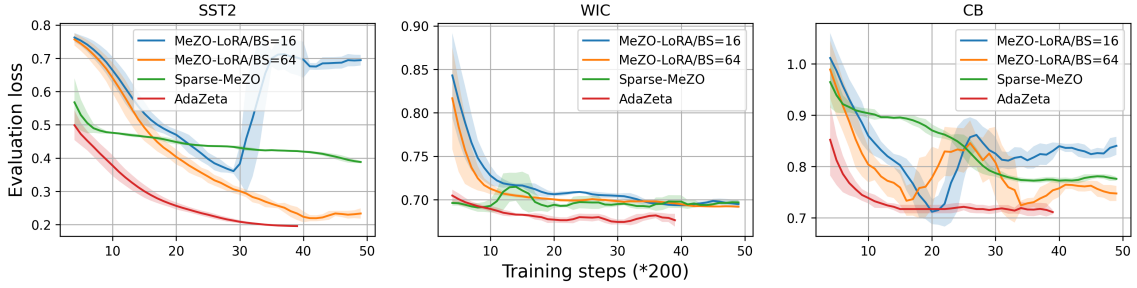


Figure 3.5: Training convergence curves on SST-2, WiC, CB, and MultiRC for LLaMA-2-7B. AdaZeta converges faster and to a lower loss than MeZO-LoRA and Sparse-MeZO across all tasks. Figure reproduced from [55].

result is explained by the regularization effect of the small parameter space: the TT adapter’s low dimensionality acts as an implicit constraint that prevents overfitting on the 1,000-example training set, where full fine-tuning and LoRA are prone to memorizing training patterns.

Convergence speed. The adaptive query schedule resolves all divergence cases observed with MeZO-LoRA across the 10 LLaMA-2-7B tasks. On the CB task, MeZO-LoRA requires approximately 6,000 gradient steps to reach a training loss of 0.4, whereas AdaZeta reaches the same loss in under 1,000 steps—a $6\times$ speedup at the same learning rate. The convergence curves across all four tracked tasks (SST2, WiC, CB, MultiRC) are shown in Figure 3.5.

Method	SST2	WiC	CB	MultiRC
MeZO-LoRA (BS=64)	3.0	4.8	8.6	30.0
MeZO-LoRA (BS=16)	0.6	1.1	3.1	10.8
Sparse-MeZO	4.1	3.6	4.3	6.4
AdaZeta	1.1	1.0	0.9	12.1

Table 3.7: GPU-hours required to reach an evaluation loss threshold on LLaMA-2-7B across four tasks. AdaZeta is competitive with or faster than all ZO baselines.

3.6.3 Memory Efficiency Analysis

AdaZeta requires only **14 GB** of peak GPU memory to fine-tune LLaMA-2-7B on the SST2 task—an $8\times$ **reduction** relative to full fine-tuning with AdamW, which requires approximately 112 GB. This memory reduction arises from two sources: the elimination of the backpropagation graph (inherited from the MeZO framework), and the minimal storage footprint of the TT adapter parameters.

Compared with other ZO fine-tuning methods, AdaZeta achieves similar or lower memory usage while delivering consistently better accuracy. The traditional approach of increasing the batch size to reduce gradient noise does not address the problem efficiently: MeZO-LoRA at batch size 64 requires 23 GB of memory— $1.6\times$ more than AdaZeta at batch size 16—and still underperforms AdaZeta on most tasks due to divergence. The accuracy-versus-memory trade-off is visualized in Figure 3.6, which shows that AdaZeta achieves the best accuracy among memory-efficient fine-tuning methods at equivalent memory budgets.

In terms of convergence time, Table 3.6.3 reports the GPU-hours required to reach a fixed evaluation loss threshold across four tasks. AdaZeta converges on-par with or faster than all ZO baselines, demonstrating that the additional forward passes from $Q_k > 1$ queries are more than compensated by the faster per-epoch progress.

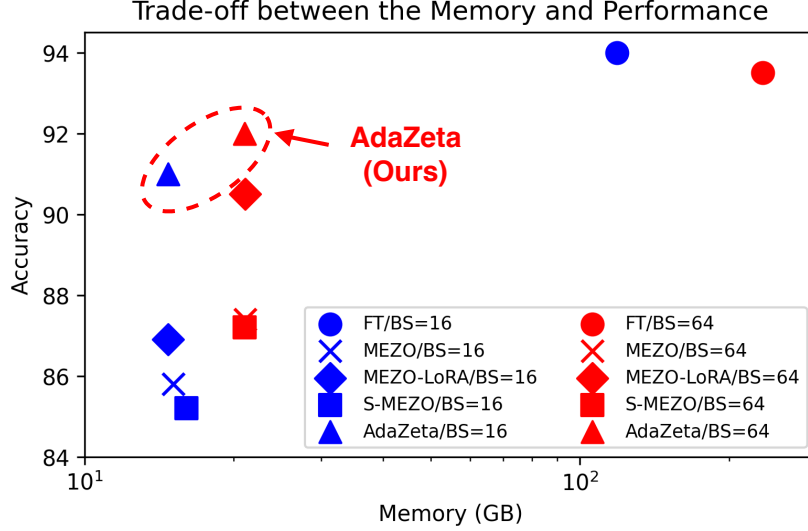


Figure 3.6: Accuracy vs. peak GPU memory trade-off for LLaMA-2-7B fine-tuning methods. AdaZeta achieves the best accuracy among memory-efficient (≤ 20 GB) methods. Figure reproduced from [55].

3.7 Conclusion

This chapter has demonstrated that tensor-train decomposition is a unifying tool for achieving both parameter efficiency and memory efficiency in LLM fine-tuning. LoRETTA shows that compressing PEFT adapter and reparameterization modules into TT format reduces trainable parameter counts to sub-1 MB levels while maintaining competitive or superior accuracy on both BERT-scale and LLaMA-2-scale models. The resulting parameter efficiency is not merely a compression trick: the small parameter space provides genuine regularization benefits, reducing overfitting and improving multi-task knowledge retention.

AdaZeta extends this TT-based compression into the backpropagation-free setting. By combining TT adapters with a fast-forward parallel contraction strategy and an adaptive query number schedule, AdaZeta resolves the divergence and variance problems that limit existing ZO methods at LLaMA-2-7B scale. The theoretical convergence analysis

provides a rigorous explanation for these empirical gains, showing that the benefits of dimensionality reduction and adaptive query growth compound multiplicatively in the convergence bound.

Together, LoRETTA and AdaZeta establish that the extreme parameter efficiency of TT decomposition directly enables extreme memory efficiency in the ZO setting—a connection that is both practically important and theoretically grounded. The next chapter extends the ZO paradigm to a qualitatively different setting: vision-language model prompt tuning under black-box inference constraints, where neither the model weights nor the gradients are accessible, and the optimization challenge is further complicated by the rugged loss landscape of multimodal objectives.

Chapter 4

Sharpness-Aware Zeroth-Order Optimization

4.1 Introduction

The previous chapter demonstrated that reducing the dimensionality of trainable parameters—through tensor-train decomposition—directly improves zeroth-order (ZO) fine-tuning of large language models. This chapter addresses a complementary challenge that arises when the trainable parameter space is already small by design: how to fine-tune vision-language models (VLMs) reliably under black-box, backpropagation-free constraints, where neither model weights nor gradients are directly accessible.

VLMs such as CLIP [31] have achieved remarkable performance on image classification, object detection, and image segmentation through large-scale pretraining on image-text pairs. Adapting these models to downstream tasks via prompt tuning [73, 74]—learning a small set of continuous tokens prepended to the model’s visual or textual input—has become a standard and highly parameter-efficient fine-tuning strategy. However, even prompt tuning requires gradient computation through the VLM backbone

via backpropagation, making it unsuitable for deployment on memory-constrained edge devices (e.g., wearable sensors, IoT devices) that support only inference-mode forward passes. This gap between the parameter efficiency of prompt tuning and the memory requirements of backpropagation motivates the development of fully backpropagation-free (BP-free) fine-tuning methods.

Prior BP-free approaches fall into two families. Evolutionary strategies (ES) such as BBT and BBTv2 [32, 75] employ Covariance Matrix Adaptation Evolution Strategy (CMA-ES) to search over the prompt parameter space using only fitness evaluations. These methods perform effective global exploration but suffer from premature convergence: the CMA-ES covariance matrix tends to collapse early, trapping optimization in sharp, narrow regions of the loss landscape. ZO gradient estimation approaches such as BlackVIP [33] and ZIP [34] avoid this collapse by performing local gradient-based descent using forward-difference approximations. However, standard ZO estimators are unbiased only with respect to a smoothed version of the objective [76], and their gradient estimates exhibit high variance in regions of sharp curvature—precisely the regions where ES methods also fail. Variance reduction through sparse perturbation (as in ZIP) addresses part of this problem but leaves a deeper issue unresolved: the loss landscape encountered at a random initialization of the visual prompt is typically sharp, and neither family of methods attempts to systematically navigate toward flatter regions before beginning local refinement.

This chapter proposes **SharpZO**, a hybrid sharpness-aware zeroth-order optimization framework for BP-free VLM prompt tuning [56]. SharpZO is motivated by a key insight from the sharpness-aware minimization (SAM) literature [36]: minimizing the worst-case loss within a neighborhood around the current parameters encourages convergence to flat minima that generalize better and present a smoother curvature profile. SharpZO adapts this insight to the BP-free setting through a two-stage approach. In the first

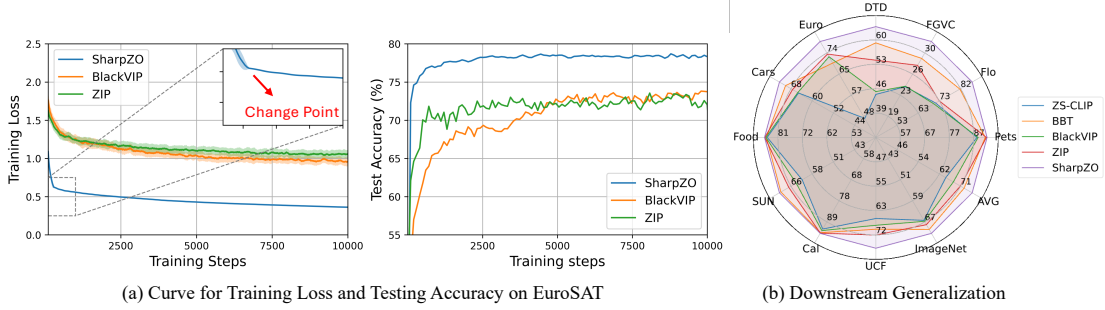


Figure 4.1: (a) Training variance comparison: SharpZO exhibits substantially narrower standard-deviation bands than ZIP and BlackVIP across training steps on ViT-B/16. (b) Per-task fine-tuned accuracy across 11 CLIP datasets; SharpZO consistently outperforms all BP-free baselines.

stage, SharpZO employs a sharpness-aware CMA-ES that centers its population sampling on the worst-case perturbation direction—estimated without backpropagation via coordinate-wise gradient estimation (CGE)—thereby steering the global search toward flat loss basins. In the second stage, SharpZO performs fine-grained local refinement using sparse ZO-SGD with a novel Z-pruning technique that applies Z-score normalized gradient sensitivity to select the most informative dimensions for perturbation, suppressing the outlier gradient estimates that are common in ZO settings. The entire pipeline requires only forward passes at every step.

The main contributions of this chapter are as follows:

- **Stage 1 — Sharpness-Aware CMA-ES.** We propose a modified CMA-ES population sampling strategy centered on the worst-case perturbation ϵ^* , estimated via CGE. This drives global optimization toward flat loss regions, providing a better initialization for Stage 2 ZO-SGD and reducing the effective curvature encountered during local refinement.
- **Stage 2 — Sparse ZO-SGD with Z-Pruning.** We introduce Z-pruning, a pruning criterion that combines weight magnitude and Z-score normalized gradient sensitivity to construct a binary mask Ω applied to the ZO perturbation vector.

Unlike magnitude-only pruning, Z-pruning is robust to the heavy-tailed outliers in ZO gradient estimates.

- **Empirical results.** SharpZO achieves an average accuracy of 75.64% on ViT-B/16 across 11 CLIP few-shot tasks, surpassing the state-of-the-art BP-free baseline ZIP by **4.7%** (and up to 7% on individual tasks). SharpZO also achieves faster convergence: 15.3 minutes to reach a common accuracy threshold on ImageNet versus 19.0 minutes for ZIP, using a single NVIDIA A100 GPU.

4.2 SharpZO: A Hybrid Backpropagation-Free Optimization Framework

4.2.1 Problem Setup

We consider the BP-free prompt tuning of a VLM with a frozen backbone. Following standard prompt tuning practice [73], for a model with hidden dimension m and K classes, the class-specific prompt is $\mathbf{p}^k = [\mathbf{p}_0, \mathbf{c}^k]$, where $\mathbf{p}_0 \in \mathbb{R}^m$ is a learnable initial embedding and $\mathbf{c}^k$ is the class label embedding. To make $\mathbf{p}_0$ trainable under BP-free constraints, we introduce a low-dimensional parameter $\mathbf{w} \in \mathbb{R}^d$ with $d \ll m$, and produce the embedding update via a fixed random projection matrix $\mathbf{A} \in \mathbb{R}^{m \times d}$:

$$\bar{\mathbf{p}} = \mathbf{w}\mathbf{A}^\top \in \mathbb{R}^m. \quad (4.1)$$

The trainable prompt is then $\mathbf{p}_0 + \bar{\mathbf{p}}$, and the optimization problem becomes

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{N} \sum_{n=1}^N \mathcal{L}([\mathbf{p}_0 + \mathbf{w}\mathbf{A}^\top, \mathbf{c}^k], (x_n, y_n)), \quad (4.2)$$

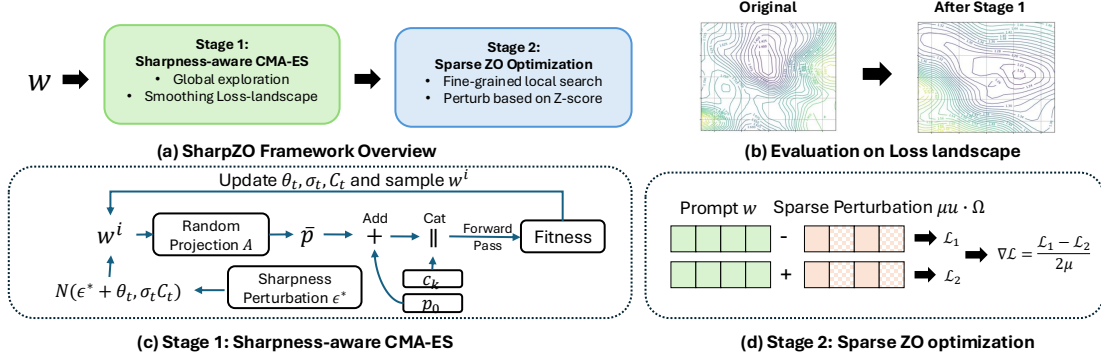


Figure 4.2: Overview of the SharpZO framework. (a) Two-stage optimization pipeline. (b) Smoothed loss landscape after Stage 1 sharpness-aware CMA-ES. (c) CMA-ES training dynamics. (d) RGE-based sparse ZO gradient estimation in Stage 2.

where $\mathcal{L}$ is the cross-entropy loss and (x_n, y_n) are image-label pairs. All parameters of the VLM backbone remain frozen; only w is updated. In practice, $d = 512$ for a CLIP prompt of length 4 with hidden dimension $m = 512$, which is far smaller than the number of model parameters ($\sim 86M$ for ViT-B/16). The ZO gradient notation from Chapter 2 applies directly: CGE and RGE are as defined in Section 2.2.1.

4.2.2 Two-Stage Optimization Pipeline

SharpZO decomposes optimization into two sequential stages. Stage 1 uses a sharpness-aware variant of CMA-ES to navigate away from sharp, high-curvature regions near random initialization and land in a flat basin. Stage 2 then performs fine-grained local refinement using sparse ZO-SGD, exploiting the low-variance gradient estimates afforded by the flat initialization. Both stages operate without backpropagation; the transition between them is determined automatically by a validation-based early-stopping criterion.

Stage 1: Sharpness-Aware CMA-ES

CMA-ES is a derivative-free optimizer that maintains a multivariate normal search distribution over the parameter space [35]. At iteration t , the algorithm maintains a distribution mean $\boldsymbol{\theta}_t \in \mathbb{R}^d$, a global step size $\sigma_t \in \mathbb{R}_{>0}$, and a covariance matrix $\mathbf{C}_t \in \mathbb{R}^{d \times d}$. A population of S candidate solutions is drawn as

$$\mathbf{w}_t^i \sim \boldsymbol{\theta}_t + \sigma_t \mathcal{N}(0, \mathbf{C}_t), \quad i = 1, \dots, S, \quad (4.3)$$

and their loss values $\mathcal{L}(\mathbf{w}_t^i)$ are evaluated via forward passes. The top- μ candidates (those with lowest loss) are used to update $\boldsymbol{\theta}_t$, σ_t , and $\mathbf{C}_t$ via weighted recombination and cumulative step-size adaptation. The covariance update captures correlations among successful search directions, enabling CMA-ES to adapt to the geometry of the loss surface.

CMA-ES enables effective global exploration: because it maintains a full covariance matrix, it can discover anisotropic search directions that purely local gradient methods miss. However, in VLM prompt tuning, the loss landscape tends to be rugged near random initialization. In sharp regions (large maximum Hessian eigenvalue), candidate evaluations vary widely across the population even for small step sizes, making elite selection noisy and causing the covariance matrix to adapt prematurely toward unrepresentative directions. This premature convergence is the dominant failure mode observed in BBT and BBTv2 [32, 75].

The key modification in Stage 1 is to shift the center of the CMA-ES sampling distribution toward the worst-case perturbation direction, following the SAM formulation

[?]. Specifically, we compute the worst-case perturbation within an ℓ_2 -ball of radius ρ :

$$\boldsymbol{\varepsilon}_t^* = \rho \frac{\nabla \mathcal{L}(\boldsymbol{\theta}_t)}{\|\nabla \mathcal{L}(\boldsymbol{\theta}_t)\|_2}, \quad (4.4)$$

where the gradient $\nabla \mathcal{L}(\boldsymbol{\theta}_t)$ is estimated using CGE (no backpropagation required), and the population is then drawn as

$$\boldsymbol{w}_t^i \sim \boldsymbol{\varepsilon}_t^* + \boldsymbol{\theta}_t + \delta_t \mathcal{N}(0, \boldsymbol{C}_t), \quad i \in [1, S]. \quad (4.5)$$

This modification can be understood via a Taylor expansion of the expected population fitness. Since the population is centered at $\boldsymbol{\theta}_t + \boldsymbol{\varepsilon}_t^*$ and drawn with noise $o \sim \mathcal{N}(0, \delta_t^2 \boldsymbol{C}_t)$ satisfying $\mathbb{E}[o] = 0$, we have:

$$\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_t + \boldsymbol{\varepsilon}_t^* + o)] \approx \mathcal{L}(\boldsymbol{\theta}_t + \boldsymbol{\varepsilon}_t^*) + \frac{1}{2} \mathbb{E}[o^\top \nabla^2 \mathcal{L}(\boldsymbol{\theta}_t + \boldsymbol{\varepsilon}_t^*) o]. \quad (4.6)$$

The first term, $\mathcal{L}(\boldsymbol{\theta}_t + \boldsymbol{\varepsilon}_t^*)$, is the same worst-case objective minimized by gradient-based SAM. The second term, involving the Hessian at the perturbed point, introduces an additional mechanism through adaptive covariance updates: the CMA-ES covariance naturally down-weights high-curvature directions because candidate solutions in those directions receive poor fitness scores, causing the covariance to contract along dimensions of large curvature. Together, these two effects bias the Stage 1 search toward parameter regions with small Hessian eigenvalues—precisely the flat minima where ZO gradient estimates are most accurate and variance is lowest.

A practical challenge is computing $\boldsymbol{\varepsilon}_t^*$ without backpropagation. We use CGE for this purpose (Section 2.2.1): CGE requires $\mathcal{O}(d)$ forward passes but provides an unbiased estimate of $\nabla \mathcal{L}(\boldsymbol{\theta}_t)$ along each coordinate, without the smoothing bias introduced by RGE. Because $d = 512$ in our prompt tuning setting, the $\mathcal{O}(d)$ query cost of CGE is

modest (512 forward passes per sharpness-aware gradient computation). This stands in contrast to the LLM ZO setting of Chapter 3, where d can be in the millions; the small d of prompt tuning makes CGE tractable for sharpness estimation while keeping the total computation within the BP-free budget.

The transition from Stage 1 to Stage 2 is determined automatically: the algorithm switches when the validation accuracy does not improve by more than a threshold over ten consecutive steps, following an early-stopping criterion. Typically, Stage 1 terminates within $T_c \leq 500$ steps, which is a small fraction of the total training budget T .

Stage 2: Sparse Zeroth-Order SGD

After Stage 1, the parameter $\mathbf{w}$ resides near a flat region of the loss landscape. Stage 2 performs fine-grained local refinement using ZO stochastic gradient descent (ZO-SGD) with a sparse perturbation vector. Sparsity is beneficial because it reduces the effective dimensionality of the gradient estimate, directly reducing variance: for a perturbation restricted to s of the d dimensions, the variance of the RGE estimator scales as $\mathcal{O}(s/d)$ relative to the dense case.

Rather than perturbing all d dimensions of $\mathbf{w}$, Stage 2 applies a binary pruning mask $\Omega \in \{0, 1\}^d$ to the random perturbation vector $\mathbf{u} \sim \mathcal{N}(0, \mathbf{I})$. The sparse ZO gradient estimator is:

$$\hat{\nabla}_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \frac{\mathcal{L}(\mathbf{w} + \mu \Omega \mathbf{u}) - \mathcal{L}(\mathbf{w} - \mu \Omega \mathbf{u})}{2\mu} \mathbf{u}, \quad (4.7)$$

where $\mu > 0$ is the smoothing parameter. The mask Ω is fixed for K consecutive training steps and then recomputed; this amortizes the CGE-based computation of Ω over multiple gradient steps.

A single query ($q = 1$) in Stage 2 is sufficient for stable training because the flat initial-

ization provided by Stage 1 ensures that the loss surface around $\mathbf{w}$ has small curvature, making the RGE estimator accurate even at $q = 1$. This contrasts with prior methods such as ZIP and BlackVIP, which require $q = 5$ or more queries per step to achieve comparable stability and still exhibit higher training variance, as shown in Figure 4.1.

4.2.3 Z-Pruning

The choice of pruning mask Ω is critical: pruning uninformative dimensions reduces variance, but pruning informative ones slows convergence. Prior sparse ZO methods [28] use the magnitude $|\mathbf{w}|^2$ as a proxy for importance. While magnitude captures the current activation strength of each prompt dimension, it ignores the sensitivity of the loss to perturbations in that dimension—information that is more directly relevant to gradient estimation.

SharpZO introduces Z-pruning, a pruning criterion that combines weight magnitude with gradient sensitivity estimated via CGE. The derivation starts from a first-order Taylor approximation of the loss change under pruning: $\mathcal{L}(\mathbf{w} + \delta\mathbf{w}) - \mathcal{L}(\mathbf{w}) \approx \nabla\mathcal{L}(\mathbf{w})^\top\delta\mathbf{w} + \frac{1}{2}\delta\mathbf{w}^\top H\delta\mathbf{w}$, where H is the Hessian. Approximating H by the Fisher information matrix $\mathbb{E}_{x \sim \mathcal{D}}[\hat{\nabla}\mathcal{L}(\mathbf{w}; x)^2]$ and using the CGE gradient estimate, the second-order saliency criterion yields:

$$\Omega = |\mathbf{w}|^2 \cdot z\left(\mathbb{E}_{x \sim \mathcal{D}}\left[\hat{\nabla}\mathcal{L}(\mathbf{w}, x)^2\right]\right), \quad (4.8)$$

where $z(\mathbf{g}) = (\mathbf{g} - \mu_g)/\sigma_g$ denotes element-wise Z-score normalization with mean μ_g and standard deviation σ_g computed over all d coordinates of the gradient vector. The top- $k\%$ dimensions by Ω score are retained (mask set to 1); the rest are zeroed.

The Z-score normalization plays a crucial role in ZO settings. ZO gradient estimates in high-curvature directions can produce extreme outlier values that would dominate a

Algorithm 2 SharpZO: Hybrid Sharpness-Aware Zeroth-Order Optimization

Require: Initial prompt parameters $\mathbf{w}_0$, total steps T , transition step T_c , sharpness radius ρ , smoothing μ , mask interval K , learning rate η

for $t = 1$ **to** T **do**

if $t < T_c$ **then**

 // **Stage 1: Sharpness-Aware CMA-ES**

 Estimate $\nabla \mathcal{L}(\boldsymbol{\theta}_t)$ via CGE (eq. 2.2.1)

 Compute $\boldsymbol{\varepsilon}_t^* \leftarrow \rho \nabla \mathcal{L}(\boldsymbol{\theta}_t) / \|\nabla \mathcal{L}(\boldsymbol{\theta}_t)\|$

 Sample candidates $\mathbf{w}_t^i \sim \boldsymbol{\varepsilon}_t^* + \boldsymbol{\theta}_t + \delta_t \mathcal{N}(0, \mathbf{C}_t)$, $i = 1, \dots, S$

 Evaluate fitness $\mathcal{L}(\mathbf{w}_t^i)$ for each candidate via forward pass

 Update CMA-ES parameters $\boldsymbol{\theta}_{t+1}$, σ_{t+1} , $\mathbf{C}_{t+1}$ from top- μ elites

else

 // **Stage 2: Sparse ZO-SGD**

if $t = T_c$ **or** $(t - T_c) \bmod K = 0$ **then**

 Recompute pruning mask Ω via Z-pruning (Eq. 4.8)

end if

 Sample $\mathbf{u} \sim \mathcal{N}(0, \mathbf{I})$; estimate $\hat{\nabla} \mathcal{L}(\mathbf{w}_t)$ via Eq. 4.7

$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \hat{\nabla} \mathcal{L}(\mathbf{w}_t)$

end if

end for

return Fine-tuned prompt vector $\mathbf{w}_T$

simple magnitude product $|\mathbf{w}|^2 \cdot \mathbb{E}[\hat{\nabla} \mathcal{L}^2]$, causing the mask to over-select a few unstable dimensions. Z-score standardization is more robust to such outliers than softmax or min-max normalization: it preserves the relative ordering of gradient magnitudes while suppressing extreme values, yielding a pruning mask that reflects the true distribution of gradient sensitivity across prompt dimensions. This is especially important in Stage 2, where the model is already near a flat optimum and the gradient landscape is relatively uniform—small estimation errors in the mask can otherwise persist over many steps.

4.2.4 Algorithm

Algorithm 2 presents the complete SharpZO procedure. Stage 1 runs for T_c steps of sharpness-aware CMA-ES; Stage 2 runs for the remaining $T - T_c$ steps of sparse ZO-SGD with Z-pruning, refreshing the mask every K steps.

4.3 Experimental Results

4.3.1 Experimental Setup

Models and tasks. We evaluate SharpZO on CLIP [31] with two visual encoder backbones: ResNet-50 (RN50) and ViT-B/16. The text encoder is a standard Transformer [77]. Evaluations span 11 image classification datasets: ImageNet, Caltech101, OxfordPets, Flowers102, FGVC Aircraft, Food101, Stanford Cars, SUN397, Describable Textures (DTD), EuroSAT, and UCF101. All experiments use a 16-shot training set. Distribution shift robustness is evaluated on four ImageNet variants: ImageNet-V2, -Sketch, -A, and -R. All model backbone weights are initialized from official pretrained CLIP checkpoints and remain frozen throughout training.

Baselines. We compare against four BP-free prompt tuning methods: Zero-Shot CLIP (ZS-CLIP), BBT [32], BlackVIP [33], and ZIP [34]. For context we also report CoOp [73], a first-order gradient-based prompt tuning method that uses backpropagation, serving as an upper bound. All BP-free baselines are trained for 20K steps.

Implementation. The trainable prompt has length 4 with hidden dimension $d = 512$ for both backbones. Parameters in $\mathbf{w}$ are initialized from $\mathcal{N}(0, 0.02)$. The sharpness radius is $\rho = 0.05$; smoothing parameter $\mu = 0.001$; mask refresh interval $K = 50$; sparsity rate 50%. Stage 1 typically transitions before 500 steps. All experiments are conducted on a single NVIDIA A100-40G GPU. No backpropagation is performed at any point during SharpZO training.

4.3.2 Results on Zeroth-Order Fine-Tuning

Few-shot classification. Table results for 11-task few-shot classification are summarized in Figure 4.2. SharpZO achieves an average accuracy of **75.64%** on ViT-B/16

and **69.76%** on RN50 across all 11 tasks, outperforming the strongest BP-free baseline, ZIP, by **4.7%** and **5.7%** respectively. Gains are consistent across all 11 tasks on ViT-B/16: SharpZO surpasses ZIP on every single task, with the largest improvements on fine-grained datasets such as Flowers102 (+15.0%), EuroSAT (+5.2%), DTD (+9.7%), and FGVC Aircraft (+4.0%). These datasets present particularly rugged loss landscapes due to high inter-class similarity, which amplifies the benefit of sharpness-aware initialization. Across both backbones, SharpZO’s performance closely approaches that of CoOp, the backpropagation-based upper bound, achieving 94–97% of CoOp accuracy without any gradient computation through the VLM.

Backbone	Methods	ImageNet	Pets	Flo	FGVC	DTD	Euro	Cars	Food	SUN	Cal	UCF	AVG
RN50	ZS-CLIP*	58.18	85.77	66.14	17.28	42.32	37.56	55.61	77.31	58.52	86.29	61.46	58.77
	BBT*	61.74	88.73	72.53	12.07	54.33	69.01	60.24	78.44	64.34	90.05	67.91	65.40
	BlackVIP	60.33	85.99	65.12	17.37	42.73	58.16	56.70	77.23	59.17	86.37	60.11	60.84
	ZIP	61.30	89.53	68.41	19.98	47.40	63.10	58.61	78.98	62.86	90.63	64.05	64.08
	SharpZO	63.29	89.51	79.50	23.97	60.58	80.77	60.58	79.28	66.17	91.24	72.43	69.76
ViT-B/16	ZS-CLIP*	66.73	89.21	71.34	24.72	44.39	47.60	65.32	86.06	62.50	92.94	66.75	65.23
	BBT*	70.15	92.70	82.41	29.49	59.26	70.48	70.19	86.42	70.33	94.75	70.48	72.42
	BlackVIP*	67.10	89.70	70.60	24.78	45.20	73.10	65.60	86.60	64.70	93.70	69.10	68.20
	ZIP (Official)*	66.20	94.00	70.40	26.80	47.80	64.60	71.09	86.40	63.30	94.00	69.80	70.57
	ZIP (Rep)	68.35	93.18	73.00	28.32	54.26	74.19	67.58	87.01	67.43	94.97	72.51	70.98
	SharpZO	71.60	94.06	88.02	32.34	63.95	79.42	72.50	87.13	70.86	95.09	77.08	75.64

Table 4.1: Few-shot performance across 11 datasets using CLIP models with ResNet and ViT backbones, trained for 20K steps. * indicate results reported in prior works [33, 78, 34]. We additionally reproduce the ZIP results, as the original paper restricted the query budget to 5K. Bold values highlight the best performance, demonstrating the superiority of SharpZO over all BP-free baselines.

Distribution shift robustness. Table 4.2 reports accuracy on four ImageNet distribution shift benchmarks. SharpZO achieves an average accuracy of **43.3%** on RN50 and **58.9%** on ViT-B/16 across the four shift variants, compared to ZIP’s 42.2% and 56.9% respectively—an improvement of 1.1% and 2.0%. The flat minima found by Stage 1 generalize better out-of-distribution, consistent with the theoretical and empirical findings from the SAM literature [?]: parameters at flat loss regions are less sensitive to covariate shift because the loss surface remains relatively stable under small changes to the input distribution. SharpZO is notably better than BlackVIP on ViT-B/16 (+4.0%), which

Method	ResNet-50						ViT-B/16					
	IN	-V2	-Sk	-A	-R	Avg	IN	-V2	-Sk	-A	-R	Avg
ZS-CLIP	58.2	51.3	33.3	21.7	56.0	40.6	66.7	60.8	46.2	47.8	74.0	57.2
BBT	61.7	54.0	33.9	23.2	58.3	42.4	70.2	63.0	47.9	49.5	76.1	59.1
BlackVIP	60.2	52.3	33.3	21.5	57.7	41.2	65.5	59.2	44.6	42.5	73.1	54.9
ZIP	61.3	53.7	33.7	23.9	57.6	42.2	68.4	59.7	45.5	47.1	75.2	56.9
SharpZO	63.3	54.8	35.2	24.5	58.7	43.3	71.6	63.8	45.0	50.3	76.6	58.9

Table 4.2: Distribution shift robustness. SharpZO vs. BP-free baselines on ImageNet and four shifted variants. Best BP-free results in **bold**.

Table 4.3: Time-to-test-accuracy (minutes) on ViT-B/16 across four datasets. Lower is better.

Method	ImageNet	Pets	DTD	EuroSAT
BlackVIP	172.6	714.8	132.0	201.5
ZIP	19.0	126.3	6.0	251.4
SharpZO	15.3	2.4	2.6	12.7

further validates the benefit of sharpness-aware initialization over pure coordinate-wise ZO without landscape-aware warm-up.

Training efficiency. Table 4.3 reports time-to-test-accuracy (in minutes) across four datasets using a ViT-B/16 backbone. SharpZO reaches the accuracy threshold in **15.3 minutes** on ImageNet, compared to 19.0 minutes for ZIP and 172.6 minutes for BlackVIP—a 20% speedup over ZIP and an $11\times$ speedup over BlackVIP. On three of the four datasets (ImageNet, Pets, DTD), SharpZO is substantially faster than all baselines. The efficiency gain comes from two sources: (1) each Stage 2 ZO step requires only $q = 1$ query rather than $q = 5$ as in ZIP, because the flat initialization makes single-query estimates sufficiently accurate; and (2) SharpZO’s prompt representation requires only a simple matrix multiply for forward pass (0.007 seconds per step), compared to ZIP’s complex reconstruction procedure (0.53 seconds per step).

Ablation studies. Figure 4.3 presents the ablation for Stage 1, comparing standard CMA-ES with sharpness-aware (S-aware) CMA-ES on EuroSAT. The sharpness-aware variant achieves faster Stage 1 convergence and a notably more stable Stage 2 training curve with reduced variance—consistent with the theoretical prediction that Stage 1 clips

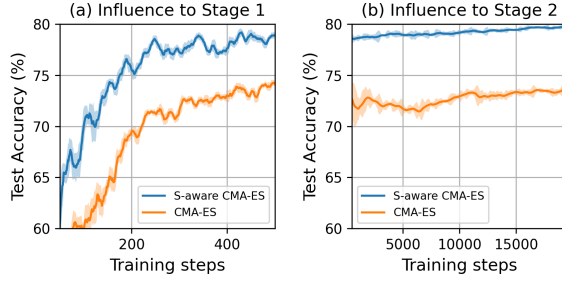


Figure 4.3: Stage 1 ablation on EuroSAT: standard CMA-ES vs. sharpness-aware (S-aware) CMA-ES for both Stage 1 and Stage 2 training curves.

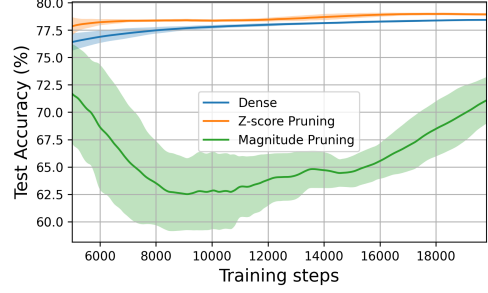


Figure 4.4: Stage 2 ablation on EuroSAT: dense ZO vs. magnitude-based sparse ZO vs. Z-pruning. Z-pruning achieves lower variance and faster convergence.

the effective smoothness constant. Figure 4.4 presents the Stage 2 ablation, comparing dense ZO, magnitude-based sparse ZO, and Z-pruning on EuroSAT. Z-pruning reduces gradient variance and improves both accuracy and convergence speed over the dense baseline, while magnitude-only pruning performs poorly in this setting: with only 512 trainable parameters, magnitude-based pruning discards dimensions with small weights that are nonetheless highly sensitive to loss—an error that the joint gradient-magnitude criterion in Z-pruning avoids.

4.4 Conclusion

This chapter introduced SharpZO, a hybrid BP-free VLM prompt tuning framework that addresses the fundamental challenge of high gradient estimation variance in ZO optimization by targeting the sharpness of the loss landscape. The central contribution is a two-stage architecture: Stage 1 sharpness-aware CMA-ES that steers global search toward flat loss basins via worst-case perturbation sampling, and Stage 2 sparse ZO-SGD with Z-pruning that efficiently exploits the flat neighborhood through robust, variance-reduced gradient estimation.

The theoretical convergence analysis makes the benefit of sharpness reduction precise: Stage 1 reduces the effective Lipschitz constant seen by Stage 2, yielding a logarithmic speedup in total convergence time. The empirical results across 11 CLIP classification benchmarks confirm this: SharpZO achieves 4.7–5.7% average accuracy improvements over the state-of-the-art BP-free baseline, with up to 7% gains on individual tasks, and converges 20% faster than ZIP in wall-clock time.

Together, SharpZO establishes loss landscape sharpness as a first-class optimization criterion for forward-only fine-tuning, complementing the dimensionality reduction approach of AdaZeta (Chapter 3). While Chapter 3 reduced gradient estimation variance by compressing the parameter space via tensor-train decomposition, SharpZO reduces it by improving the quality of the initialization through sharpness-aware warm-up. These two approaches address the same root challenge—high ZO gradient variance—from complementary angles: parameter compression and landscape smoothing. Besides these works, we also proposed other ZO methods involving quantization and multi-task learning in our QuZO [79] and MAZO [80] works.

The next chapter shifts focus from training-time efficiency to inference-time efficiency. Rather than reducing the cost of updating model parameters, Chapter 5 addresses the challenge of reducing the cost of deploying already-trained models through a novel regional gradient pruning strategy that achieves state-of-the-art accuracy-efficiency trade-offs at post-training sparsity levels.

Chapter 5

Accelerating LLM Inference via Pruning

5.1 Introduction

The preceding chapters addressed efficiency at training time: reducing the number of trainable parameters (LoRETTA), eliminating backpropagation (AdaZeta), and improving forward-only optimization via sharpness-aware initialization (SharpZO). This chapter shifts attention to inference-time efficiency. Once a large language model is deployed, every user request incurs memory bandwidth cost proportional to the model’s weight count and compute cost proportional to its parameter density. Pruning—the removal of structurally unimportant weights—is the most direct path to reducing both costs permanently, without any change to the inference software stack.

Post-training pruning of LLMs has attracted significant attention because it requires no retraining: given a calibration dataset of a few hundred samples, a pruning algorithm zeroes out weights according to a saliency criterion and the model can be immediately deployed. Two methods define the current state of the art. **Wanda** [11] computes a

simple per-weight saliency score $S_{ij} = |W_{ij}| \cdot \|X_j\|_2$, where $|W_{ij}|$ is the weight magnitude and $\|X_j\|_2$ is the ℓ_2 norm of the j -th input activation channel over calibration samples. Wanda is remarkably fast—55 seconds to prune a 7B model on a single GPU—but it treats each layer independently and ignores how weight removal propagates through the model’s non-linear computations. **GBLM** [39] addresses this by augmenting Wanda’s score with a full-model cross-entropy gradient G_{ij} , computing $S_{ij} = (\alpha G_{ij} + \|X_j\|_2) \cdot |W_{ij}|$. The gradient signal substantially improves pruning quality—perplexity drops from 7.26 to 7.15 on LLaMA-1-7B at 50% sparsity—but the cost is prohibitive: GBLM requires a full backward pass through the entire model per calibration sample, taking 5,801 seconds (97 minutes) for a 7B model and 49,663 seconds for a 65B model, with memory requirements that scale to 269 GB.

The core observation motivating this chapter is that the gradient quality matters more than the gradient scope. GBLM gains accuracy because its gradient captures how each weight influences the cross-entropy loss through the model’s full computational graph. But most of this benefit comes from the within-block non-linearities (attention operations, MLP activations) rather than from inter-block dependencies. This suggests a natural decomposition: compute gradients locally within each decoder block, using a block-level surrogate loss that requires only a single backward pass through that block. We call this the regional gradient and build on it to define two complementary techniques.

This chapter presents **Wanda++** [57], a post-training LLM pruning framework built on two innovations. First, the **Regional Gradient Score (RGS)** replaces Wanda’s activation-norm proxy with a gradient signal derived from a block-level ℓ_2 surrogate loss $\mathcal{L}_{\text{RGS}}^l = \|f^l(\mathbf{X}^l)\|_2$, where f^l is the l -th decoder block and $\mathbf{X}^l$ is its input. Gradients of this loss with respect to in-block weights are fully self-contained: no activations from other blocks are needed. Second, **Regional Optimization (RO)** performs a small number of block-local weight updates after pruning each block, minimizing the mean-

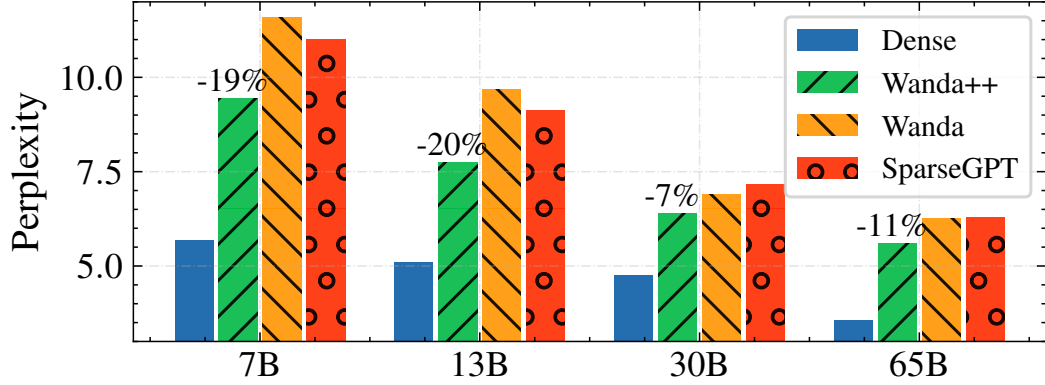


Figure 5.1: Perplexity improvement of Wanda++ over Wanda across LLaMA model sizes at 2:4 semi-structured sparsity. Regional gradient scoring and optimization consistently reduce the perplexity gap between pruned and dense models.

squared discrepancy between the dense and pruned block outputs $\|f^l(\mathbf{X}) - \hat{f}_k^l(\mathbf{X})\|^2$.

RO is fully orthogonal to subsequent sparsity-aware fine-tuning: applying LoRA after Wanda++ further reduces perplexity, and the gains compound rather than saturate.

The main contributions of this chapter are:

- **Regional Gradient Score (RGS):** A block-level ℓ_2 -norm gradient signal that captures within-block non-linearity without full-model backpropagation. RGS improves perplexity over Wanda by up to **32%** on OpenLLaMA-7B at 2:4 sparsity and matches GBLM quality at 40× lower computation cost.
- **Regional Optimization (RO):** An iterative prune-then-optimize procedure that minimizes block-output discrepancy using only local calibration samples. RO is orthogonal to LoRA fine-tuning and contributes complementary perplexity improvements.
- **Practical efficiency:** Wanda++ prunes a LLaMA-1-7B model in under 5 minutes on a single NVIDIA H100 GPU. At 2:4 semi-structured sparsity, the pruned model achieves 28% weight memory reduction and 33–47% time-to-first-token (TTFT) reduction on NVIDIA sparse tensor cores.

- **Breadth of evaluation:** Results across LLaMA-1 (7B–65B), OpenLLaMA (3B, 7B, 70B), and LLaMA-3.1 (8B) confirm that regional gradients generalize across model families and sparsity patterns (unstructured, 2:4, 4:8, structured row-wise).

5.2 Regional Gradient-Based Scoring

5.2.1 Prior Pruning Methods

We briefly formalize the two pruning baselines that bookend the accuracy-efficiency trade-off, as both directly inform the design of RGS.

Wanda [11] assigns each weight W_{ij} in a linear layer the saliency score:

$$S_{ij}^{\text{Wanda}} = |W_{ij}| \cdot \|X_j\|_2, \quad (5.1)$$

where $\|X_j\|_2 = \sqrt{\frac{1}{N} \sum_{n=1}^N (X_j^{(n)})^2}$ is the root-mean-square of the j -th input channel over N calibration samples. Intuitively, a weight with large magnitude connected to a frequently active input channel contributes substantially to the layer’s output, so it should be preserved. Wanda applies this criterion layer by layer with no inter-layer communication, making it embarrassingly parallel and very fast (55 s for LLaMA-1-7B on one GPU). However, the activation norm $\|X_j\|_2$ is purely a data-dependent input statistic: it captures input magnitude but ignores how the downstream loss depends on each weight through the model’s non-linear computations.

GBLM [39] addresses this limitation by incorporating gradient information:

$$S_{ij}^{\text{GBLM}} = (\alpha G_{ij} + \|X_j\|_2) \cdot |W_{ij}|, \quad G_{ij} = \sqrt{\frac{1}{N} \sum_{n=1}^N \left(\nabla_{W_{ij}} \mathcal{L}_{\text{CE}}(\mathbf{X}_n^{(0)}) \right)^2}, \quad (5.2)$$

where $\mathcal{L}_{\text{CE}}$ is the cross-entropy loss evaluated at the model’s output and $\alpha = 100$ is a

scaling factor. The gradient G_{ij} is the root-mean-square of the cross-entropy gradient over calibration samples. GBLM substantially improves pruning quality (WikiText-2 PPL 7.15 vs. 7.26 for Wanda on LLaMA-1-7B at 50% sparsity) precisely because the cross-entropy gradient propagates sensitivity information from the output layer all the way back to each weight. Unfortunately, this requires storing activations across all layers simultaneously and running a full backward pass per calibration sample—5,801 s and 26 GB for LLaMA-1-7B, scaling to 49,663 s and 269 GB for the 65B model. These costs make GBLM impractical for any model that does not fit in a data-center cluster with ample time budget.

5.2.2 Regional Gradient Score

The key design question is: can we obtain gradient information that is nearly as informative as GBLM but nearly as cheap as Wanda? Wanda++ answers yes by restricting gradient computation to individual decoder blocks.

Block-level surrogate loss. For decoder block l with function $f^l : \mathbb{R}^{B \times T \times m} \rightarrow \mathbb{R}^{B \times T \times m}$ (batch size B , sequence length T , hidden dimension m) and calibration inputs $\{\mathbf{X}_n^l\}_{n=1}^N$, we define:

$$\mathcal{L}_{\text{RGS}}^l(\mathbf{X}_n^l) = \|f^l(\mathbf{X}_n^l)\|_2. \quad (5.3)$$

This is the Frobenius norm of the block’s output. Computing gradients $\nabla_{W_{ij}} \mathcal{L}_{\text{RGS}}^l$ requires only one backward pass through block l with block-local activations—no gradients need to flow across block boundaries. The entire computation is self-contained.

Regional Gradient Score. The RMS gradient over calibration samples gives the

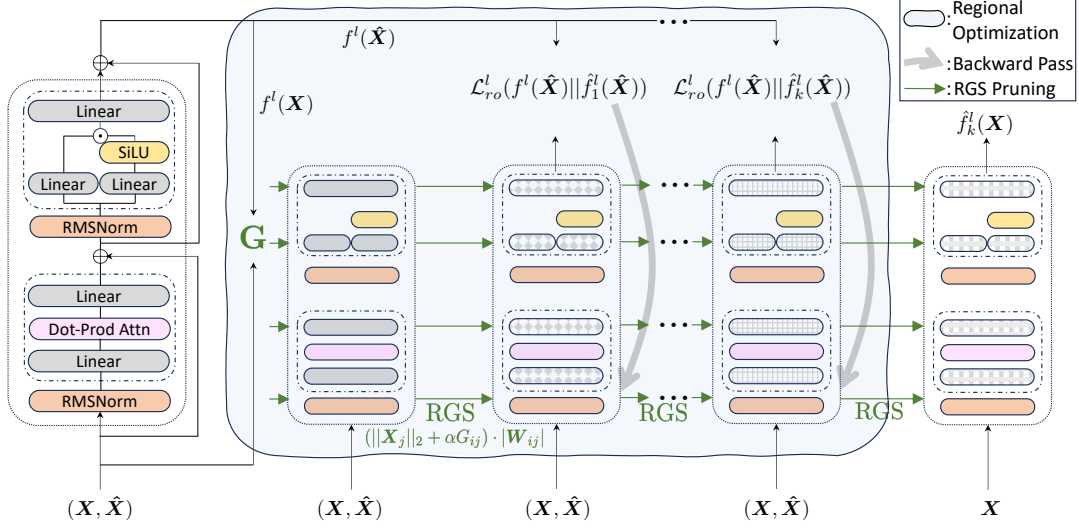


Figure 5.2: Illustration of Wanda++, which leverages regional gradients in two ways: first, we estimate the block-level gradient $\mathbf{G}$, used in Regional-Gradient Scoring (RGS) for layer-wise pruning in each block. Next, we calculate the pruning-induced loss $\mathcal{L}_{ro}$ for lightweight Regional Optimization (RO). RO can be iterated multiple times to infer better pruning masks for each decoder block.

per-weight gradient signal:

$$G_{ij} = \sqrt{\frac{1}{N} \sum_{n=1}^N (\nabla_{W_{ij}} \mathcal{L}_{\text{RGS}}^l(\mathbf{X}_n^l))^2}. \quad (5.4)$$

The final RGS pruning criterion blends this regional gradient with the layer-wise activation norm, following the same additive structure as GBLM:

$$S_{ij}^{\text{RGS}} = (\alpha G_{ij} + \|\mathbf{X}_j\|_2) \cdot |\mathbf{W}_{ij}|, \quad \alpha = 100. \quad (5.5)$$

Weights with the lowest S_{ij}^{RGS} within each row are zeroed to achieve the target sparsity pattern (unstructured, 2:4, 4:8, or row-wise structured).

Why ℓ_2 norm, not cross-entropy? Four properties make the ℓ_2 output norm superior to cross-entropy as a regional surrogate:

1. Self-contained computation. The ℓ_2 norm of block l 's output depends only on com-

putations within block l . No activations from other blocks, no label embeddings, and no output projection are needed. One backward per block suffices.

2. Noise reduction. Cross-entropy loss has label-dependent variance: its gradients fluctuate based on how confident the model is on each sample’s ground-truth label. The ℓ_2 output norm is purely data-driven and does not depend on labels, reducing the noise in the gradient estimate.
3. Captures non-linearity. Unlike Wanda’s activation norm $\|X_j\|_2$, which is computed on the input and therefore reflects a linear proxy for weight importance, the ℓ_2 output norm integrates the effects of attention softmax, layer normalization, and MLP activations within the block. The resulting gradient G_{ij} therefore reflects how each weight influences the block’s output through these non-linear operations.
4. Speed and memory. Computing G_{ij} via $\mathcal{L}_{\text{RGS}}^l$ for all L blocks costs L forward-backward pass pairs through individual blocks—roughly $2\times$ the wall-clock time of a Wanda forward pass. RGS-only runtime is 147 s for LLaMA-1-7B (compared to 55 s for Wanda and 5,801 s for GBLM), a $40\times$ speedup over GBLM with comparable pruning quality.

5.3 Regional Optimization

5.3.1 Motivation and Objective

Pruning with any saliency score—including RGS—inevitably introduces a residual output discrepancy: the output of the pruned block $\hat{f}_k^l$ differs from the output of the dense block f^l on the same inputs. SparseGPT [12] addresses this discrepancy at the weight level via an approximate Hessian update (derived from the Optimal Brain Surgeon

framework), correcting unmasked weights to compensate for the masked ones. This is effective but expensive: the Hessian inversion requires $\mathcal{O}(d^3)$ operations per layer.

Wanda++ takes a different approach: rather than correcting individual weights, we directly minimize the output discrepancy for each block via a small number of gradient steps on the in-block weights. Because this optimization is entirely local to block l —it only requires the block’s forward and backward pass—it inherits the scalability of RGS and adds only a modest runtime overhead.

5.3.2 Regional Optimization Method

At pruning iteration k for block l , RO minimizes the mean-squared error between the dense block output $f^l(\hat{\mathbf{X}}_m^l)$ and the current pruned block output $\hat{f}_k^l(\hat{\mathbf{X}}_m^l)$:

$$\mathcal{L}_{\text{RO}}^{l,k}(\hat{\mathbf{X}}_m^l) = \left\| f^l(\hat{\mathbf{X}}_m^l) - \hat{f}_k^l(\hat{\mathbf{X}}_m^l) \right\|^2, \quad (5.6)$$

where $\hat{\mathbf{X}}_m^l$ is a mini-batch of $M = 32$ calibration samples randomly drawn from the full calibration set of $N = 128$ samples. The dense reference $f^l(\hat{\mathbf{X}}_m^l)$ is computed once before pruning and cached. Optimization uses RMSprop with learning rate 3×10^{-7} .

Iterative prune-and-optimize. Wanda++ performs $K = 5$ iterations of alternating pruning and optimization per decoder block:

1. Prune with RGS to k/K of the target sparsity level.
2. Run one RO pass over M samples to reduce output discrepancy.
3. Re-compute the RGS gradient G_{ij} with the updated weights.
4. Prune to $(k + 1)/K$ of target sparsity; repeat until $k = K$.
5. Final RO pass after reaching full target sparsity.

Algorithm 3 Wanda++: Pruning with Regional Gradients

Require: Calibration inputs $\{\mathbf{X}^l\}_{l=1}^L$, scaling factor α , RO iterations K , RO batch size M

for $l = 1$ **to** L **do**

 // **Compute regional gradient**

 Forward-backward f^l on $\{\mathbf{X}_n^l\}$ using $\mathcal{L}_{\text{RGS}}^l$ (Eq. 5.3); accumulate G_{ij}

for $k = 1$ **to** K **do**

 Sample mini-batch $\hat{\mathbf{X}}_m^l \subset \mathbf{X}^l$, $|\hat{\mathbf{X}}_m^l| = M$

 Compute S_{ij}^{RGS} via Eq. 5.5; prune to sparsity level k/K

for each $\hat{\mathbf{X}}_m^l$ **do**

 Compute RO loss $\mathcal{L}_{\text{RO}}^{l,k}(\hat{\mathbf{X}}_m^l)$ (Eq. 5.6); update weights with RMSprop

end for

end for

 // **Final gradient refresh and prune**

 Re-compute G_{ij} with updated weights; apply final RGS pruning to target sparsity

end for

return Pruned model

This graduated approach avoids the instability of one-shot pruning to the final sparsity level: by incrementally introducing sparsity and correcting at each step, the weights that remain non-zero have the opportunity to adapt before the next pruning decision is made.

Algorithm. Algorithm 3 presents the complete Wanda++ procedure.

5.4 Experimental Results

5.4.1 Experimental Setup

Models. We evaluate on three LLM families spanning a wide range of scales: LLaMA-1 (7B, 13B, 30B, 65B) [3], OpenLLaMA (3B, 7B, 70B), and LLaMA-3.1 (8B). All models use FP16 precision.

Sparsity patterns. Unstructured sparsity at 50%, 60%, 70%, and 80%; semi-structured 2:4 and 4:8 patterns; structured row-wise at 10%, 30%, and 50%.

Calibration. 128 samples from the C4 training set with context length 2048

(Wanda++ full) or 128 (Wanda++ (M), medium variant). All methods use the same calibration set.

Evaluation. WikiText-2 perplexity (language modeling); 9 zero-shot downstream tasks via the `lm-evaluation-harness`: WiC, MRPC, HellaSwag, ARC-Easy, ARC-Challenge, Winogrande, BoolQ, RTE, and MMLU; inference latency (TTFT and TPOT on A100).

Baselines. SparseGPT [12], Wanda [11], GBLM [39]; and Wanda++ ablations (RGS-only; RO-only).

Hardware. Single NVIDIA H100-80G for models up to 13B; 4× H100 for 30B and 65B models.

5.4.2 Language Modeling Perplexity

Unstructured 50% sparsity. Table results across LLaMA-1 show consistent improvement over all baselines. On LLaMA-1-7B, Wanda++ achieves WikiText-2 PPL of **7.02**, compared to 7.26 for Wanda and 7.15 for GBLM — matching or exceeding GBLM quality at 40× lower cost (290 s vs. 5,801 s). Ablations confirm that RGS (7.18) and RO (7.07) each contribute independently, and their combination (7.02) is strictly better than either alone. The pattern holds across scales: 6.00 at 13B, 5.10 at 30B, and 4.43 at 65B, all improvements of approximately 3% over Wanda.

2:4 semi-structured sparsity. This is the setting where Wanda++ shows its largest gains, and where the accuracy gap between training-free pruning and dense models is widest. At 2:4 sparsity, Wanda degrades severely on smaller models: LLaMA-1-7B PPL rises to 11.53 (from dense 5.68), and OpenLLaMA-7B rises to 28.04 (from 7.27). Wanda++ reduces these to **9.43** (−18%) and **19.03** (−32%) respectively. On LLaMA-3.1-8B, Wanda++ achieves 18.32 versus Wanda’s 24.83 (−26%), demonstrating that the

benefit generalizes to newer model families. Figure ?? summarizes results across methods and sparsity levels.

Higher unstructured sparsity (60%–80%). At 70% sparsity, Wanda PPL rises to 76.17 (catastrophic degradation) while GBLM recovers to 54.60. Wanda++ achieves 55.52—competitive with GBLM and substantially better than Wanda—at $20\times$ lower computation cost. At 80% sparsity, all methods degrade severely, but Wanda++ remains below Wanda, indicating that the regional gradient signal provides consistent signal even at extreme sparsity.

Method	Sparsity	LLaMA-1				OpenLLaMA		LLaMA-3.1	
		7B	13B	30B	65B	3B	7B	70B	8B
Baseline	-	5.68	5.09	4.77	3.56	7.27	6.49	4.30	6.39
SparseGPT*	0.5	7.22	6.21	5.31	4.57	10.41	8.57	-	-
Wanda*		7.26	6.15	5.24	4.57	12.37	9.15	5.25	9.99
GBLM		7.15	6.11	5.18	-	10.75	8.49	-	9.90
Wanda++ RO		7.07	6.08	5.12	4.43	9.86	8.27	5.14	9.34
Wanda++ RGS		7.18	6.12	5.15	4.48	10.78	8.50	5.19	9.92
Wanda++		7.02 (-3%)	6.00 (-2%)	5.10 (-3%)	4.43 (-3%)	9.25 (-25%)	7.82 (-15%)	5.11 (-3%)	9.22 (-7%)
SparseGPT*	2:4	11.00	9.11	7.16	6.28	15.91	11.62	-	-
Wanda*		11.53	9.58	6.90	6.25	28.04	15.35	6.47	24.83
GBLM		11.33	9.16	6.87	-	24.75	13.19	-	24.34
Wanda++ RO		10.78	7.89	6.51	5.86	19.41	11.69	6.37	19.43
Wanda++ RGS		11.46	9.44	6.93	6.23	24.77	13.27	6.40	24.54
Wanda++		9.43 (-19%)	7.75 (-20%)	6.39 (-7%)	5.59 (-11%)	19.03 (-32%)	11.30 (-26%)	6.35 (-2%)	18.32 (-26%)
SparseGPT*	4:8	8.61	7.40	6.17	5.38	12.20	9.79	-	-
Wanda*		8.57	7.40	5.97	5.30	16.83	11.38	5.73	14.63
GBLM		8.48	7.26	5.89	-	14.86	10.38	-	14.29
Wanda++ RO		8.34	7.18	5.73	5.11	13.10	9.52	5.67	12.88
Wanda++ RGS		8.58	7.33	5.90	5.17	14.92	10.42	5.70	14.32
Wanda++		7.88 (-8%)	6.75 (-9%)	5.65 (-5%)	5.07 (-4%)	12.54 (-25%)	9.42 (-17%)	5.65 (-1%)	12.55 (-14%)

Table 5.1: Wikitext perplexity comparison on LLaMA-1, OpenLLaMA, and LLaMA-3.1 model families. * indicates results from either the previous paper [11]. - means results are not applicable from running their source code directly or out-of-memory issue. Bold highlights relative perplexity improvements over Wanda of 5% or more.

5.4.3 Zero-Shot Downstream Tasks

We evaluate on 9 zero-shot benchmarks using LLaMA-1-7B at 2:4 sparsity. Wanda++ outperforms Wanda on 8 of 9 tasks. The improvements are largest on tasks requiring multi-step reasoning:

- **MRPC** (paraphrase detection): Wanda++ 68.38% vs. Wanda 46.81% — a **46% relative improvement**, nearly recovering the dense model score of 69.12%.
- **RTE** (textual entailment): Wanda++ 62.09% vs. Wanda 49.82% (+**24% relative**); also outperforms GBLM (57.43%).
- **HellaSwag** (commonsense NLI): Wanda++ 45.31% vs. Wanda 41.66% (+9%), and matches GBLM (41.80%) at a fraction of the cost.

Averaged over all 9 tasks, Wanda++ scores 52.2% versus Wanda’s 47.8%, a **+4.4% absolute improvement**. These results confirm that better perplexity from regional gradient pruning translates to better downstream task accuracy, not just to better language modeling.

Method	Wic	Mrpc	Hellaswag	Arc_easy	Arc_challenge	Winogrande	BoolQ	RTE	MMLU
Baseline	49.84	69.12	56.96	75.29	41.80	70.00	75.02	66.43	35.10
Wanda	48.75	46.81	41.66	59.34	27.47	61.96	69.60	49.82	25.85
GBLM	49.32	65.31	41.80	61.43	30.45	63.24	71.20	57.43	26.34
<i>Wanda++ RGS</i>	49.37 (1%)	64.46 (38%)	41.43 (-1%)	62.42 (5%)	31.06 (13%)	62.83 (1%)	67.95 (-2%)	58.48 (17%)	26.40 (-2%)
<i>Wanda++</i>	50.00 (2%)	68.38 (46%)	45.31 (8%)	63.72 (7%)	29.27 (6%)	65.04 (4%)	67.80 (-2%)	62.09 (24%)	27.52 (6%)

Table 5.2: Zero-shot accuracy (%) from LLaMA-1 7B across various tasks under 2:4 sparsity.

5.4.4 Pruning Efficiency and RO Convergence

Pruning time. Wanda++ (M) prunes LLaMA-1-7B in **290 seconds** (under 5 minutes) on a single H100, using context length 128 for calibration. This is 20× faster than GBLM (5,801 s) and 5× slower than Wanda (55 s)—a cost that is fully amortized over any realistic deployment scenario. At 65B, Wanda++ (M) completes in 1,821 s versus GBLM’s 49,663 s (27× faster), while using 280 GB of memory versus GBLM’s 269 GB (comparable).

RO convergence. Figure 5.3 shows perplexity after each RO iteration on OpenLLaMA-3B. The block-output discrepancy converges within $K = 5$ optimization

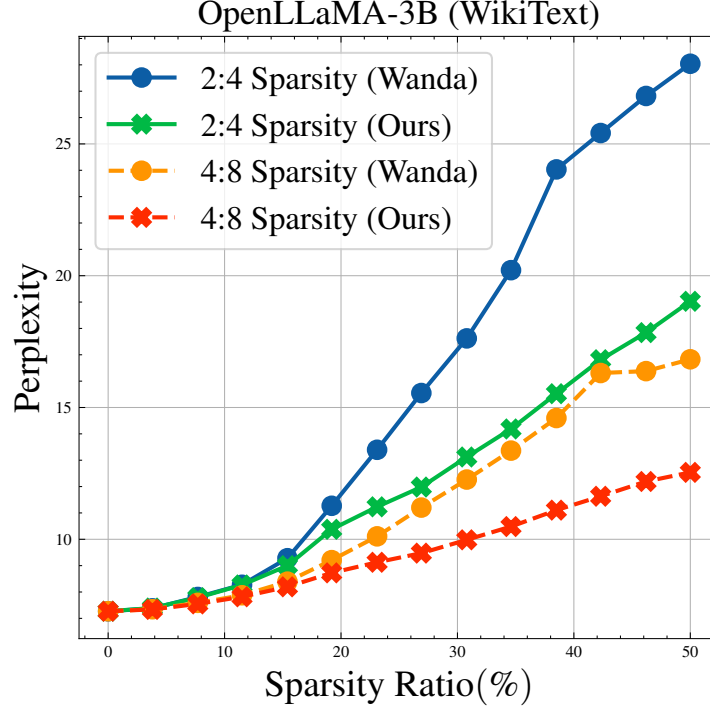


Figure 5.3: WikiText-2 perplexity of OpenLLaMA-3B during RO iterations at 2:4 sparsity. The block-output discrepancy converges within $K = 5$ steps, confirming that RO overhead is bounded.

steps in all evaluated settings, confirming that the RO overhead is genuinely bounded. The $K = 5$ default adds approximately 145 s to the total Wanda++ runtime for 7B, which is the difference between the 147 s RGS-only and 290 s full Wanda++ timings.

5.4.5 Orthogonality with Sparsity-Aware Fine-Tuning

A key property of RO is that it operates entirely within the pruned model’s weight space: it adjusts the values of non-zero weights to minimize local output discrepancy, but does not change the sparsity mask. Subsequent sparsity-aware fine-tuning methods such as LoRA [7] add low-rank corrections on top of the pruned (and RO-corrected) weights, targeting global task performance rather than local output fidelity. These two objectives are complementary.

Empirically, on LLaMA-1-7B at 2:4 sparsity: Wanda alone achieves PPL 11.53; Wanda + LoRA recovers to 8.23 (−29%); Wanda++ achieves PPL 9.43; Wanda++ + LoRA recovers to **6.88** (−27%). The relative gain from LoRA is similar in both cases, confirming that RO does not “use up” the slack that LoRA exploits. Indeed, starting from a better-pruned model (Wanda++) and then applying LoRA yields a final PPL of 6.88, which is 17% lower than the Wanda + LoRA baseline of 8.23.

5.4.6 Extension to Structured Pruning

Beyond unstructured and semi-structured sparsity, Wanda++ extends naturally to structured row-wise pruning, which removes entire rows of weight matrices and enables actual inference speedup on standard dense hardware without requiring specialized sparse tensor core support.

For row-wise pruning, the RGS score is aggregated per row: $S_i = \sum_j S_{ij}^{\text{RGS}}$ for row i , and entire rows with the lowest aggregate score are removed. RO then minimizes the same block-level output discrepancy loss, but now the sparse structure is enforced at the row level.

Results on LLaMA-1-7B demonstrate that the regional gradient signal generalizes cleanly to this setting. At 10% row sparsity, Wanda++-SP achieves PPL 7.84 versus Wanda-SP’s 9.10 (−14%); at 30% row sparsity, the gap widens dramatically: 63.45 versus 118.72 (−46%). Structured pruning is inherently more aggressive than unstructured pruning at the same sparsity rate, and the benefit of capturing non-linear weight interactions via regional gradients is proportionally larger.

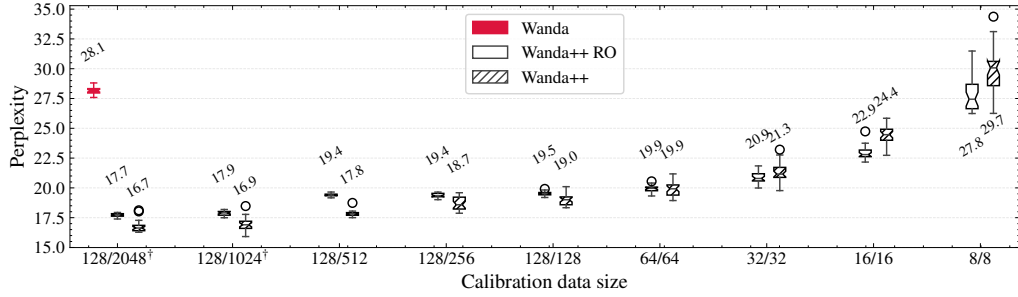


Figure 5.4: Inference latency improvements at 2:4 sparsity on an NVIDIA A100 GPU (FP16). Time-to-first-token (TTFT) reduction reaches 33–47% across input lengths 128–4096. Time-per-output-token (TPOT) reduction is 7–11%, reflecting the memory-bandwidth-bound nature of the decode phase.

5.4.7 Hardware Efficiency at 2:4 Sparsity

NVIDIA’s 2:4 structured sparsity format [81] is directly supported in the sparse tensor cores of A100 and H100 GPUs: weights are stored in a compressed format that retains 2 of every 4 consecutive values, reducing the weight memory footprint by 50% and enabling sparse matrix-multiplication routines that deliver actual throughput improvements beyond the memory savings.

Table 5.4.7 summarizes the measured latency and memory improvements for Wanda++-pruned LLaMA-1-7B at 2:4 sparsity. Across all tested batch sizes (1 and 4) and input lengths (128–4096 tokens), weight memory is reduced by a consistent **28%**. TTFT reduction ranges from 33% (short inputs at batch 1) to 47% (long inputs), reflecting the compute-bound nature of the prefill phase where sparse operations deliver near-theoretical speedup. TPOT reduction is more modest at 7–11%, since the decode phase is memory-bandwidth-bound and the reduction in parameter count provides the main benefit.

Batch	Input len.	Output len.	TTFT re.	TPOT re.	Weight mem.
1	128	64	33%	10%	−28%
1	1024	64	47%	11%	−28%
1	2048	64	47%	10%	−28%
1	4096	64	46%	10%	−28%
4	128	64	45%	11%	−28%
4	1024	64	47%	11%	−28%
4	4096	64	43%	7%	−28%

Table 5.3: Inference latency and memory reduction for Wanda++-pruned LLaMA-1-7B at 2:4 sparsity (FP16, single NVIDIA A100-80G GPU).

5.5 Conclusion

This chapter introduced Wanda++, a post-training LLM pruning framework that bridges the gap between fast, training-free pruning (Wanda) and accurate but costly gradient-based pruning (GBLM) through the principle of regional gradient computation. The core insight is that the information gradient-based methods use—how each weight influences the model’s output—is largely self-contained within each decoder block. By defining a block-level ℓ_2 surrogate loss and computing block-local gradients, Wanda++ captures within-block non-linear weight interactions at a fraction of the cost of full-model backpropagation.

Two innovations build on this insight. The Regional Gradient Score (RGS) integrates block-level gradients into the Wanda saliency criterion, improving WikiText-2 perplexity by up to 32% over Wanda at 2:4 semi-structured sparsity. Regional Optimization (RO) further reduces pruning-induced output discrepancy through iterative block-local weight updates that are fully orthogonal to subsequent sparsity-aware fine-tuning. Together, RGS and RO prune a 7B model in under 5 minutes on a single GPU while matching GBLM quality at $40\times$ lower computation cost. At deployment time, the 2:4 sparsity pattern yields 28% memory reduction and 33–47% TTFT reduction on NVIDIA sparse tensor cores with no software changes.

More broadly, the regional gradient principle introduced here reflects a recurring theme across this dissertation: efficiency is achieved by aligning the scope of computation with the granularity of the problem. Just as tensor-train decomposition in Chapter 3 achieved parameter efficiency by matching the dimensionality of adapters to the intrinsic dimension of fine-tuning, and just as SharpZO in Chapter 4 achieved optimization efficiency by restricting perturbations to dimensions identified as locally important, Wanda++ achieves pruning efficiency by restricting gradient computation to the block granularity at which pruning decisions are made.

The final chapter of this dissertation turns from model compression to model capability, asking whether a small language model can be trained to use visual tools effectively in a multimodal reasoning setting. Chapter 6 presents IAPO, an attribution-aware reinforcement learning method that addresses the credit assignment problem in multimodal tool-use training—a challenge that, like the challenges addressed in this chapter, requires rethinking the scope and structure of the supervision signal rather than simply scaling existing methods.

Chapter 6

Improving Tool Use in Small LLM through Reinforcement Learning

6.1 Introduction

Reinforcement learning (RL) has become a central paradigm for improving performance on complex long-horizon reasoning tasks in large language models (LLMs) [15, 16, 82, 83]. Owing to its ability to directly optimize sequential decision making, RL has been widely used to train LLMs as autonomous agents through online interaction with dynamic rollout trajectories [?]. At the same time, the increasing scale of LLMs has made inference increasingly expensive, which motivates growing interest in small language model (SLM) agents that retain useful reasoning and tool-use ability at substantially lower deployment cost.

Recent work has begun to explore how stronger teacher agents can be used to improve smaller models. For example, [46] propose an agent distillation framework based on supervised fine-tuning, while [47] introduce a data synthesis pipeline for RL-based SLM agent training. These developments suggest that RL can provide stronger supervision

than supervised fine-tuning alone in agent settings [84]. However, despite this progress, RL training for SLM agents still largely relies on vanilla algorithms such as Group Relative Policy Optimization (GRPO), which optimize final-task correctness but provide limited supervision over the quality of intermediate tool-use decisions.

This limitation is particularly pronounced in multimodal tool-use settings. Existing methods for tool-use reward design, such as ToRL and TRM, were primarily developed for textual tool-use scenarios [52, 53]. These approaches often rely on rigid format checks, execution heuristics, or exact matching against ground-truth tool trajectories. Such assumptions become problematic in multimodal tasks, where multiple tool-use paths may be equally valid. For example, when reasoning over a chart, a model may choose to highlight, mask, or draw boxes around relevant regions; these actions differ syntactically but can all provide useful visual evidence for solving the task. As a result, directly adapting text-domain reward designs to multimodal tool use is suboptimal.

Meanwhile, recent multimodal tool-use systems such as VTool-R1 and OpenThinkIMG still rely on vanilla GRPO for training [50, 51]. However, as will be shown later in this chapter, multimodal SLMs trained in this way continue to produce a substantial number of misused tool calls even after extended training. This suggests that the core challenge is not only final answer correctness, but also whether the model learns to ground its tool-use decisions in the appropriate parts of the multimodal input.

To address this issue, this chapter introduces **Input Attribution-Aware Policy Optimization** (IAPO), a reinforcement learning algorithm designed specifically for improving tool use in small multimodal agents. The key idea is to use a stronger teacher model to supervise the student’s intermediate decision process through input attribution. Instead of rewarding only the final answer, IAPO measures how much different parts of the input contribute to the generated tool call, and encourages the student model to align its attribution pattern with that of a stronger teacher. This provides a dense,

process-level training signal without requiring annotated tool trajectories.

The main contributions of this chapter are as follows:

- We identify a key limitation of vanilla GRPO for multimodal SLM tool-use training: outcome-only rewards fail to adequately supervise intermediate tool-selection decisions.
- We introduce an integrated-gradients-based attribution score that quantifies the contribution of different input components to generated tool calls.
- We propose IAPO, which incorporates an attribution alignment penalty into RL training to transfer teacher grounding behavior to a smaller multimodal policy model.
- We show through case studies and experiments that IAPO improves tool-use behavior and yields stronger downstream visual question answering performance than existing baselines.

6.2 Background on Visual Tool Use

This chapter considers a multimodal tool-use setting in which a vision-language model (VLM) is trained to invoke visual editing tools during intermediate reasoning. The goal of these tools is to transform the input image in a way that makes relevant evidence more salient for downstream reasoning. This formulation has recently emerged as a practical approach for enabling models to reason more effectively over charts, tables, and other structured visual inputs [49, 50, 51].

In the setting studied here, the model receives a prompt $\mathbf{x}$ together with an image $\mathbf{I}$ and produces a tool-call sequence followed by a final answer. The prompt contains several

semantically distinct parts, including the user query, image placeholder, tool definitions, and task instructions. Following the original paper, I divide the input prompt into an ordered sequence of semantic blocks,

$$\mathbf{x} = [\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_K],$$

where each block belongs to one of four types: pure-text context (P), user query (R), image placeholder (I), and tool definition (T). This block structure is important because it allows the model’s tool-call behavior to be analyzed in terms of which parts of the input most strongly influence its decision.

As illustrated in Fig. 6.1, the prompt template includes both the multimodal input and the definitions of a fixed set of visual editing tools. In this work, the tool set follows the design used in VTool-R1 and includes six functions: highlighting relevant columns, highlighting relevant rows, masking irrelevant columns, masking irrelevant rows, drawing bounding boxes over relevant columns, and drawing bounding boxes over relevant rows. These tools offer different but often functionally similar ways of focusing the model’s attention on useful visual evidence.

The rollout process in this chapter follows a two-turn setup. In the first turn, the model generates a tool-call sequence $\mathbf{y}'$ conditioned on the prompt and image,

$$\mathbf{y}' \sim \pi_{\theta}(\cdot \mid \mathbf{x}, \mathbf{I}),$$

and the generated code is executed by a tool executor $\mathcal{T}$ to obtain an edited image $\mathbf{I}' = \mathcal{T}(\mathbf{y}', \mathbf{I})$. In the second turn, the model reasons over both the original and edited

System prompt template with an example division

```

{Image I} <Image tokens> </Image tokens>
{User Query R} <Query> </Query>
{Tool T1} def focus_on_columns_with_highlight ...
{Tool T2} def focus_on_rows_with_highlight ...
{Tool T3} def focus_on_columns_with_mask ...
{Tool T4} def focus_on_rows_with_mask ...
{Tool T5} def focus_on_columns_with_draw ...
{Tool T6} def focus_on_rows_with_draw ...
{Pure-text P1} # GOAL #: Based on the above tools, I want you to reason about how to solve the #
USER REQUEST # and generate the actions step by step (each action is a python function call) to
solve the request. You may need to use the tools above to process the images and make decisions
based on the visual outputs of the previous code blocks. You should only use the tools above, you
should not use other functions or code that will not be executed.
{Pure-text P2} # EXAMPLE #: Simple question that does not require any tool ... <Thought
0><Action 0><Observation><Edited Image><Thought 1><Answer>
# USER Bounding Box Info: x_values_bbox, storing x values and coordinates. y_values_bbox,
storing x values and coordinates. The x values in the image are: <x_values>. The y values in
the image are: <y_values>.
# USER IMAGE stored in image_1, as PIL image.

```

Figure 6.1: System prompt template with an example division used in IAPO rollouts.

visual input to generate the final answer,

$$\mathbf{y} \sim \pi_{\theta}(\cdot \mid \mathbf{x}, \mathbf{y}', \mathbf{I}, \mathbf{I}').$$

Existing visual tool-use methods typically train the model with vanilla GRPO using only a binary outcome reward R_i^{out} that indicates whether the final answer is correct. For each input sample, GRPO samples a group of responses and computes a normalized group-relative advantage:

$$\hat{A}_i = \frac{R_i^{\text{out}} - \text{mean}(\{R_j^{\text{out}}\}_{j=1}^G)}{\text{std}(\{R_j^{\text{out}}\}_{j=1}^G)}.$$

This objective improves final-task accuracy, but it does not directly evaluate whether the intermediate tool call was appropriate. A model may invoke an irrelevant tool, attend to the wrong region, or rely on spurious prompt correlations, yet still receive a high reward if it recovers and produces the correct final answer. This creates a severe credit assignment problem, especially for smaller models that are less able to infer correct tool-use behavior

from sparse, outcome-level supervision.

This limitation motivates the central question of this chapter: can a multimodal SLM be trained not only to produce correct answers, but also to ground its tool-use decisions in the right parts of the input? The proposed IAPO method answers this question by introducing an attribution-based reward that supervises the model’s intermediate reasoning process.

6.3 Input-Attribution-Aware Policy Optimization

In this section, I present **Input Attribution-Aware Policy Optimization (IAPO)**, a reinforcement learning algorithm for multimodal SLM training that augments the standard outcome reward with a teacher-guided attribution alignment penalty. The central idea is that correct tool-use behavior should be associated with correct input grounding: when a model selects the right tool, it should rely more heavily on the relevant parts of the prompt and image context, particularly the appropriate tool definitions. The IAPO method is illustrated in Fig. 6.2.

Integrated gradient score over input blocks

To quantify how different input components influence the generated tool call, I define an attribution score over the prompt blocks introduced above. After combining the textual prompt $\mathbf{x}$ with the image $\mathbf{I}$ and passing them through the multimodal processor, the model produces a merged input embedding matrix

$$\mathbf{E} = [\mathbf{e}_1, \dots, \mathbf{e}_S] \in \mathbb{R}^{S \times d},$$

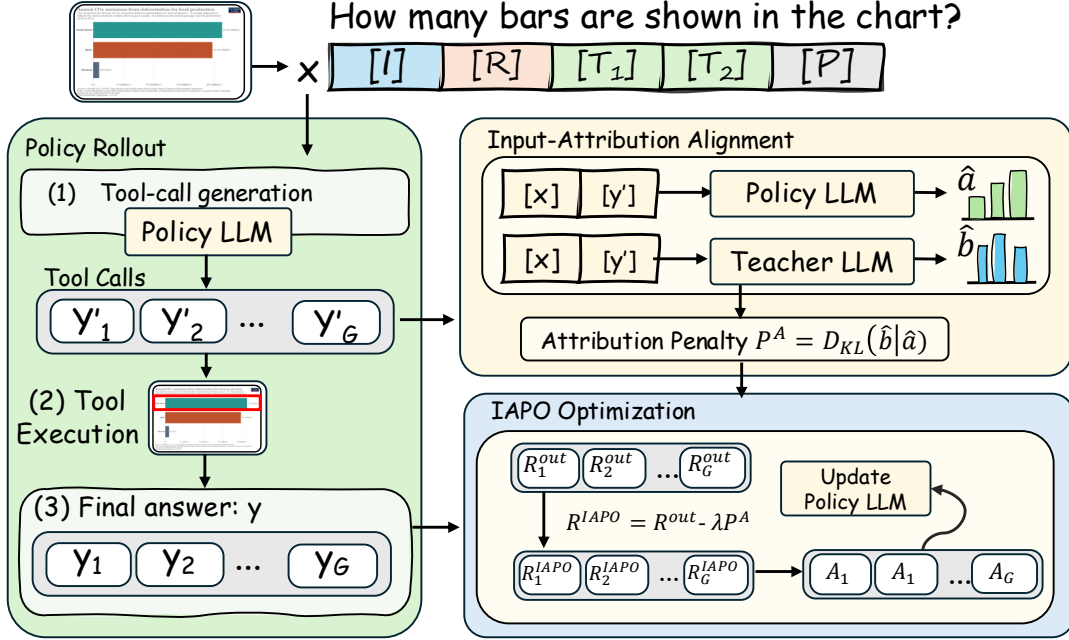


Figure 6.2: Overview of the IAPO method. The attribution penalty P^A is computed from the tool-call sequence $\mathbf{y}'$ generated in the first rollout. The final IAPO reward combines the standard outcome-based reward R^{out} with the attribution penalty P^A .

where S is the tokenized multimodal input length and d is the hidden dimension. Let $\Omega_k \subseteq \{1, \dots, S\}$ denote the set of token indices corresponding to block $\mathbf{b}_k$.

For a given rollout, let $\mathbf{y}' = [y'_1, \dots, y'_{L'}]$ be the generated tool-call sequence. I first define a scalar attribution target as the log-probability of the full tool-call sequence under the current policy:

$$s(\mathbf{E}) = \log \pi_{\theta}(\mathbf{y}' | \mathbf{E}) = \sum_{t=1}^{L'} \log \pi_{\theta}(y'_t | y'_{<t}, \mathbf{E}).$$

The gradient of this score with respect to the input embedding of token i is

$$\mathbf{g}_i = \frac{\partial s(\mathbf{E})}{\partial \mathbf{e}_i}, \quad i = 1, \dots, S.$$

Using a gradient-times-embedding formulation, the token-level attribution score is defined

as

$$o_i = \|\mathbf{g}_i \odot \mathbf{e}_i\|_2,$$

where $\odot$ denotes element-wise multiplication. Intuitively, o_i measures how much the i -th token contributes to increasing the probability of the generated tool-call sequence.

Finally, token-level scores are aggregated within each semantic block:

$$O_k = \sum_{i \in \Omega_k} o_i.$$

The resulting block-level attribution score O_k indicates how strongly each prompt component influences the model’s tool-use decision.

Teacher-guided attribution alignment

The empirical motivation for IAPO is that stronger teacher models exhibit more appropriate attribution patterns when generating tool calls. In particular, when a teacher selects the correct tool, it tends to assign more attribution mass to the corresponding tool definitions and other relevant input components. This suggests that attribution can be used as a distillation target.

Let

$$\mathbf{a}_i = \{O_1^a, \dots, O_K^a\}, \quad \mathbf{b}_i = \{O_1^b, \dots, O_K^b\},$$

denote the block-level attribution scores of the actor and teacher, respectively, for the i -th sampled trajectory. After normalizing them into distributions $\hat{\mathbf{a}}_i$ and $\hat{\mathbf{b}}_i$, I define the attribution alignment penalty as

$$P_i^A = D_{\text{KL}}(\hat{\mathbf{b}}_i \parallel \hat{\mathbf{a}}_i).$$

This penalty is small when the student attends to the same input blocks as the teacher and large when its attribution pattern deviates.

IAPO reward and policy optimization

IAPO modifies the trajectory reward by combining the standard outcome reward with the attribution penalty:

$$R_i^{\text{IAPO}} = R_i^{\text{out}} - \lambda P_i^A, \quad (6.1)$$

where λ controls the strength of attribution alignment. Given a group of G sampled responses associated with the same input, the group-relative advantage is computed as

$$A_i^{\text{IAPO}} = \frac{R_i^{\text{IAPO}} - \text{mean}(\{R_j^{\text{IAPO}}\}_{j=1}^G)}{\text{std}(\{R_j^{\text{IAPO}}\}_{j=1}^G) + \epsilon}.$$

Following standard GRPO, this advantage is broadcast to all response tokens and used in the clipped surrogate objective. In this way, the policy is encouraged not only to maximize downstream task accuracy, but also to align its intermediate tool-use behavior with the teacher’s input-grounding pattern.

Conceptually, IAPO addresses the main weakness of vanilla GRPO. Instead of relying solely on sparse outcome supervision, it provides a denser process-level signal that helps the student learn why a tool should be selected, not merely whether the final answer was correct. This is particularly beneficial for smaller multimodal models, whose limited reasoning capacity makes them more vulnerable to spurious shortcuts and weak credit assignment.

6.4 Experimental Results

6.4.1 Experimental Setup

I evaluate IAPO on visual question answering tasks with visual editing tools using the Qwen2.5-VL model family [85]. RL training is conducted with the VeRL framework [86] and the AdamW optimizer [63]. Unless otherwise noted, the student backbone is Qwen2.5-VL-3B-Instruct, and experiments are run on NVIDIA Tesla A100-40GB GPUs.

The overall training pipeline contains two stages. First, I perform a short cold-start phase using standard GRPO so that the model acquires basic tool-calling ability. Second, I switch to the proposed IAPO reward in Eq. (6.1) for the main training stage. To provide teacher guidance, I first train a Qwen2.5-VL-7B-Instruct model with standard GRPO on the same training set, and then use its attribution scores during student training.

The tool set follows VTool-R1 for fair comparison and contains six visual editing functions:

- `focus_on_columns_with_highlight`
- `focus_on_rows_with_highlight`
- `focus_on_columns_with_mask`
- `focus_on_rows_with_mask`
- `focus_on_columns_with_draw`
- `focus_on_rows_with_draw`

These tools allow the model to manipulate charts and tables by highlighting, masking, or drawing boxes around relevant rows or columns.

I compare IAPO against two categories of baselines:

- **Off-the-shelf models:** GPT-4o and Qwen2.5-VL-3B under inference-only tool use.
- **RL baselines:** VTool-R1 trained with vanilla GRPO, TRM adapted to the visual setting using GPT-4o as the verifier, and ToRL adapted by incorporating an execution penalty following the original method.

The evaluation covers both chart and table reasoning benchmarks. The chart benchmarks include CharXiv and the horizontal and vertical subsets used in prior work, while the table benchmarks include VWTQ, VWTQ_syn, and VTabFact. The same evaluation protocol is applied across all compared methods.

6.4.2 Case Study

To build intuition for why IAPO is effective, I first conduct a case study that analyzes tool-calling behavior and attribution patterns during training. The goal of this study is to understand whether vanilla GRPO sufficiently teaches a multimodal SLM to select the correct tool, and whether the proposed attribution score can expose failure modes in this process.

For this analysis, I construct a validation subset of 256 ChartQA samples that are clearly either column-divisible or row-divisible. In such examples, the correct tool family is unambiguous: column-based tools should be used for column-divisible plots, and row-based tools should be used for row-divisible plots. This setup enables direct measurement of tool-selection correctness.

During vanilla GRPO training of a 3B model, I evaluate this subset every 5 training steps and record two statistics: the total number of misused tool calls and the normalized attribution mass assigned to correct versus incorrect tool definitions. I also evaluate a well-trained 7B teacher model under the same protocol for reference. The results are

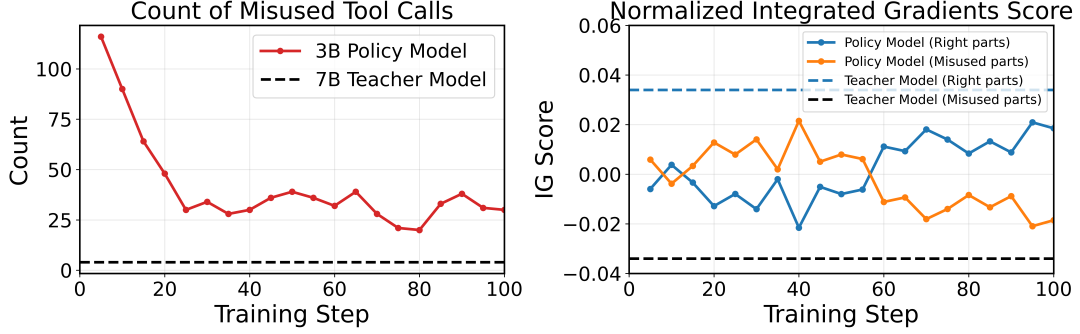


Figure 6.3: Case study: Count for misused tool calls and IG Scores with selected dataset. We use Qwen2.5-VL-3B as the policy model and a well-trained Qwen2.5-VL-7B model as the teacher. Both models are trained with the vanilla GRPO method used in VTool-R1.

shown in Fig. 6.3.

Several observations follow from this study. First, even after 100 steps of vanilla GRPO training, the 3B model still makes 30 incorrect tool calls out of 256 evaluated samples. This shows that outcome-only RL does not sufficiently penalize wrong intermediate actions, because an incorrect tool call may still be followed by a correct final answer. Second, the attribution score is highly correlated with tool-call correctness: as training progresses, the attribution assigned to the correct tool definitions gradually increases and eventually exceeds the attribution assigned to incorrect tools. Third, the 7B teacher model exhibits both fewer tool-selection errors and a clearer separation in attribution mass between correct and incorrect tool groups, suggesting that stronger models possess more reliable input-grounding patterns.

6.4.3 Comparison with Baseline Methods

I next compare IAPO with the baseline methods on chart and table visual question answering tasks. For each method, I train separate models for the chart and table settings and report the test accuracy on six evaluation sets in Table 6.1.

Three conclusions are especially noteworthy. First, small multimodal models struggle

	Chart			Table			Avg
Methods	CharXiv	Horizontal	Vertical	VWTQ	VWTQ_syn	VTabFact	
Large Scale Multimodal LLMs							
GPT-4o	47.9	75.7	81.7	67.5	72.8	91.6	72.9
Qwen-2.5VL-7B	31.0	69.6	48.7	37.5	52.6	45.6	47.5
+ VTool-R1 (GRPO)	39.2	81.3	75.4	52.6	68.4	69.1	64.3
Small Scale Multimodal LLMs							
Qwen-2.5VL-3B	18.9	25.7	22.8	14.2	26.3	36.8	24.1
+ VTool-R1 (GRPO)	23.1	63.7	60.9	46.6	52.6	70.5	52.9
+ TRM*	23.6	64.3	62.7	44.3	54.6	70.9	53.4
+ ToRL*	24.1	64.0	63.1	44.4	56.1	71.0	53.8
+ IAPO	26.1	66.0	64.1	46.7	60.2	72.1	55.9

Table 6.1: Performance comparison of baseline methods. * Since TRM and ToRL were originally proposed for text-only tool-use settings, we adapt them to visual tool-use setup. For TRM, we directly use GPT-4o as the tool-call reward model for convenience, instead of training a separate offline model on GPT-4o-generated data as done in original paper.

to use tools effectively without targeted training. The off-the-shelf Qwen2.5-VL-3B model performs poorly under inference-only tool use, while larger models such as GPT-4o and Qwen2.5-VL-7B achieve substantially stronger performance. This highlights the difficulty of tool use for smaller models.

Second, existing reward designs remain suboptimal in the multimodal setting. Although TRM and ToRL improve over vanilla GRPO, both consistently underperform IAPO. This suggests that deterministic tool verification or fixed execution penalties do not fully address the ambiguity and diversity of valid tool-use strategies in visual reasoning tasks.

Third, IAPO demonstrates stronger generalization. The most notable gain appears on the out-of-distribution CharXiv benchmark, where IAPO achieves the best performance among the small-model training methods. This indicates that attribution-based teacher guidance helps the student focus on more relevant evidence and avoid overfitting to narrow training patterns.

Overall, the experimental results show that IAPO improves not only tool-use behavior during training, but also end-task accuracy across a range of chart and table reasoning

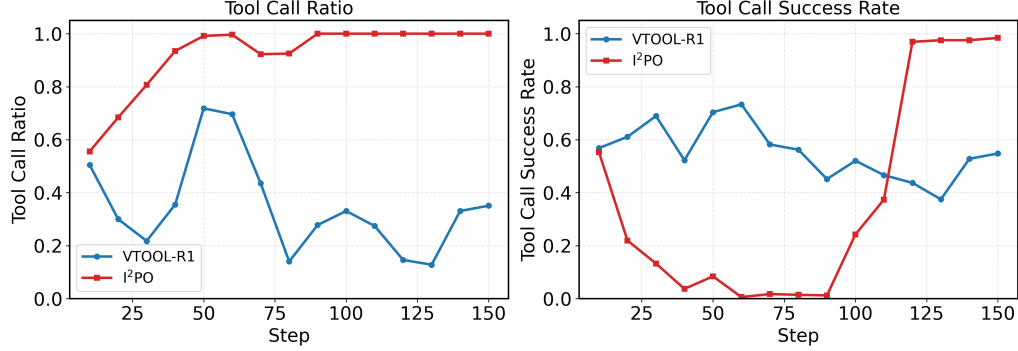


Figure 6.4: Tool use behaviors during training.

benchmarks.

6.5 Tool-calling Behavior

To further analyze the training dynamics of IAPO, I compare it against vanilla GRPO using training-time tool-use statistics. As shown in Fig. 6.4, IAPO maintains a substantially higher and more stable tool-call ratio throughout training, indicating that the model learns to rely on tools more consistently rather than falling back to internal reasoning. More importantly, although the tool-call success rate of IAPO is initially modest, it rises sharply during later training and eventually approaches nearly perfect performance, substantially exceeding the GRPO baseline. This suggests that the student first learns to call tools more actively, and then gradually aligns its attribution pattern with teacher guidance, leading to improved tool selection.

6.6 Conclusion

This chapter presented IAPO, a reinforcement learning method for improving tool use in small multimodal agents through attribution-guided policy optimization. The core idea is to move beyond outcome-only supervision and instead incorporate process-

level guidance by aligning the student model’s input attribution pattern with that of a stronger teacher. To achieve this, I introduced an integrated-gradients-based attribution score that quantifies how different parts of the input contribute to the generated tool call, and used the discrepancy between student and teacher attribution distributions as an additional training penalty.

The resulting method addresses a key weakness of vanilla GRPO in multimodal tool-use settings: correct final answers do not necessarily imply correct intermediate tool decisions. By providing a denser signal over the model’s decision process, IAPO encourages the policy to ground tool selection in the appropriate evidence. Empirically, this leads to better tool-use behavior during training and stronger downstream performance on visual question answering benchmarks.

More broadly, the results in this chapter suggest that attribution can serve as an effective bridge between teacher knowledge and online RL optimization for small multimodal agents. This opens a promising direction for designing RL algorithms that supervise not only what an agent outputs, but also how it arrives at its intermediate decisions.

Beyond reinforcement learning, we were also inspired by the idea of GRPO in our recent work on zeroth-order optimization. Building on this insight, we proposed GRZO [87], a GRPO-inspired ZO method that significantly improves the performance of zeroth-order optimization.

Chapter 7

Conclusion

7.1 Summary of Contributions

This dissertation has investigated efficient training and inference for large language models from four complementary angles: parameter-efficient adaptation via tensor-train decomposition, backpropagation-free fine-tuning via zeroth-order optimization, post-training inference acceleration via regional pruning, and capability improvement for small language model agents via attribution-aware reinforcement learning. Across these settings, a common design principle has emerged: efficiency is best achieved by aligning the scope of computation with the granularity of the problem, rather than by uniformly reducing resources or uniformly increasing them.

Chapter 3: Efficient Fine-Tuning via Tensor-Train Decomposition. The first line of work established that tensor-train (TT) decomposition provides a principled basis for both parameter efficiency (LoRETTA) and memory efficiency (AdaZeta) in LLM fine-tuning. LoRETTA demonstrated that replacing adapter and reparameterization modules with TT-format weight representations reduces trainable parameter counts to sub-1 MB levels on LLaMA-2-7B—80–100 $\times$ fewer than LoRA—without sacrificing downstream ac-

curacy. The key insight is that TT decomposition respects the intrinsic dimensionality of fine-tuning: because the optimal fine-tuning update lies in a low-dimensional subspace [17], a TT representation with small rank can capture this subspace more compactly than any matrix rank-1 factorization. This connection is not accidental: the TT format naturally imposes a structured low-rank constraint that operates at a finer granularity than LoRA’s single-matrix factorization, enabling greater compression at the same or better accuracy.

AdaZeta extended this TT-based compression into the backpropagation-free setting. By combining TT adapters with a fast-forward parallel contraction strategy and an adaptive query schedule $Q_k = \min(\alpha e_k^\beta, Q_{\max})$, AdaZeta resolved the divergence and variance problems that limit existing zeroth-order methods at LLaMA-2-7B scale. The theoretical convergence analysis—showing that the bound $\mathcal{O}((R + \epsilon^2 L + C(d, \epsilon) \sum_k 1/Q_k)/(K\epsilon))$ tightens multiplicatively with both smaller d and growing Q_k —provides a rigorous explanation for why TT compression and adaptive queries compound rather than substitute. Empirically, AdaZeta outperforms MeZO, MeZO-LoRA, and Sparse-MeZO on 8 of 10 LLaMA-2-7B tasks while requiring only 14 GB of GPU memory, an $8\times$ reduction compared to full fine-tuning.

Chapter 4: Sharpness-Aware Zeroth-Order Optimization. The second line of work addressed a qualitatively different ZO setting: vision-language model prompt tuning on inference-only edge devices, where neither model weights nor gradients are accessible. In this setting, the dimensionality reduction approach of AdaZeta offers limited leverage—the trainable prompt already lives in a 512-dimensional space—so variance must be attacked from a different angle. SharpZO’s central insight is that zeroth-order gradient estimates are unreliable not primarily because of high dimension, but because of high curvature: in regions where the loss surface is sharp (large Hessian eigenvalues), small perturbations produce large and inconsistent gradient estimates. Stage 1

of SharpZO addresses this directly by steering CMA-ES global search toward flat loss basins via worst-case perturbation sampling; Stage 2 then exploits the flat neighborhood through sparse ZO-SGD with Z-pruning, a gradient-sensitivity criterion robust to the heavy-tailed outliers endemic to ZO estimation.

SharpZO achieved a 4.7% average accuracy improvement over the state-of-the-art BP-free baseline ZIP across 11 CLIP classification benchmarks, with up to 7% gains on fine-grained datasets. It also converged 20% faster in wall-clock time, requiring only one forward query per Stage 2 step rather than the five required by ZIP. The theoretical convergence bound $t \approx T_c + \mathcal{O}((\eta\mu)^{-1} \log(L^3\eta^2\rho^2/\varepsilon))$ makes the Stage 1 contribution explicit: the sharpness radius ρ clips the effective Lipschitz constant seen by Stage 2, compounding the benefit of a warm-start initialization with a reduction in the optimization difficulty of the local search problem.

Chapter 5: Accelerating LLM Inference via Regional Pruning. The third line of work shifted from training-time to inference-time efficiency. Post-training pruning offers a one-shot path to reducing the memory footprint and compute cost of deployed LLMs, but existing methods occupy two unattractive extremes: training-free methods like Wanda are fast but inaccurate at high sparsity, while gradient-informed methods like GBLM are accurate but require full-model backpropagation that scales impractically to large models (97 minutes and 269 GB for LLaMA-65B). Wanda++ resolved this tension through the principle of regional gradient computation: by defining a block-level ℓ_2 surrogate loss $\mathcal{L}_{\text{RGS}}^l = \|f^l(\mathbf{X}^l)\|_2$, Wanda++ computes gradient signals that are fully self-contained within each decoder block and therefore 40× faster to compute than GBLM, while capturing within-block non-linear weight interactions that Wanda’s activation-norm proxy misses entirely.

The Regional Optimization (RO) component further minimized pruning-induced output discrepancy through iterative block-local weight updates, with the crucial property

of orthogonality to LoRA sparsity-aware fine-tuning. Together, RGS and RO improved WikiText-2 perplexity by up to 32% over Wanda at 2:4 semi-structured sparsity on OpenLLaMA-7B, and improved zero-shot downstream accuracy by 4.4% on average across 9 benchmarks. At deployment time, 2:4 sparsity yields 28% weight memory reduction and 33–47% time-to-first-token reduction on NVIDIA sparse tensor cores. The entire pruning pipeline completes in under 5 minutes for a 7B model on a single H100.

Chapter 6: Improving Tool Use in Small Language Models. The fourth line of work addressed a different aspect of efficiency: the capability of small models to perform effectively as multimodal agents. Rather than compressing large models, this chapter asked how to train small vision-language models that use visual tools reliably in a reinforcement learning setting. IAPO identified a fundamental limitation of vanilla GRPO in this context: outcome-only rewards cannot distinguish between correct tool calls grounded in the right visual evidence and spurious correct calls that exploit task-irrelevant features. This credit assignment failure is especially pronounced for small models, which lack the implicit regularization of scale and therefore learn the wrong input-output associations unless explicitly supervised.

IAPO’s solution is to add an attribution alignment penalty to the RL objective, derived from integrated-gradients attribution scores computed without access to the teacher’s model weights. By measuring the discrepancy between the student’s and teacher’s attribution distributions over the prompt’s semantic blocks, IAPO provides dense process-level supervision that rewards the student not only for producing correct answers, but for arriving at them through the right intermediate decisions. Empirically, IAPO improved VQA accuracy by 3% on average over vanilla GRPO across six test sets on Qwen2.5-VL-3B, while substantially reducing the rate of misused tool calls.

7.2 Unifying Themes

Viewed together, the four lines of work in this dissertation reflect a coherent perspective on what it means to design efficient algorithms for large language models. Three themes recur across chapters.

Scope alignment. Every method in this dissertation achieves efficiency by matching the scope of computation to the granularity of the problem, rather than operating at the full model scale. LoRETTA and AdaZeta align the parameter space of fine-tuning with the intrinsic dimension of the task. SharpZO aligns the scope of perturbation with the locally important dimensions identified by Z-pruning. Wanda++ aligns the scope of gradient computation with the decoder block at which pruning decisions are made. IAPO aligns the scope of supervision with the semantic blocks of the prompt that are most relevant to the tool-use decision. In each case, the algorithm’s efficiency gain is not merely a reduction in computation—it is a redirection of computation toward the part of the problem where it is most informative.

Locality as a computational principle. A related theme is the value of local computation. Local operations—whether within a TT adapter, within a decoder block, or within a semantic block of a prompt—are not only cheaper: they are often more appropriate. They avoid the noise that global computations introduce when many irrelevant signals are aggregated. TT adapters avoid the gradient instability of full-model ZO updates. Regional gradients avoid the label-dependent noise of cross-entropy loss. Z-pruning avoids the outlier sensitivity of raw gradient magnitudes. Attribution penalties avoid the credit assignment ambiguity of outcome-only rewards. In each case, restricting the scope of computation reduces noise as much as it reduces cost.

Structure-aware design. Efficient algorithms for large language models benefit from exploiting the structures that modern transformers provide: the block (decoder)

structure for pruning, the low-rank structure of weight updates for adaptation, the flat-minimum structure of the loss landscape for ZO optimization, and the semantic block structure of prompts for attribution. These structures are not incidental properties of current architectures—they are consequences of the inductive biases that make transformers effective. Algorithms that align with these structures not only run faster; they often generalize better, as evidenced by SharpZO’s improved distribution-shift robustness and LoRETTA’s anti-overfitting effect in low-data settings.

This dissertation also contributes three co-authored works that extend its core themes in complementary directions. Federated LoRETTA applies TT-based PEFT to federated learning settings, where communication efficiency imposes an additional constraint on adapter parameter counts. FLAT-LLM [88] extends the inference acceleration program of Wanda++ to structured low-rank decomposition, achieving further latency reduction on long-context tasks. MaZO introduces a momentum-accelerated zeroth-order optimizer that complements AdaZeta’s query schedule with improved update stability. Together, these extensions suggest that the design principles identified in this dissertation transfer beyond the specific problem settings they were developed for.

7.3 Limitations and Future Directions

Each line of work in this dissertation has limitations that point toward productive future research directions.

Scaling of TT-based methods. LoRETTA and AdaZeta were evaluated primarily on LLaMA-2-7B and smaller models. At larger scales (70B+), the memory footprint of TT adapter forward passes may become non-trivial due to the intermediate activations generated during tensor core contraction. Adaptive TT rank selection—determining which layers benefit most from TT compression and at what rank—remains an open

problem, particularly when the downstream task distribution is unknown at compression time. Future work could investigate TT decomposition with learned ranks, potentially combined with second-order information about the fine-tuning loss landscape.

ZO optimization at scale. AdaZeta’s adaptive query schedule relies on a fixed functional form $Q_k = \min(\alpha e_k^\beta, Q_{\max})$ for the query number schedule. More sophisticated schedule designs—perhaps informed by online estimates of gradient variance or loss curvature—could further improve convergence. SharpZO’s Stage 1 CGE gradient computation has cost $\mathcal{O}(d)$, which is tractable for $d = 512$ but would become expensive if applied to larger prompt representations. Extending the sharpness-aware CMA-ES to settings with $d \sim 10^4$ – 10^5 would require approximations to the CGE gradient, such as random projection-based estimates or block-coordinate CGE.

Post-training pruning recovery. Wanda++ demonstrates that regional gradients substantially improve post-training pruning quality, but a gap between pruned and dense model performance persists—especially at high sparsity (70–80%) where even Wanda++ degrades severely. This gap suggests that one-shot pruning methods have a fundamental accuracy ceiling that cannot be overcome without some form of task-aware fine-tuning. An important direction is the design of efficient sparsity-aware fine-tuning methods—combining the RO principle of local output-discrepancy minimization with LoRA-style low-rank updates in a joint optimization framework that avoids full-model backpropagation.

Attribution-based RL for diverse tool sets. IAPO was evaluated on a fixed six-tool visual editing vocabulary. In practical multi-tool agent settings, the tool set may be large, heterogeneous, and partially compositional, making attribution alignment harder to define and compute. Extending IAPO to settings where (a) tools have different modalities (text, code, image), (b) the teacher’s tool-use vocabulary differs from the student’s, or (c) multiple tools are invoked in sequence during a single reasoning turn

are all open challenges. More broadly, the question of how to scale attribution-based supervision to longer rollouts and larger action spaces is an important direction for RL-based agent training.

Cross-theme opportunities. Several connections between the dissertation’s research threads remain unexplored. ZO optimization of pruned models—running AdaZeta-style fine-tuning on a Wanda++-pruned model—could combine the memory efficiency of sparsity with the backpropagation-free adaptation of ZO, potentially enabling fine-tuning of very large models without any dense weight storage. Tensor-train compression of the sparse weight matrices produced by Wanda++ could further reduce the storage footprint of pruned LLMs deployed on edge devices. Attribution-aware training signals, analogous to those in IAPO, could be incorporated into ZO fine-tuning objectives to supervise intermediate reasoning in LLM agents without backpropagation. These directions suggest that the methods developed in this dissertation are not isolated contributions but building blocks of a broader framework for efficient language model systems.

7.4 Concluding Remarks

The central challenge this dissertation has addressed—making large language models efficient enough to train, fine-tune, and deploy within realistic computational constraints—is unlikely to diminish in importance. Model scale continues to increase, and the gap between the computational resources required by frontier models and those available on most hardware platforms continues to widen. At the same time, the demand for personalized, on-device, and low-latency AI applications continues to grow, creating persistent pressure for efficiency techniques that do not compromise capability.

The methods developed in this dissertation suggest that this gap can be narrowed not by accepting lower quality, but by being more precise about where quality comes

from. Fine-tuning quality comes from a small intrinsic subspace, not from the full weight space—and TT decomposition can target that subspace. ZO optimization quality comes from gradient estimates in flat regions, not sharp ones—and sharpness-aware warm-up can navigate there. Pruning quality comes from gradient information at the block level, not from activation statistics alone—and regional computation can provide it. Agent quality comes from grounded intermediate decisions, not from correct final answers alone—and attribution-based supervision can reinforce it.

Taken together, these contributions reflect a view of efficiency not as a constraint to be satisfied, but as a design objective in its own right: one that rewards algorithms that understand the structure of their problem deeply enough to do more with less. As language models continue to evolve, this perspective—aligning computation with the granularity of the problem—will remain a productive principle for researchers and engineers working to close the gap between what large models can do and what practical systems can afford.

Bibliography

- [1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, [arXiv preprint arXiv:1810.04805](#) (2018).
- [2] L. Floridi and M. Chiriatti, *Gpt-3: Its nature, scope, limits, and consequences*, *Minds and Machines* **30** (2020) 681–694.
- [3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et. al., *Llama: Open and efficient foundation language models*, [arXiv preprint arXiv:2302.13971](#) (2023).
- [4] M. Agarwal, A. Qureshi, N. Sardana, L. Li, J. Quevedo, and D. Khudia, *LLM inference performance engineering: Best practices*, Oct., 2023.
- [5] J. Tian, S. Azizi, Y. Zhao, E. B. Potraghloo, S. McPherson, S. N. Sridhar, Z. Wang, Z. Zhang, M. Pedram, and S. Kundu, *Skipkv: Selective skipping of kv generation and storage for efficient inference with large reasoning models*, [arXiv preprint arXiv:2512.07993](#) (2025).
- [6] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, *Parameter-efficient transfer learning for nlp*, in *International Conference on Machine Learning*, pp. 2790–2799, PMLR, 2019.
- [7] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, *Lora: Low-rank adaptation of large language models*, [arXiv preprint arXiv:2106.09685](#) (2021).
- [8] X. L. Li and P. Liang, *Prefix-tuning: Optimizing continuous prompts for generation*, in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597, 2021.

- [9] B. Lester, R. Al-Rfou, and N. Constant, *The power of scale for parameter-efficient prompt tuning*, in Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp. 3045–3059, 2021.
- [10] S. Malladi, T. Gao, E. Nichani, A. Damian, J. D. Lee, D. Chen, and S. Arora, *Fine-tuning language models with just forward passes*, Advances in Neural Information Processing Systems **36** (2023) 53038–53075.
- [11] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter, *A simple and effective pruning approach for large language models*, arXiv preprint arXiv:2306.11695 (2023).
- [12] E. Frantar and D. Alistarh, *Sparsegpt: Massive language models can be accurately pruned in one-shot*, in International Conference on Machine Learning, pp. 10323–10337, PMLR, 2023.
- [13] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, *Awq: Activation-aware weight quantization for on-device llm compression and acceleration*, Proceedings of Machine Learning and Systems **6** (2024) 87–100.
- [14] J. Tian, C. Fang, H. Wang, and Z. Wang, *Bebert: Efficient and robust binary ensemble bert*, in ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1–5, IEEE, 2023.
- [15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, arXiv preprint arXiv:1707.06347 (2017).
- [16] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu, et. al., *Deepseekmath: Pushing the limits of mathematical reasoning in open language models*, arXiv preprint arXiv:2402.03300 (2024).
- [17] A. Aghajanyan, L. Zettlemoyer, and S. Gupta, *Intrinsic dimensionality explains the effectiveness of language model fine-tuning*, arXiv preprint arXiv:2012.13255 (2020).
- [18] C. Li, H. Farkhoor, R. Liu, and J. Yosinski, *Measuring the intrinsic dimension of objective landscapes*, in International Conference on Learning Representations, 2018.
- [19] J. Pfeiffer, A. Kamath, A. Rücklé, K. Cho, and I. Gurevych, *Adapterfusion: Non-destructive task composition for transfer learning*, arXiv preprint arXiv:2005.00247 (2020).
- [20] A. Rücklé, G. Geigle, M. Glockner, T. Beck, J. Pfeiffer, N. Reimers, and I. Gurevych, *AdapterDrop: On the efficiency of adapters in transformers*, in Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp. 7930–7946, 2021.

- [21] Q. Zhang, M. Chen, A. Bukharin, N. Karampatziakis, P. He, Y. Cheng, W. Chen, and T. Zhao, *AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning*, in International Conference on Learning Representations, 2023.
- [22] S.-Y. Liu, C.-Y. Wang, H. Yin, P. Molchanov, Y.-C. F. Wang, K.-T. Cheng, and M.-H. Chen, *Dora: Weight-decomposed low-rank adaptation*, arXiv preprint arXiv:2402.09353 (2024).
- [23] Y. Mao, L. Mathias, R. Hou, A. Almahairi, H. Ma, J. Han, S. Yih, and M. Khabsa, *Unipelt: A unified framework for parameter-efficient language model tuning*, in Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 6253–6264, 2022.
- [24] X. Liu, K. Ji, Y. Fu, W. L. Tam, Z. Du, Z. Yang, and J. Tang, *P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks*, arXiv preprint arXiv:2110.07602 (2021).
- [25] S. Jie and Z.-H. Deng, *Fact: Factor-tuning for lightweight adaptation on vision transformer*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, pp. 1060–1068, 2023.
- [26] J. C. Spall, *Multivariate stochastic approximation using a simultaneous perturbation gradient approximation*, IEEE transactions on automatic control **37** (1992), no. 3 332–341.
- [27] Y. Nesterov and V. Spokoiny, *Random gradient-free minimization of convex functions*, Foundations of Computational Mathematics **17** (2017), no. 2 527–566.
- [28] Y. Liu, Z. Zhu, C. Gong, M. Cheng, C.-J. Hsieh, and Y. You, *Sparse mezo: Less parameters for better performance in zeroth-order llm fine-tuning*, arXiv preprint arXiv:2402.15751 (2024).
- [29] T. Gautam, Y. Park, H. Zhou, P. Raman, and W. Ha, *Variance-reduced zeroth-order methods for fine-tuning language models*, in International Conference on Machine Learning, pp. 15180–15208, PMLR, 2024.
- [30] S. Jiang, Q. Chen, Y. Pan, Y. Xiang, Y. Lin, X. Wu, C. Liu, and X. Song, *Zo-adamu optimizer: Adapting perturbation by the momentum and uncertainty in zeroth-order optimization*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, pp. 18363–18371, 2024.
- [31] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et. al., *Learning transferable visual models from natural language supervision*, in Proceedings of the 38th International Conference on Machine Learning (ICML), 2021.

- [32] T. Sun, Y. Shao, H. Qian, X. Huang, and X. Qiu, *Black-box tuning for language-model-as-a-service*, in International Conference on Machine Learning, pp. 20841–20855, PMLR, 2022.
- [33] C. Oh, H. Hwang, H.-y. Lee, Y. Lim, G. Jung, J. Jung, H. Choi, and K. Song, *Blackvip: Black-box visual prompting for robust transfer learning*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 24224–24235, 2023.
- [34] S. Park, J. Jeong, Y. Kim, J. Lee, and N. Lee, *ZIP: An efficient zeroth-order prompt tuning for black-box vision-language models*, in The Thirteenth International Conference on Learning Representations, 2025.
- [35] N. Hansen, S. D. Müller, and P. Koumoutsakos, *Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)*, Evolutionary computation **11** (2003), no. 1 1–18.
- [36] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, *Sharpness-aware minimization for efficiently improving generalization*, in International Conference on Learning Representations, 2021.
- [37] S. Han, J. Pool, J. Tran, and W. Dally, *Learning both weights and connections for efficient neural network*, Advances in neural information processing systems **28** (2015).
- [38] J. Frankle and M. Carbin, *The lottery ticket hypothesis: Finding sparse, trainable neural networks*, arXiv preprint arXiv:1803.03635 (2018).
- [39] R. J. Das, M. Sun, L. Ma, and Z. Shen, *Beyond size: How gradients shape pruning decisions in large language models*, arXiv preprint arXiv:2311.04902 (2023).
- [40] P. Dong, L. Li, Z. Tang, X. Liu, X. Pan, Q. Wang, and X. Chu, *Pruner-zero: Evolving symbolic pruning metric from scratch for large language models*, in Forty-first International Conference on Machine Learning, 2024.
- [41] S. Ashkboos, M. L. Croci, M. G. d. Nascimento, T. Hoeffler, and J. Hensman, *Sliceqpt: Compress large language models by deleting rows and columns*, arXiv preprint arXiv:2401.15024 (2024).
- [42] C. Liang, H. Jiang, Z. Li, X. Tang, B. Yin, and T. Zhao, *Homodistil: Homotopic task-agnostic distillation of pre-trained transformers*, arXiv preprint arXiv:2302.09632 (2023).
- [43] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao, *ReAct: Synergizing reasoning and acting in language models*, in International Conference on Learning Representations, 2023.

- [44] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, *Toolformer: Language models can teach themselves to use tools*, Advances in Neural Information Processing Systems **36** (2023).
- [45] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et. al., *Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning*, arXiv preprint arXiv:2501.12948 (2025).
- [46] M. Kang, J. Jeong, S. Lee, J. Cho, and S. J. Hwang, *Distilling llm agent into small models with retrieval and code tools*, arXiv preprint arXiv:2505.17612 (2025).
- [47] Y. Lyu, C. Wang, L. Shen, J. Huang, and T. Xu, *Mock worlds, real skills: Building small agentic language models with synthetic tasks, simulated environments, and rubric-based rewards*, arXiv preprint arXiv:2601.22511 (2026).
- [48] Y. Hu, W. Shi, X. Fu, D. Roth, M. Ostendorf, L. Zettlemoyer, N. A. Smith, and R. Krishna, *Visual sketchpad: Sketching as a visual chain of thought for multimodal language models*, Advances in Neural Information Processing Systems **37** (2024) 139348–139379.
- [49] X. Fu, M. Liu, Z. Yang, J. Corring, Y. Lu, J. Yang, D. Roth, D. Florencio, and C. Zhang, *Refocus: Visual editing as a chain of thought for structured image understanding*, in Proceedings of the 42nd International Conference on Machine Learning (ICML), 2025.
- [50] M. Wu, J. Yang, J. Jiang, M. Li, K. Yan, H. Yu, M. Zhang, C. Zhai, and K. Nahrstedt, *VTool-r1: VLMs learn to think with images via reinforcement learning on multimodal tool use*, in The Fourteenth International Conference on Learning Representations, 2026.
- [51] Z. Su, L. Li, M. Song, Y. Hao, Z. Yang, J. Zhang, G. Chen, J. Gu, J. Li, X. Qu, et. al., *Openthinking: Learning to think with images via visual tool reinforcement learning*, arXiv preprint arXiv:2505.08617 (2025).
- [52] X. Li, H. Zou, and P. Liu, *Torl: Scaling tool-integrated rl*, arXiv preprint arXiv:2503.23383 (2025).
- [53] D. Ma, Z. Yang, H. Xu, H. Fang, K. Yu, and L. Chen, *Empowering LLM tool invocation with tool-call reward model*, in The Fourteenth International Conference on Learning Representations, 2026.
- [54] Y. Yang, J. Zhou, N. Wong, and Z. Zhang, *LoRETTA: low-rank economic tensor-train adaptation for ultra-low-parameter fine-tuning of large language models*, in Proceedings of the 2024 Conference of the North American Chapter of

- the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pp. 3161–3176, 2024.
- [55] Y. Yang, K. Zhen, E. Banijamali, A. Mouchtaris, and Z. Zhang, *Adazeta: Adaptive zeroth-order tensor-train adaption for memory-efficient large language models fine-tuning*, in Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pp. 977–995, 2024.
- [56] Y. Yang, Z. Zhang, R. V. Swaminathan, J. Liu, N. Susanj, and Z. Zhang, *SharpZO: Hybrid sharpness-aware vision language model prompt tuning via forward-only passes*, in Advances in Neural Information Processing Systems, 2025.
- [57] Y. Yang, K. Zhen, B. Ganesh, A. Galstyan, G. Huybrechts, M. Müller, J. M. Kübler, R. V. Swaminathan, A. Mouchtaris, S. B. Bodapati, N. Susanj, Z. Zhang, J. FitzGerald, and A. Kumar, *Wanda++: Pruning large language models via regional gradients*, in Findings of the Association for Computational Linguistics: ACL 2025, (Vienna, Austria), pp. 4321–4333, Association for Computational Linguistics, July, 2025.
- [58] Y. Yang, Z. Zhang, J. Tian, L. Tan, and Z. Zhang, *IAPO: Input attribution-aware policy optimization for tool use in small multimodal agents*, 2026.
- [59] A. Chen, Y. Zhang, J. Jia, J. Diffenderfer, J. Liu, K. Parasyris, Y. Zhang, Z. Zhang, B. Kailkhura, and S. Liu, *Deepzero: Scaling up zeroth-order optimization for deep model training*, arXiv preprint arXiv:2310.02025 (2023).
- [60] I. V. Oseledets, *Tensor-train decomposition*, SIAM Journal on Scientific Computing **33** (2011), no. 5 2295–2317.
- [61] C. Hawkins, X. Liu, and Z. Zhang, *Towards compact neural networks via end-to-end training: A bayesian tensor approach with automatic rank determination*, SIAM Journal on Mathematics of Data Science **4** (2022), no. 1 46–71.
- [62] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM review **51** (2009), no. 3 455–500.
- [63] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, in International Conference on Learning Representations, 2019.
- [64] X. Liu, K. Ji, Y. Fu, W. Tam, Z. Du, Z. Yang, and J. Tang, *P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks*, in Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pp. 61–68, 2022.

- [65] E. B. Zaken, Y. Goldberg, and S. Ravfogel, *Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models*, in Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pp. 1–9, 2022.
- [66] H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. A. Raffel, *Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning*, Advances in Neural Information Processing Systems **35** (2022) 1950–1965.
- [67] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, *Glue: A multi-task benchmark and analysis platform for natural language understanding*, arXiv preprint arXiv:1804.07461 (2018).
- [68] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, *Superglue: A stickier benchmark for general-purpose language understanding systems*, Advances in neural information processing systems **32** (2019).
- [69] P. He, X. Liu, J. Gao, and W. Chen, *Deberta: Decoding-enhanced bert with disentangled attention*, in International Conference on Learning Representations, 2021.
- [70] M. Valipour, M. Rezagholizadeh, I. Kobyzev, and A. Ghodsi, *Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation*, arXiv preprint arXiv:2210.07558 (2022).
- [71] S. Ghiasvand, Y. Yang, Z. Xue, M. Alizadeh, Z. Zhang, and R. Pedarsani, *Communication-efficient and tensorized federated fine-tuning of large language models*, in Findings of the Association for Computational Linguistics: ACL 2025, (Vienna, Austria), pp. 24192–24207, Association for Computational Linguistics, July, 2025.
- [72] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, *Roberta: A robustly optimized bert pretraining approach*, arXiv preprint arXiv:1907.11692 (2019).
- [73] K. Zhou, J. Yang, C. C. Loy, and Z. Liu, *Learning to prompt for vision-language models*, International Journal of Computer Vision **130** (2022), no. 9 2337–2348.
- [74] K. Zhou, J. Yang, C. C. Loy, and Z. Liu, *Conditional prompt learning for vision-language models*, in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 16816–16825, 2022.
- [75] T. Sun, Z. He, H. Qian, Y. Zhou, X.-J. Huang, and X. Qiu, *Bbtv2: Towards a gradient-free future with large language models*, in Proceedings of the 2022

- Conference on Empirical Methods in Natural Language Processing, pp. 3916–3930, 2022.
- [76] S. Ghadimi and G. Lan, *Stochastic first-and zeroth-order methods for nonconvex stochastic programming*, SIAM journal on optimization **23** (2013), no. 4 2341–2368.
- [77] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need*, in Advances in Neural Information Processing Systems (NeurIPS), 2017.
- [78] Z. Wang, J. Liang, R. He, Z. Wang, and T. Tan, *Connecting the dots: Collaborative fine-tuning for black-box vision-language models*, in International Conference on Machine Learning, pp. 50931–50943, PMLR, 2024.
- [79] J. Zhou, Y. Yang, K. Zhen, Z. Liu, Y. Zhao, E. Banijamali, A. Mouchtaris, N. Wong, and Z. Zhang, *QuZO: Quantized zeroth-order fine-tuning for large language models*, in Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, (Suzhou, China), pp. 5341–5359, Association for Computational Linguistics, Nov., 2025.
- [80] Z. Zhang, Y. Yang, K. Zhen, N. Susanj, A. Mouchtaris, S. Kunzmann, and Z. Zhang, *MaZO: Masked zeroth-order optimization for multi-task fine-tuning of large language models*, in Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, (Suzhou, China), pp. 18526–18543, Association for Computational Linguistics, Nov., 2025.
- [81] C. Fang, A. Zhou, and Z. Wang, *An algorithm–hardware co-optimized framework for accelerating n : M sparse transformers*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems **30** (2022), no. 11 1573–1586.
- [82] T. Xie, Z. Gao, Q. Ren, H. Luo, Y. Hong, B. Dai, J. Zhou, K. Qiu, Z. Wu, and C. Luo, *Logic-rl: Unleashing llm reasoning with rule-based reinforcement learning*, arXiv preprint arXiv:2502.14768 (2025).
- [83] K. Zhang, Y. Zuo, B. He, Y. Sun, R. Liu, C. Jiang, Y. Fan, K. Tian, G. Jia, P. Li, et. al., *A survey of reinforcement learning for large reasoning models*, arXiv preprint arXiv:2509.08827 (2025).
- [84] X. Mai, H. Xu, Z.-Z. Li, W. Wang, J. Hu, Y. Zhang, W. Zhang, et. al., *Agent rl scaling law: Agent rl with spontaneous code execution for mathematical problem solving*, arXiv preprint arXiv:2505.07773 (2025).
- [85] S. Bai, K. Chen, X. Liu, J. Wang, W. Ge, S. Song, K. Dang, P. Wang, S. Wang, J. Tang, H. Zhong, Y. Zhu, M. Yang, Z. Li, J. Wan, P. Wang, W. Ding, Z. Fu, Y. Xu, J. Ye, X. Zhang, T. Xie, Z. Cheng, H. Zhang, Z. Yang, H. Xu, and J. Lin, *Qwen2.5-vl technical report*, 2025.

- [86] G. Sheng, C. Zhang, Z. Ye, X. Wu, W. Zhang, R. Zhang, Y. Peng, H. Lin, and C. Wu, *Hybridflow: A flexible and efficient rlhf framework*, in Proceedings of the Twentieth European Conference on Computer Systems, pp. 1279–1297, 2025.
- [87] L. Tan, Y. Zhao, Y. Yang, R. Zhang, X. Yu, and Z. Zhang, *Grzo: Group-relative zeroth-order optimization for large language model fine-tuning*, arXiv preprint arXiv:2606.02857 (2026).
- [88] J. Tian, R. Solgi, J. Lu, Y. Yang, H. Li, and Z. Zhang, *FLAT-LLM: Fine-grained low-rank activation space transformation for large language model compression*, in Findings of the Association for Computational Linguistics: EACL 2026, (Rabat, Morocco), pp. 2988–3002, Association for Computational Linguistics, Mar., 2026.
- [89] S. L. Lohr, Sampling: design and analysis. Nelson Education, 2009.
- [90] S. Liu, B. Kailkhura, P.-Y. Chen, P. Ting, S. Chang, and L. Amini, *Zeroth-order stochastic variance reduction for nonconvex optimization*, Advances in Neural Information Processing Systems **31** (2018).

Proof for Theorem 3.1

To provide the proof of Theorem 3.1, we first present a Lemma regarding the bound of gradient noise. Recall from the gradient estimation rule that:

$$\nabla \hat{\ell}(\mathbf{w}_k) = \frac{1}{B} \sum_{b_i \in \mathcal{B}} \hat{g}(\mathbf{w}_k; b_i) \quad (.1)$$

$$\hat{g}(\mathbf{w}_k; b_i) = \frac{1}{Q_k} \sum_{j=1}^{Q_k} \hat{g}(\mathbf{w}_k; b_i, u_{i,j}), \quad (.2)$$

where there are two sources of randomness: a) The randomness leads by the mini-batch sampling and b) The randomness leads by the presence of ZO gradient estimation. Based on these two randomnesses, we define two gradient noises as h_k and e_k , respectively.

$$h_k := \nabla \hat{\ell}(\mathbf{w}_k) - \nabla \ell(\mathbf{w}_k) = \frac{1}{B} \sum_{b_i \in \mathcal{B}} \hat{g}(\mathbf{w}_k; b_i) - \nabla \ell(\mathbf{w}_k) \quad (.3)$$

$$e_k := \hat{g}(\mathbf{w}_k; b_i) - \nabla \ell(\mathbf{w}_k) = \frac{1}{Q_k} \sum_{j=1}^{Q_k} \hat{g}(\mathbf{w}_k; b_i, u_{i,j}) - \nabla \ell(\mathbf{w}_k) \quad (.4)$$

Here, we first bound the gradient noise h_k with a fact given in stochastic gradient descent theory. We consider the noise concerning the mean of the ZO estimated gradient $\nabla \ell(\mathbf{w}_k)$, where the loss function ℓ is a randomized smoothing version of ℓ .

Lemma 1. *Based on the definition in eq. (.3) and the Assumption A2, we can bound the L2-norm of the gradient noise h_k by taking expectation:*

$$\mathbb{E}[\|h_k\|^2] \leq \frac{N-B}{NB(B-1)Q_k} \sum_i (2d\delta^2 + \frac{\epsilon^2 L^2 d^2}{2} + 2\delta^2) \quad (.5)$$

Proof. For convenience, we consider a general case that the mini-batch $\mathcal{B}$ is formed by uniform sampling without replacement and follows the i.i.d. fashion. Then, according to [89][Section 2.8, Page 48], the following holds for a random sampling noise:

$$\mathbb{E}[\|h_k\|^2] = \frac{N-B}{NB} \Lambda^2, \quad (.6)$$

where Λ^2 is the sample variance of the gradient $\hat{g}(\mathbf{w}_k; b_i)$, which is defined as:

$$\Lambda^2 = \frac{1}{B-1} \sum_{i=1}^B \|\hat{g}(\mathbf{w}_k; b_i) - \nabla \ell(\mathbf{w}_k)\|^2 \quad (.7)$$

$$= \frac{1}{B-1} \sum_{i=1}^B \|\nabla \ell(\mathbf{w}_k) + e_k - \nabla \ell(\mathbf{w}_k)\|^2 \quad (.8)$$

$$= \frac{1}{B-1} \sum_{i=1}^B \|e_k\|^2, \quad (.9)$$

where e_k is defined as the gradient noise leads by the ZO estimation in eq. (.3).

Finally, we need to bound the variance Λ^2 , related to the ZO gradient estimation noise. Taking expectation with respect to the i.i.d. random perturbation vector $\mathbf{u}$, we have:

$$\mathbb{E}_{\mathbf{u}}[\Lambda^2] \leq \mathbb{E}_{\mathbf{u}}\left[\frac{1}{B-1} \sum_{i=1}^B \|e_k^i\|^2\right] \quad (.10)$$

$$\leq \frac{1}{(B-1)Q_k^2} \sum_i \mathbb{E}_{\mathbf{u}}\left[\left\|\sum_{j=1}^{Q_k} (\hat{g}(\mathbf{w}_k; b_i, u_{i,j}) - \nabla \ell(\mathbf{w}_k))\right\|^2\right] \quad (.11)$$

$$\stackrel{(a)}{=} \frac{1}{(B-1)Q_k} \sum_i \mathbb{E}_{\mathbf{u}}[\|\hat{g}(\mathbf{w}_k; b_i, u_{i,1}) - \nabla \ell(\mathbf{w}_k)\|^2], \quad (.12)$$

where (a) is given under the case that $u_{i,j}$ is i.i.d, which obtain:

$$\mathbb{E}_{\mathbf{u}}[\|\hat{g}(\mathbf{w}_k; b_i, u_{i,j}) - \nabla \ell(\mathbf{w}_k)\|^2] = \mathbb{E}_{\mathbf{u}}[\|\hat{g}(\mathbf{w}_k; b_i, u_{i,1}) - \nabla \ell(\mathbf{w}_k)\|^2] \quad (.13)$$

Finally, we need to bound the term $\mathbb{E}_{\mathbf{u}}[\|\hat{g}(\mathbf{w}_k; b_i, u_{i,1}) - \nabla \ell(\mathbf{w}_k)\|^2]$, which gives:

$$\mathbb{E}_{\mathbf{u}}[\|\hat{g}(\mathbf{w}_k; b_i, u_{i,1}) - \nabla \ell(\mathbf{w}_k)\|^2] \quad (.14)$$

$$\leq \mathbb{E}_{\mathbf{u}}[\|\hat{g}(\mathbf{w}_k; b_i, u_{i,1}) - \nabla \ell(\mathbf{w}_k; b_i)\|^2] + \mathbb{E}_{\mathbf{u}}[\|\nabla \ell(\mathbf{w}_k; b_i) - \nabla \ell(\mathbf{w}_k)\|^2] \quad (.15)$$

$$\stackrel{(a)}{\leq} 2d\|\hat{g}(\mathbf{w}_k; b_i, u_{i,1})\|^2 + \frac{\epsilon^2 L^2 d^2}{2} + \mathbb{E}_{\mathbf{u}}[\|\nabla \ell(\mathbf{w}_k)\|^2] + \mathbb{E}_{\mathbf{u}}[\|\nabla \ell(\mathbf{w}_k; b_i)\|^2] \quad (.16)$$

$$\stackrel{(b)}{\leq} 2d\delta^2 + \frac{\epsilon^2 L^2 d^2}{2} + 2\delta^2, \quad (.17)$$

where (a) follows a similar idea of the proof in [76][eq. (3.21)] and (b) is given by using the bound of the gradient in Assumption A2.

Putting it all together we can obtain the upper bound for the gradient noise $\|h_k\|$. $\square$

Now we begin to present the proof of Theorem 3.1:

We start from the gradient updating rule in the AdaZeta algorithm, which gives $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla \hat{\ell}(\mathbf{w}_k)$. By using Taylor's theorem on the exact smoothed loss $\ell(\mathbf{w}_k)$, we have:

$$\ell(\mathbf{w}_{k+1}) = \ell(\mathbf{w}_k - \eta \nabla \hat{\ell}(\mathbf{w}_k)) \quad (.18)$$

$$= \ell(\mathbf{w}_k) - \eta \nabla \hat{\ell}(\mathbf{w}_k)^\top \nabla \ell(\mathbf{w}_k) + \frac{\eta^2}{2} \nabla \hat{\ell}(\mathbf{w}_k)^\top \nabla^2 \ell(\mathbf{w}_k) \nabla \hat{\ell}(\mathbf{w}_k) \quad (.19)$$

Taking expectations on both sides gives:

$$\begin{aligned} \mathbb{E}_{\mathbf{w}_t}[\ell(\mathbf{w}_{k+1})] &= \mathbb{E}_{\mathbf{w}_t}[\ell(\mathbf{w}_k)] - \eta \mathbb{E}_{\mathbf{w}_t}[\nabla \hat{\ell}(\mathbf{w}_k)^\top \nabla \ell(\mathbf{w}_k)] + \frac{\eta^2}{2} \mathbb{E}_{\mathbf{w}_t}[\nabla \hat{\ell}(\mathbf{w}_k)^\top \nabla^2 \ell(\mathbf{w}_k) \nabla \hat{\ell}(\mathbf{w}_k)] \\ &\stackrel{(a)}{\leq} \mathbb{E}_{\mathbf{w}_t}[\ell(\mathbf{w}_k)] - \eta \mathbb{E}_{\mathbf{w}_t}[\nabla \ell(\mathbf{w}_k)^2] + \frac{\eta^2 L}{2} \mathbb{E}_{\mathbf{w}_t}[\nabla \hat{\ell}(\mathbf{w}_k)^2], \end{aligned}$$

where (a) can be proved with the use of the Lipschitz smoothness gradient denied in Assumption A1 that gives x and y , we have $\|\nabla \ell(x) - \nabla \ell(y)\| \leq L\|x - y\|$. Additionally, by the mean value theorem for vector-valued functions, there exists for any point c on the line segment between x and y such that:

$$\nabla f(y) - \nabla f(x) = \nabla^2 f(c)(y - x). \quad (.20)$$

Taking the norms on both sides and using the Lipschitz condition, we have:

$$\|\nabla^2 f(c)(y - x)\| = \|\nabla f(y) - \nabla f(x)\| \leq L\|y - x\|. \quad (.21)$$

Finally, since this must hold for any y and x , and since the norm of the Hessian matrix is the supremum of $\|V^2 f(c)(y - x)\| / \|y - x\|$ for non-zero $y - x$, it follows that:

$$\|\nabla^2 f(c)\| \leq L \quad (.22)$$

Rearrange and we obtain:

$$\eta \mathbb{E}[\|\nabla \ell(\mathbf{w}_k)\|^2] \leq \mathbb{E}[\ell(\mathbf{w}_k)] - \mathbb{E}[\ell(\mathbf{w}_{k+1})] + \frac{\eta^2 L}{2} \mathbb{E}[\nabla \hat{\ell}(\mathbf{w}_k)^2] \quad (.23)$$

Taking summation over steps $k = 1, \dots, K$ gives:

$$\sum_{k=1}^K \eta \mathbb{E}[\|\nabla \ell(\mathbf{w}_k)\|^2] \leq \mathbb{E}[\ell(\mathbf{w}_0) - \ell(\mathbf{w}_K)] + \sum_{k=1}^K \frac{\eta^2 L}{2} \mathbb{E}[\nabla \hat{\ell}(\mathbf{w}_k)^2] \quad (.24)$$

$$\stackrel{(a)}{\leq} \mathbb{E}[\ell_0 - \ell^*] + \epsilon^2 L + \sum_{k=1}^K \frac{\eta^2 L}{2} \mathbb{E}[\nabla \hat{\ell}(\mathbf{w}_k)^2] \quad (.25)$$

$$\stackrel{(b)}{\leq} R + \epsilon^2 L + \sum_{k=1}^K \frac{\eta^2 L}{2} \mathbb{E}[\nabla \hat{\ell}(\mathbf{w}_k)^2], \quad (.26)$$

where (a) is using the Lemma 1 in [90] that $\ell(\mathbf{w}_0) - \ell(\mathbf{w}_T) \leq \ell(\mathbf{w}_0) - \ell(\mathbf{w}_0) + \ell^* - \ell^* \leq (\ell(\mathbf{w}_0) - \ell^*) + \epsilon^2 L$ and (b) is given by setting $R := \ell(\mathbf{w}_1) - \ell^*$. Now, the key to the bound comes from the last term in the right of the inequation.

To bound the last term, we first represent the noise gradient $\nabla \hat{\ell}_k(\mathbf{w}_k)^2$ as a combination of the true gradient and the gradient noise introduced in eq. (.3), which gives:

$$\nabla \hat{\ell}(\mathbf{w}_k) := \nabla \ell(\mathbf{w}_k) + h_k \quad (.27)$$

Taking eq. (.27) back into eq. (.24), using the results from Lemma 1, taking the expectation over all randomness and average over the maximum steps K , we obtain:

$$\begin{aligned} & \frac{1}{K} \sum_{k=1}^K \eta \mathbb{E}[\|\nabla \ell(\mathbf{w}_k)\|^2] \\ & \leq \frac{R}{K} + \frac{\epsilon^2 L}{K} + \frac{1}{K} \sum_{k=1}^K \frac{\eta^2 L}{2} \mathbb{E}[\nabla \hat{\ell}(\mathbf{w}_k)^2] \\ & \leq \frac{R}{K} + \frac{\epsilon^2 L}{K} + \frac{1}{K} \sum_{k=1}^K \frac{\eta^2 L}{2} \mathbb{E}[\|\nabla \ell(\mathbf{w}_k)\| + \|h_k\|] \\ & = \frac{R}{K} + \frac{\epsilon^2 L}{K} + \frac{\eta^2 L \delta}{2} + \frac{1}{K} \sum_{k=1}^K \frac{\eta^2 L}{2} \frac{N-B}{NB} \Lambda^2 \\ & \leq \frac{R}{K} + \frac{\epsilon^2 L}{K} + \frac{\eta^2 L \delta}{2} + \frac{1}{K} \sum_{k=1}^K \frac{\eta^2 L}{2} \frac{N-B}{NB} \left(\frac{1}{(B-1)Q_k} \sum_i \mathbb{E}_{\mathbf{u}}[\|e^{i,1}\|] + \frac{B\epsilon^2 L}{2(B-1)} \right) \end{aligned}$$

$$\begin{aligned}
&\leq \frac{R}{K} + \frac{\epsilon^2 L}{K} + \frac{\eta^2 L \delta}{2} + \frac{1}{K} \sum_{k=1}^K \frac{\eta^2 L}{2} \frac{N-B}{NB} \left(\frac{\sum_i (2d\delta^2 + \frac{\epsilon^2 L^2 d^2}{2} + 2\delta)}{(B-1)Q_k} + \frac{B\epsilon^2 L}{2(B-1)} \right) \\
&= \frac{R + \epsilon^2 L + C(d, \epsilon) \sum_k \frac{1}{Q_k}}{K} + \frac{\eta^2 L \delta}{2} + \frac{B\epsilon^2 L}{2(B-1)} \\
&= \mathcal{O}\left(\frac{R + \epsilon^2 L + C(d, \epsilon) \sum_k \frac{1}{Q_k}}{K}\right),
\end{aligned}$$

where $C(d, \epsilon)$ is a constant defined as $C(d, \epsilon) := \sum_{k=1}^K \frac{\eta^2 L}{2} \frac{N-B}{NB} \left(\frac{\sum_i (2d\delta^2 + \frac{\epsilon^2 L^2 d^2}{2} + 2\delta)}{(B-1)} \right)$

Divide both side with η and use the trick to introduce some randomly chosen $\mathbf{w}_T$ from the history with probability $P(T = k) = \frac{1}{K}$, we finish the proof as:

$$\mathbb{E}[\|\nabla \ell(\mathbf{w}_T)\|^2] = \frac{1}{K} \sum_{k=1}^K \mathbb{E}[\|\nabla \ell(\mathbf{w}_k)\|^2] \leq \mathcal{O}\left(\frac{R + \epsilon^2 L + C(d, \epsilon) \sum_k \frac{1}{Q_k}}{K\epsilon}\right) \quad (.28)$$