INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2025IPPAT054

Thèse de doctorat

Energy and Memory-Efficient Artificial Intelligence for On-Device Learning

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat: Mathématiques et Informatique

Thèse présentée et soutenue à Palaiseau, le 03/12/2025, par

Aël QUELENNEC

Composition du Jury :

Amel Bouzeghoub Professeur, Télécom SudParis	Présidente Examinatrice
Ngoc-Son Vu Maître de Conférences, ENSEA	Rapporteur
Dewei Yi Professeur Associé, Université d'Aberdeen	Rapporteur
Umut Şimşekli Chargé de Recherche, INRIA	Examinateur
Van-Tam Nguyen Professeur, Télécom Paris (LTCI)	Directeur de thèse
Pavlo Mozharovskyi Professeur, Télécom Paris (LTCI)	Co-encadrant de thèse
Enzo Tartaglione Professeur Associé, Télécom Paris (LTCI)	Co-encadrant de thèse Invité
Vito Paolo Pastore Professeur Assistant, MaLGa	Invité

This page is intentionally left blank.

Acknowledgments

"*No one accomplishes anything in this life on his or her own*"

Allan Hamilton

Being a PhD student can sometimes be a solitary task as you are typically the only one to really understand what you are doing (that is, when you understand anything at all). Despite this, I was never alone throughout this adventure and I received overwhelming support from all walks of life. I wish to express my gratitude to all those who were at my side at one point or another and helped me grow both personally and academically during these last three years.

I wish to begin by thanking Van-Tam, Enzo and Pavlo, my thesis supervisors. Throughout my PhD they trusted me to explore my own research directions with autonomy while providing solid advice and guidance, preventing me from getting lost in endless experimental loops. These three years have been a great opportunity for me to develop my critical thinking and knowledge, and I am grateful to the three of you for it.

I would also like to thank Dr. Ngoc-Son Vu and Dr. Dewei Yi for accepting to review my work and helping me improve it. Many thanks to the other jury members, Pr. Amel Bouzeghoub, Dr. Umut Şimşekli and Dr. Vito Paolo Pastore for taking interest in my research and joining my thesis committee.

During my PhD I shared an office with Rémi, Yinghao and Zhu, and I am deeply grateful for the bond we have built over these three years. I cherished our times sharing food, trying to say words in each other's language and helping each other out. Special thanks to Rémi for our sports sessions and all the time spent debating about anything and everything. My thoughts also go to the PhD track students which I helped supervise, especially Trung as I greatly enjoyed collaborating with you and witnessing your incredible growth in only one year.

My life at Télécom predates my PhD, and during my engineering studies I had the chance to meet so many great people that I am now proud to call my friends. Many thanks to the members of the soberly named messenger conversation "Films": Agathe, Alex, Amélie, Chloé, Elisa, Fanny, Geoffroy, Hiba, Julie, Juliette, Marion, Maxime, Oumar, Raphaël and Victor. I remember fondly our time spent together and from treks to weddings, there is no occasion that would stop us from talking about nerdy engineering stuff. Special mention to my favorite roommate/neighbour Domitille for the lengthy talks and the laughs making strange noises. If we go by messenger conversation names, how could I not talk about the 701 gang: Anton, Axel, Basile, Eric, Gaspard, Maxence, Nathan and Raphaël, for all our trips, our gaming sessions until the end of the night, our time spent trying to solve

mathematical enigmas and talking about PhD stuff. I would not have it any other way. More specifically, I express my warmest gratitude to Zachary for all the help he provided, from the manuscript and defence templates to the administrative guidance and preparation of the celebration.
Rewinding time even before engineering studies, I express my profound gratitude to the Boule De Potes, an amalgamation of friends from high school—Charlotte, Emma, Liam, Louis, Lucas and Nathan—alongside people brought in by Liam through his studies at ISRP: Adèle, Quentin, Suzy and Thomas. Some may say that there is no stronger bond than the one shared over a winter raclette after playing "plaît-il" all ski day long despite the urgency of reaching the last cable car in the middle of a snow storm. They would be absolutely correct, and I am eager to renew that bond at any time. I also welcome promising new members of this group, Lilian and Solenn, for always being so thoughtful and supportive.
Finally, I express my thankfulness to the members of my family for being a steady and reliable source of comfort, strength and advice. Mom and Dad, for teaching me that an analytical mind is not incompatible with love and care. Aliénor, for your artistic spirit, your cheerfulness and our improvised climbing sessions. Aurian and Louise, for the impromptu navette trips and for kindly welcoming me in your flat whenever I needed a listening ear. To my grandparents Gabrielle, Michel and Monique, my uncle and aunts Anne and Erwan, Isabelle and Yacine, Thierry and Marianne, my cousins Rafaël and Nathanaël, for keeping on putting your faith in me despite what I was doing probably sounding obscure to you. To my girlfriend's family for taking me in as one of their own—Catherine, Lea, Enzo and the rest of the Darios—I look forward to sharing again your passion for rugby.
Last but not least, I express my utmost gratitude to the one with whom I share my daily life, who has seen me go through sleepless nights to finalize paper submissions, watched me connect to computational resources from anywhere at any time to launch experiments, and heard me complain when I would spend hours trying to figure out why my code was not working. Countless times have I told her that "this time the results should be good, I really think I am onto something" only to witness my immediate disappointment when plotting some performance metric. I could never thank you enough Valentine for staying by my side and supporting me during these past three years and for your unwavering trust in my capacities.

Paris 14, Octobre 2025

Aël Quélennec

Abstract

This thesis addresses two fundamental bottlenecks preventing practical on-device learning through strategic subnetwork selection for efficient fine-tuning and activation map compression for memory-efficient training. Both approaches aim at enabling continuous learning and adaptation of neural networks on edge devices with severely constrained resources.

The first part introduces dynamic methods that adaptively identify the most important network components for updating at each training step. Beginning with Velocity metric-based neuron selection, we advance to a comprehensive framework incorporating heavy-tailed gradient theory for layer and channel selection. This yields **Tra**ining **Dy**namics (TraDy), both a gradient study framework and an algorithm that dynamically selects efficient subnetworks while respecting strict memory constraints. We further develop **Me**mory-constrained **Dy**namic Upd**ate** (MeDyate), an adaptive channel selection strategy introducing stochasticity and sampling mechanisms to improve performance while maintaining memory efficiency. Experimental validation across multiple architectures and datasets demonstrates state-of-the-art performance in memory-constrained fine-tuning scenarios.

The second part addresses the memory bottleneck in backpropagation through tensor decomposition-based activation map compression. We identify activation storage as the primary memory constraint during training and propose compression using High-Order Singular Value Decomposition (HOSVD) with controlled information loss and theoretical convergence guarantees. To overcome HOSVD's computational overhead, we develop ASI (Activation Subspace Iteration), which leverages short-term stability of activation maps during training. By performing rank selection once before training and utilizing single subspace iterations with warm starts, ASI achieves significant memory reduction (up to 120× compression) and computational speedup (up to 91× faster than HOSVD) while maintaining comparable performance.

Theoretical contributions include formal analysis of fine-tuning dynamics through heavy-tailed gradient theory, convergence guarantees for compressed activation training, and computational complexity analysis of tensor decomposition methods. Extensive experimental validation demonstrates practical effectiveness across diverse architectures (MobileNet to transformers), datasets (ImageNet, CIFAR, Oxford Pets), and real-world deployment scenarios including Raspberry Pi implementations.

Résumé

L'apprentissage embarqué permet aux réseaux de neurones de s'adapter continuellement sur des dispositifs en *edge*, offrant une meilleure confidentialité des données, une latence réduite et une efficacité énergétique améliorée par rapport aux approches nécessitant un recours systématique au cloud. Dans ce contexte, le transfert de connaissances depuis des modèles pré-entraînés vers de nouvelles tâches applicatives constitue une stratégie centrale, permettant de tirer parti de représentations génériques apprises sur de larges jeux de données. Cependant, les ressources mémoire et computationnelles limitées des dispositifs embarqués posent des défis considérables, en particulier durant la phase de rétropropagation du gradient qui correspond à un coût en calcul conséquent et qui exige de stocker l'intégralité des cartes d'activation intermédiaires. Cette thèse aborde ces verrous à travers deux approches complémentaires et orthogonales : la sélection stratégique de sous-réseaux pour un fine-tuning efficace, et la compression des cartes d'activation pour un apprentissage économe en mémoire.
La première ligne de travail s'inscrit dans le paradigme de la sélection dynamique de sous-réseaux. Plutôt que de mettre à jour l'ensemble des paramètres d'un modèle, ces méthodes identifient de manière adaptative les composants les plus informatifs à chaque étape d'entraînement, réduisant ainsi l'empreinte mémoire liée au calcul et au stockage des gradients. Un premier travail explore une stratégie de sélection basée sur la dynamique d'apprentissage des neurones, en utilisant leur vélocité comme indicateur d'importance, posant ainsi les fondements d'approches plus élaborées. Nous développons ensuite **Tra**ining **Dy**namics (TraDy), un cadre théorique et algorithmique s'appuyant sur la théorie des gradients à queues lourdes (*heavy-tailed*) pour caractériser formellement les dynamiques de fine-tuning et guider la sélection dynamique de sous-réseaux. Cette théorie, initialement appliquée à l'analyse de la généralisation des réseaux de neurones, est ici exploitée pour justifier la sparsité naturelle des gradients durant le transfert d'apprentissage. Enfin, nous proposons **Me**mory-constrained **Dy**namic Upd**ate** (MeDyate), une stratégie de sélection adaptative de canaux intégrant une composante stochastique et des mécanismes d'échantillonnage pour mieux explorer l'espace des configurations possibles tout en respectant des contraintes mémoire strictes. La validation expérimentale de ces méthodes, conduite sur diverses architectures et jeux de données, démontre des performances à l'état de l'art pour le fine-tuning sous contraintes mémoire strictes.
La seconde ligne de travail traite du goulot d'étranglement mémoire engendré par le stockage des activations lors de la rétropropagation, en proposant une compression de ces tenseurs par décomposition tensorielle. Nous développons dans un premier temps une approche fondée sur la décomposition en valeurs singulières

d'ordre élevé, la *High-Order Singular Value Decomposition* (HOSVD), permettant de compresser les cartes d'activation avec une perte d'information maîtrisée. Ce travail fournit des garanties théoriques de convergence pour l'entraînement conduit avec des activations compressées, ainsi qu'une analyse rigoureuse de la complexité computationnelle associée et de l'erreur introduite dans le calcul des gradients. Néanmoins, le coût computationnel inhérent à l'application de HOSVD à chaque itération d'entraînement limite son applicabilité pratique sur des dispositifs à ressources contraintes. Pour surmonter cette limitation, nous développons ASI (*Activation Subspace Iteration*), une méthode qui exploite la stabilité à court terme des sous-espaces des cartes d'activation au cours de l'entraînement. En effectuant la sélection de rang une unique fois avant l'entraînement et en recourant à des itérations de sous-espaces simplifiées avec initialisation à chaud (*warm start*), ASI approxime efficacement la décomposition tensorielle complète à moindre coût. Les résultats expérimentaux démontrent des gains substantiels : jusqu'à 120× de compression mémoire, 1,86× d'accélération de l'entraînement, et 91× de réduction de la latence par rapport à HOSVD sur matériel embarqué, tout en préservant des performances comparables à l'entraînement standard.

Les contributions théoriques de cette thèse comprennent l'analyse formelle des dynamiques de fine-tuning au travers de la théorie des gradients à queues lourdes, l'établissement de garanties de convergence pour l'entraînement avec activations compressées, ainsi que l'analyse de la complexité algorithmique des méthodes de décomposition tensorielle intégrées dans la boucle d'entraînement. L'ensemble de ces travaux fait l'objet d'une validation expérimentale extensive sur des architectures variées, allant de réseaux compacts tels que MobileNet jusqu'à des modèles de type transformeur, sur des jeux de données de référence incluant ImageNet, CIFAR, Oxford Pets, ainsi que des scénarios applicatifs réels impliquant des déploiements sur Raspberry Pi. Cette double approche, combinant sélection de sous-réseaux et compression d'activations, offre ainsi une boîte à outils complémentaire et rigoureusement fondée pour l'apprentissage embarqué efficace en mémoire.

Contents

List of Figures

List of Tables

Publications

In this PhD thesis we present the following publications:

C1 **Aël Quélennec**, Enzo Tartaglione, Pavlo Mozharovskyi and Van-Tam Nguyen. "Towards on-device learning on the edge: Ways to select neurons to update under a budget constraint". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2024. **Oral Presentation**

C2 **Aël Quélennec**, Nour Hezbri, Pavlo Mozharovskyi, Van-Tam Nguyen and Enzo Tartaglione. "Study of Training Dynamics for Memory-Constrained Fine-Tuning". In: *International Conference on Learning Representations (ICLR)*. 2026. **Under Review**

C3 **Aël Quélennec**, Pavlo Mozharovskyi, Van-Tam Nguyen and Enzo Tartaglione. "Memory Constrained Dynamic Subnetwork Update for Transfer Learning". In: *Transactions on Machine Learning Research*. 2026. **Under Review**

C4 Le-Trung Nguyen, **Aël Quélennec**, Enzo Tartaglione, Samuel Tardieu and Van-Tam Nguyen. "Activation Map Compression through Tensor Decomposition for Deep Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2024. **Poster Presentation**

C5 Le-Trung Nguyen, **Aël Quélennec**, Van-Tam Nguyen and Enzo Tartaglione. "Beyond Low-rank Decomposition: A Shortcut Approach for Efficient On-Device Learning". In: *International Conference on Machine Learning (ICML)*. 2025. **Poster Presentation**

Publications C1 to C3 constitute a personal research line focused on efficient subnetwork selection under memory constraints and are presented in detail in Part I. Publications C4 and C5 were developed in collaboration with Le-Trung Nguyen during my involvement in his PhD track and internship supervision. For publication C4, my contributions encompassed establishing the literature review, developing the activation tensor decomposition methodology with efficient derivative computation in the decomposed subspace, and conducting the error analysis. Le-Trung focused on reproducing state-of-the-art baselines and designing and executing experiments to evaluate our method's performance. The manuscript was co-written collaboratively.
For publication C5, I proposed adapting the PowerSGD approach from Vogels et al.

[2019] to our activation compression framework. I guided Le-Trung through the conceptual development of adapting the perplexity metric to our activation setup and the design of the rank selection algorithm. Le-Trung authored the majority of the article, while I provided detailed feedback to enhance the overall writing quality and clarity.

Chapter 1

Introduction and Motivations

"*En route mauvaise troupe !*"

Raoul Comte

Contents

1.1 Thesis Objectives

The rapid evolution of artificial intelligence and deep learning has enabled unprecedented capabilities across diverse domains, from computer vision (Hu et al. [2018]) to natural language processing (Radford et al. [2018]). However, the deployment of these powerful models on resource-constrained edge devices presents significant challenges that limit their practical applicability in real-world scenarios (Dhar et al. [2021]). This thesis addresses a fundamental question in modern machine learning: *how can we enable efficient on-device learning while respecting the severe memory and computational constraints of embedded systems?*

Neural Decoding Context. This research originated within a broader initiative focused on developing energy and memory-efficient AI solutions for on-device neural decoding (Glaser et al. [2020]; Wang et al. [2024]). Neural decoding refers to the process of interpreting brain activity patterns to infer cognitive states or intentions, a field that has gained significant traction with advances in machine learning applied to neurophysiological signals (Nicolas-Alonso and Gomez-Gil [2012]). Our work would enable electroencephalographic (EEG)-based applications, where non-invasive sensors capture electrical brain activity through electrodes placed on the scalp (Kirschstein and Köhling [2009]). The ultimate goal involves deploying cognitive state classification algorithms—such as attention level detection or mental workload assessment—directly on EEG headsets. This enables real-time neurofeedback systems, where users receive immediate feedback about their cognitive state to facilitate training and enhancement of specific mental capacities, such as sustained attention or cognitive control (Hammond [2007]). Such on-device processing offers the following compelling advantages.

- **Enhanced Privacy and Security.** Processing sensitive neurophysiological data locally eliminates the need for transmitting personal information to external servers, significantly reducing privacy risks and ensuring compliance with data protection regulations (Pujari et al. [2023]; Tonga et al. [2022]).
- **Adaptive User Personalization.** On-device learning capabilities enable continuous adaptation to individual users' unique neural patterns, potentially improving classification performance over time through personalized model refinement (Ouyang et al. [2024]; Wen et al. [2024]).
- **Reduced Energy Consumption.** Local processing minimizes energy overhead associated with wireless data transmission and cloud-based computation, contributing to more sustainable and practical wearable AI solutions (Fraga-Lamas et al. [2021]; Wang et al. [2025]).
- **Network Independence.** By processing data locally, the system operates

independently of network connectivity, ensuring reliable functionality in various environmental conditions (Li et al. [2019]; Xu et al. [2021]).

On-Device Learning Motivation. Beyond the specific EEG application context, this thesis addresses fundamental bottlenecks that prevent practical implementation of on-device learning across various embedded AI applications. The severe constraints imposed by edge devices create formidable challenges for deploying sophisticated deep learning models that were originally designed for resource-rich environments (Meuser et al. [2024]).
The emergence of the Internet of Things (IoT) and edge computing paradigms has amplified the importance of efficient on-device processing capabilities. Modern applications demand intelligent systems that can adapt and learn continuously without relying on centralized infrastructure, necessitating novel approaches to memory-efficient training and inference (Incel and Bursa [2023]).

1.2 Overview of On-device Learning Challenges

As Sevilla et al. [2022] points out, the remarkable advances in deep learning performance over the past decade have been largely driven by the relentless expansion of model scale and complexity. This growth trajectory, characterized by parameter counts that have increased exponentially since the emergence of large-scale neural architectures, introduces fundamental tensions between computational requirements and practical deployment realities. The substantial energy demands for training and inference (Desislavov et al. [2021]), coupled with the necessity for aggressive model compression to enable real-world deployment (Dantas et al. [2024]), create a widening gap between what is theoretically possible and what is practically achievable on resource-constrained devices. We present below the key challenges that must be addressed for efficient on-device deployment.
Memory Constraints. Traditional deep learning workflows typically assume abundant computational resources and memory capacity. However, edge devices operate under drastically different conditions, with only kilobytes to megabytes of RAM and one order of magnitude more storage capacity (Jhang et al. [2021]).
The memory requirements for neural network training comprise several components, the most straightforward being *parameter storage*. Where modern deep networks contain millions to billions of parameters, each requiring storage space for weights and associated optimization states (momentum, running averages, etc.), several lines of research aim at reducing the memory footprint of network parameters for inference (Liang et al. [2021]). We provide an overview of these approaches in Sec. 2.1.
For training, Linnainmaa [1970]'s backpropagation algorithm remains the most

widely adopted strategy but is significantly more resource-intensive than inference (Li et al. [2012]). This stems from the necessity of forward propagation, intermediate storage of activation maps, loss backpropagation, and weight derivative computation.

Activation Maps. They refer to the intermediate values that propagate through neural network layers (Kılıçarslan et al. [2021]). During inference, these values can be dynamically discarded after processing, thus not representing a significant memory burden. However, backpropagation requires storing all activation maps in memory during the forward pass, as they are essential for gradient computation. Cai et al. [2020] demonstrate that their memory footprint typically far exceeds that of network parameters, making their compression a critical challenge for on-device learning.

Computational Efficiency. Edge devices typically feature specialized low-power processors with limited computational throughput compared to datacenter GPUs 'Zhu et al. [2021]). This constraint necessitates algorithms that achieve acceptable performance within strict computational budgets. Furthermore, energy efficiency becomes indispensable in battery-powered applications, where computational complexity directly impacts device longevity (Zhu et al. [2022]).

Data Drift. Static models deployed on edge devices suffer from performance degradation over time due to data drift (Agrahari and Singh [2022]; Sahiner et al. [2023]), where the distribution of real-world data diverges from the training distribution. This issue is particularly pronounced in personalized applications like EEG-based systems, where individual differences and temporal variations can significantly impact model performance (Pontes et al. [2024]).

1.3 Contributions and Research Directions Explored

This thesis presents two complementary research directions that address distinct aspects of the on-device learning challenge. Both approaches are grounded in rigorous theoretical analysis and validated through comprehensive experimental evaluation.

Subnetwork Selection for Efficient Fine-Tuning. The first research direction, presented in Part I, focuses on developing precise parameter selection strategies that identify the most critical network components for updating during fine-tuning under severe memory constraints. Rather than updating entire networks, we propose dynamic approaches that adapt to the specific characteristics of both downstream tasks and network architectures.

Chapter 3 establishes the foundation for subnetwork selection by exploring an initial dynamic neuron selection strategy based on our proposed importance metric termed *Velocity*. While this preliminary approach revealed important insights

about the benefits of dynamic selection over static approaches, it also highlighted the need for more sophisticated selection mechanisms (Quélennec et al. [2024]).
Our development of the **Tra**ining **Dy**namics (TraDy) for a memory-constrained subnetwork update framework in Chapter 4 represents a significant advancement in understanding network fine-tuning dynamics. We leverage insights from heavy-tailed theory and gradient behavior to construct a theoretically grounded framework for efficient subnetwork selection under memory constraints. We establish that fine-tuning exhibits predictable patterns that can be exploited for efficient parameter selection, demonstrating that strategic dynamic random selection of channels within carefully chosen layers can outperform existing state-of-the-art approaches (Quélennec et al. [2025a]).
Chapter 5 represents the most advanced development of this research direction. We extend the insights developed in the previous chapter to improve our layer selection methodology, exploit temporal channel properties to advance from TraDy's random selection to a task-adaptive approach, and demonstrate that stochastic selection outperforms deterministic alternatives. These improvements are unified under our **Me**mory-constrained **Dy**namic subnetwork upd**ate** (MeDyate) framework (Quélennec et al. [2025b]).
Activation Map Compression. The second research direction addresses the memory footprint of activation maps during backpropagation by developing novel compression techniques that drastically reduce memory requirements while providing bounds on performance loss. This approach presented in Part II has the potential of complementing subnetwork selection by further reducing the memory usage of the retained network components.
While tensor decomposition is a well-established strategy for weight compression, we observed that activation tensor compression represented an underexplored area in the research field. In Chapter 6, we apply Higher-Order Singular Value Decomposition (HOSVD) to compress activation tensors while preserving essential information for backpropagation. Unlike traditional SVD, which operates on matrices, HOSVD performs decomposition across all tensor modes simultaneously, thus offering enhanced compression for multi-dimensional activation data. Our approach includes theoretical analysis of compression error bounds and convergence guarantees. Experimental results demonstrate the significant potential of tensor decomposition for activation compression, reducing memory footprint by up to 95% for similar accuracy levels. However, this approach remains limited in practicality as HOSVD computation introduces substantial computational overhead, significantly increasing overall latency (Nguyen et al. [2024]).
To address the computational limitations of direct HOSVD application, Chapter 7 develops our Activation Subspace Iteration (ASI) strategy, a computationally efficient approach that maintains the benefits of tensor decomposition while reducing

associated computational costs. ASI leverages the stability of activation maps during fine-tuning to pre-compute projection tensors and then applies Subspace Iteration at each epoch, effectively approximating HOSVD decomposition with reduced computational cost. Experimental results demonstrate that ASI maintains similar compression and accuracy levels as HOSVD while significantly reducing latency compared to traditional backpropagation (Nguyen et al. [2025]).

Chapter 2

General Background

"*If I have seen further it is by standing on the shoulders of Giants*"

Isaac Newton

Contents

Chapter 2 Abstract

The proliferation of deep learning across diverse application domains has accelerated the demand for deployment on resource-constrained environments, particularly edge devices with stringent memory and computational limitations. In this chapter, we provide background information related to the literature revolving around the field of efficient AI and that forms the basis for this thesis.

2.1 Overview of Compression Literature in Deep Learning

The field of neural network compression has experienced remarkable growth, driven by the imperative to deploy increasingly complex models in resource-limited scenarios. Contemporary compression strategies can be categorized into five fundamental approaches: efficient architectural design, quantization, network sparsification (pruning), knowledge distillation, and tensor decomposition (Cheng et al. [2018]; Deng et al. [2020]).

Efficient Architecture Design. Here, research focuses on developing inherently compact network structures that minimize computational requirements without sacrificing representational capacity. Pioneering works such as Sandler et al. [2018]'s MobileNets introduced depthwise separable convolutions to reduce parameter count and computational complexity. Tan and Le [2019]'s EfficientNets advanced this direction through compound scaling strategies that balance network width, depth, and resolution. For extremely constrained environments, MCUNet (Lin et al. [2020]) demonstrated the feasibility of deploying neural networks on microcontroller units with memory capacities in the hundreds of kilobytes.

Quantization Techniques. They reduce the precision of network parameters and activations, typically from 32-bit floating-point representations to 8-bit or even binary formats. Post-training quantization methods (Krishnamoorthi [2018]) enable straightforward compression of pre-trained models, while quantization-aware training (Jacob et al. [2018]) incorporates precision constraints during the learning process to minimize accuracy degradation.

Network Sparsification. It has been observed that many neural network parameters contribute minimally to model performance. Magnitude-based pruning (Han et al. [2015]) removes parameters with smallest absolute values, while more sophisticated approaches consider gradient information (Lee et al. [2018]) or Hessian-based sensitivity measures (LeCun et al. [1989]). Structured pruning methods (Wen et al. [2016]) demonstrate that sparsity can be effectively induced at multiple granularities

within deep neural networks (including layers, filters, and channels) while achieving substantial weight reduction while often maintaining or even improving model accuracy.

Knowledge Distillation. In this framework, knowledge is transfered from large teacher networks to compact student networks (Hinton et al. [2015]). This paradigm enables the creation of lightweight models that retain much of the performance of their larger counterparts while requiring significantly fewer computational resources.

Tensor Decomposition. Yielding traditional algebraic strategies, the factorization of weight tensors into products of lower-rank matrices reduces both parameter count and computational requirements. Low-rank approximations using Singular Value Decomposition (Denton et al. [2014]) and Tucker decomposition (Kim et al. [2015]) have demonstrated substantial compression ratios with controlled performance degradation. We provide a more detailed review of tensor decomposition techniques in Sec. 2.4, as these methods form a fundamental component of our activation compression approach presented in Part II.

However, a critical limitation of existing compression literature is its primary focus on inference optimization. While these techniques successfully reduce deployment costs, they do not address the memory bottlenecks encountered during training. This gap motivates the exploration of training-specific compression strategies that enable on-device learning capabilities.

2.2 Transfer Learning and Fine-Tuning Techniques

Transfer learning and fine-tuning represent closely related paradigms for adapting pre-trained neural networks to new tasks or domains. Transfer learning broadly encompasses any approach that leverages knowledge acquired from a source task to improve performance on a related target task, while fine-tuning specifically refers to the process of continuing training on a pre-trained model with task-specific data. In practice, the distinction between these terms is often minimal, as fine-tuning serves as the primary mechanism for implementing transfer learning in deep neural networks. The fundamental premise underlying both approaches is that features learned on large-scale datasets contain generalizable representations that can be effectively adapted to downstream applications, particularly when target data is scarce or computational resources are limited.

The success of transfer learning stems from the hierarchical feature representations learned by deep networks, where lower layers capture general-purpose features while higher layers encode task-specific patterns (Yosinski et al. [2014]).

Selective layer updates represent a natural approach to reducing training computational requirements. The fundamental insight is that fine-tuning all network parameters is often unnecessary and potentially detrimental due to catastrophic for-

getting (French [1999]). Lee et al. [2022] provided empirical evidence that strategic layer selection based on distribution shift characteristics can improve transfer learning performance while reducing computational overhead. Their analysis revealed that the optimal set of layers to fine-tune depends on the type and magnitude of domain shift between source and target data. They introduce gradient-to-parameter norm ratio as a principled metric for identifying layers that benefit most from fine-tuning. This approach reflects the intuition that layers with large gradients relative to their current parameter values require more significant updates for optimal performance.
Similarly, parameter-efficient fine-tuning methods have gained considerable attention in large language model adaptation. Adapter layers (Houlsby et al. [2019]) insert small bottleneck modules between transformer layers, requiring updates to only a fraction of the original parameters. Hu et al. [2022]'s Low-Rank Adaptation (LoRA) constrains weight updates to lie in low-dimensional subspaces, achieving competitive performance while updating less than 1% of model parameters.

2.3 Existing Approaches to Memory-Efficient Training

The memory and computational requirements of neural network training significantly exceed those of inference due to the inherent complexity of backpropagation, which necessitates: (i) forward pass computation with intermediate storage of activation maps for gradient calculations, (ii) backward gradient propagation through the network via the chain rule, (iii) computation of weight derivatives at each layer, and (iv) parameter updates based on the computed gradients.
In that context, a series of work explore alternative learning paradigms, standing as fundamentally different optimization strategies to avoid the memory overhead of backpropagation. Among these methods, we can mention the Forward-Forward algorithm (Hinton [2022]) which replaces backpropagation with layer-wise contrastive learning, eliminating the need to store activations across the entire network depth. Hyperdimensional computing approaches (Yang et al. [2023b]) utilize high-dimensional vector spaces for learning, offering inherent memory efficiency. However, these methods consistently underperform compared to backpropagation-based techniques, creating an accuracy-efficiency trade-off that remains unresolved.
Alternatively, several strategies have been developed to reduce either the memory or computational costs associated with backpropagation. Gradient checkpointing (Chen et al. [2016]) reduces memory consumption by storing only a subset of activations during the forward pass and recomputing the remainder as needed

during backpropagation. While this approach effectively reduces peak memory usage, it introduces computational overhead through redundant forward passes, trading memory for computation. Dynamic memory management strategies offer alternative approaches to resource optimization during training. Mixed-precision training (Micikevicius et al. [2017]) reduces memory requirements by performing forward and backward passes using lower-precision arithmetic while maintaining full precision for parameter updates, achieving memory savings without significant accuracy degradation. Gradient accumulation enables training with larger effective batch sizes by distributing computations across multiple smaller batches, allowing practitioners to work within strict memory constraints (Andersson et al. [2022]). However, while these methods successfully address memory bottlenecks, they still require full gradient computation across all network parameters. Consequently, they fail to address the energy consumption challenges that are particularly critical for battery-powered edge devices in on-device learning scenarios.

Sparse Training Methods specifically target the on-device learning scenario by selecting strategic subsets of parameters for update during fine-tuning. Lin et al. [2022] introduced the pioneering Sparse Update (SU) framework, which combines selective parameter updating with operator reordering and quantization-aware scaling to enable fine-tuning on extreme edge devices with as little as 256kB of memory. Their approach demonstrated that carefully selected subnetworks can achieve acceptable performance on downstream tasks while dramatically reducing both computational and memory requirements. However, SU suffers from significant practical limitations: determining adequate sparse configurations requires computationally expensive offline analysis through evolutionary search algorithms applied to each network-budget combination, making the approach labor-intensive and costly. Additionally, the resulting static selections are applied uniformly across all downstream tasks under the implicit assumption that optimal layer-channel configurations remain fixed throughout training.

Building upon this foundation, Kwon et al. [2024] addressed some of SU's limitations by enhancing adaptability across different architectures, datasets, and memory budgets. Their approach employs Fisher information computation on downstream task activations to rank layers and channels, followed by reweighting based on parameter count and MAC operations. While this method provides increased flexibility compared to SU's evolutionary approach, it introduces a fundamental contradiction: computing Fisher information for all network channels requires more memory than the gradient computation it seeks to optimize, violating the original memory constraints that motivate the approach. Moreover, like SU, this method still employs static selection that does not adapt during training, potentially missing opportunities for dynamic optimization as the network converges.

2.4 Tensor Decomposition in Deep Learning

Tensor decomposition, alternatively referred to as low-rank compression, has gained prominence as a theoretically principled and practically viable solution for neural network compression (Xinwei et al. [2023]). The mathematical foundations of these techniques trace back to systems theory and signal processing disciplines (Markovsky [2008]), where they have since garnered substantial attention within the deep learning research community.

Initial applications were constrained to relatively simple scenarios, such as employing Singular Value Decomposition (SVD) for compressing fully connected layers (Xue et al. [2013]). The field has since evolved to encompass more sophisticated decomposition strategies, including Generalized Kronecker Product Decomposition (GKPD) applied to both linear and convolutional architectures (Hameed et al. [2022]), and Semi-Tensor Product (STP) methods designed to achieve enhanced compression ratios while minimizing performance degradation (Zhao et al. [2021]). Recent developments have extended these techniques to modern architectures, with applications to vision transformers demonstrating accelerated inference and backpropagation processes (Yang et al. [2023a]).

The successive progresses in tensor decomposition research can be characterized by progressive improvements across multiple dimensions: enhanced compression ratios, reduced computational latency, decreased power consumption, and expanded applicability across diverse network architectures, all while maintaining minimal impact on model performance making it a well-established strategy for weight compression. In contrast, the application of tensor decomposition to activation compression represents a relatively underexplored area compared to weight compression, presenting opportunities for novel contributions to memory-efficient training methodologies as exposed in this thesis.

Part I

Subnetwork Selection for Fine-Tuning

Chapter 3

Exploration of a Dynamic Neuron Selection Strategy

"*So it begins*"

Theoden

Contents

Chapter 3 Abstract

This chapter introduces a novel paradigm for dynamic subnetwork selection in memory-constrained fine-tuning scenarios. Moving beyond static parameter selection approaches, we propose a budget-constrained framework that adaptively selects neurons based on their learning dynamics. Our approach establishes the groundwork for the more sophisticated selection strategies developed in subsequent chapters.

3.1 Introduction and Related Works

This chapter addresses the challenge of subnetwork selection under strict memory budgets. We begin by establishing in Sec. 3.1.1 the theoretical foundations of gradient computation and memory usage in neural network training. We analyze in Sec. 3.1.2 existing static selection strategies and their limitations, and in Sec. 3.1.3 we introduce an energy-efficient learning approach that inspired our work. Sec. 3.2 presents our dynamic velocity-based approach for memory-efficient learning and Sec. 3.3 details the experimental results.
Our primary contribution lies in the development of a neuron importance metric that captures learning dynamics through velocity measurements, enabling adaptive parameter selection throughout the training process. Unlike static approaches that fix subnetwork configurations a priori, our method dynamically adjusts the set of updated parameters based on their current learning status, leading to improved performance under equivalent memory constraints.

3.1.1 Notations and Details of Gradient Computation

Let us consider a convolutional neural network (CNN) represented as a sequence of N convolutional layers (excluding the bias for simplicity). The forward pass of input $\mathcal{X}$ writes:

$$\mathcal{F}(\mathcal{X}) = (\mathcal{C}_{\mathcal{W}_N} \circ \mathcal{C}_{\mathcal{W}_{N-1}} \circ \cdots \circ \mathcal{C}_{\mathcal{W}_2} \circ \mathcal{C}_{\mathcal{W}_1})(\mathcal{X}), \tag{3.1}$$

where $\mathcal{W}_i \in \mathbb{R}^{C'_i \times C_i \times D_i \times D_i}$ represents the filter parameters of the i-th layer with kernel size $D_i \times D_i$, C_i and C'_i the number of input and output channels respectively. This layer processes an input activation tensor $\mathcal{A}_i \in \mathbb{R}^{B \times C_i \times H_i \times W_i}$ and produces an output tensor $\mathcal{A}_{i+1} \in \mathbb{R}^{B \times C_{i+1} \times H_{i+1} \times W_{i+1}}$, where B denotes the batch size, H_i the height and W_i the width (note that $C'_i = C_{i+1}$).
The loss $\mathcal{L}$ is computed at the output of the network and backpropagated until the

i^{th} layer as $\frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}}$. At this stage, the filter parameters are updated thanks to the computation of $\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i}$ and the loss is propagated to the previous layer as $\frac{\partial \mathcal{L}}{\partial \mathcal{A}_i}$. The computation of these terms follows the chain rule:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i} = \frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}} \cdot \frac{\partial \mathcal{A}_{i+1}}{\partial \mathcal{W}_i} = \text{conv}\left(\mathcal{A}_i, \frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}}\right), \tag{3.2}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{A}_i} = \frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}} \cdot \frac{\partial \mathcal{A}_{i+1}}{\partial \mathcal{A}_i} = \text{conv}_{\text{full}}\left[\frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}}, \text{rot}(\mathcal{W}_i)\right], \tag{3.3}$$

where $\text{conv}(\cdot)$ is the traditional convolution operation, convolving the kernel $\frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}}$ with the input $\mathcal{A}_i$; while $\text{conv}_{\text{full}}(\cdot)$ is the convolutional operation that naturally maps the input $\frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}}$ to an output with the same dimensions as $\mathcal{A}_i$ by using the 180° rotated kernel $\mathcal{W}_i$.
From Eq. (3.2) and Eq. (3.3), it becomes evident that gradient computation requires simultaneous access to the forward activation maps $\mathcal{A}_i$, the backward gradients $\frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}}$ and the network weights $\mathcal{W}_i$. Moreover, due to this dual computation of weight and activation derivatives, it results in the backward pass being around two times more computationally expensive than the forward pass. Similar analysis can be conducted with fully-connected layers (convolution operations are essentially replaced with linear tensor product).
To save memory and reduce computational cost, we propose in this Part to sparsely update the weights, thus naturally reducing the computational cost of weight derivatives computation and the memory overhead of loading weights in RAM. In Sec. 3.1.2 and 3.1.3 we provide details on existing implementations of strategies performing subnetwork selection for memory reduction and computational speedup.

3.1.2 Details of Static Selection Strategies

Static selection strategies address the memory constraint by predetermining which network components to update during training. These approaches leverage the principle that fine-tuning pre-trained models often requires updating only a subset of parameters to achieve satisfactory performance on downstream tasks.
Sparse Update (SU) Framework. Lin et al. [2022] pioneered the Sparse Update (SU) approach as part of their comprehensive framework for on-device training. The SU methodology involves four key components:

1. sparse parameter selection,
2. utilization of memory-efficient architectures,
3. quantization-aware scaling,

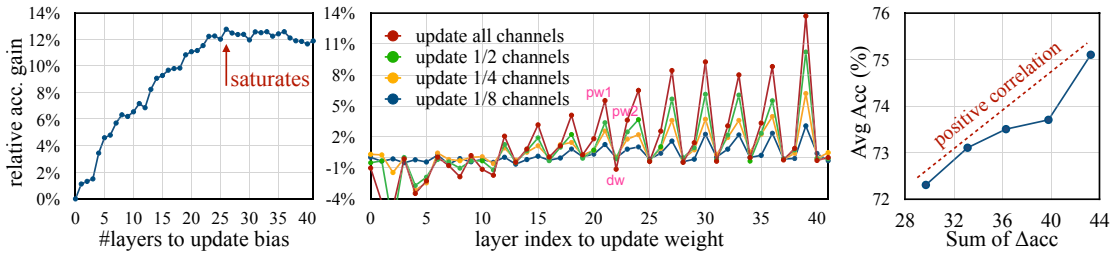
4. and optimized training engines.

The sparse parameter selection component identifies a fixed subset of layers and channels to update throughout training, while freezing the remaining parameters. The determination of optimal SU configurations follows a systematic three-stage methodology designed to quantify the individual contribution of different network components before combining them through evolutionary optimization.

1. *Bias Contribution Analysis.* The first stage evaluates the utility of updating bias parameters at different network depths. Starting from a baseline that updates only the classifier layer, Lin et al. conduct individual training sessions where biases from the i-th layer onward (encompassing all layers from i to the output) are enabled for updates. Each configuration yields a relative accuracy improvement ΔA_i^{bias} compared to the classifier-only baseline. Through comprehensive empirical analysis across multiple depths, the authors identify optimal bias update strategies, typically observing accuracy improvements that increase with depth until reaching saturation at intermediate network layers. This analysis reveals that bias updates in deeper layers provide greater utility while incurring lower memory costs due to reduced activation requirements. Fig. 3.1.(a) represents this accuracy gain with deeper bias training until a saturation point is reached.

2. *Layer-wise Weight Analysis.* The second stage systematically evaluates the impact of partial channel updates within individual layers. For each layer i, multiple training experiments are conducted using different channel update ratios $\zeta \in \{0, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1\}$, measuring the resulting accuracy gains $\Delta\mathcal{A}_\zeta^{\mathcal{W}_i}$. This comprehensive evaluation requires $|\text{layers}| \times |\text{ratios}|$ individual training runs, providing detailed insights into layer-specific sensitivity and optimal sparsity levels. The analysis reveals heterogeneous importance patterns across layers, with certain layers demonstrating high sensitivity to partial updates while others show diminishing returns beyond minimal channel ratios. Fig. 3.1.(b) represents the results of such extensive evaluation.

3. *Memory Cost Characterization.* Parallel to accuracy analysis, Lin et al. [2022] conduct detailed memory profiling to understand the computational costs associated with each layer configuration. The analysis distinguishes between activation memory costs $C_{\mathcal{A}_i}$ (memory required to store input tensors for gradient computation) and weight memory costs $C_{\mathcal{W}_i}$ (memory needed to load layer parameters). This profiling exposed in Fig. 3.2.(a) reveals distinct memory patterns: early layers exhibit high activation costs but low weight costs due to large spatial dimensions and fewer parameters, late layers show



(a) Contribution of last k biases $\Delta\text{acc}_{\mathbf{b}_{[:k]}}$ (b) Contribution of a certain weight $\Delta\text{acc}_{\mathbf{W}_{i,r}}$ (c) Effectiveness of search

Figure 3.1: Contribution analysis of updating biases (a) and weights (b). (c) represents the correlation between downstream task accuracy and $\sum \Delta\text{acc}$. Credits to Lin et al. [2022].

the inverse relationship with compact activations but substantial parameter counts, while intermediate layers demonstrate both low activation and weight costs, making them particularly attractive for memory-constrained updFates.

4. *Evolutionary Configuration Optimization.* The final stage combines individual contribution measurements through evolutionary search to identify optimal sparse configurations. Lin et al. [2022] leverage the empirical observation that combined training accuracy correlates positively with the sum of individual component contributions: $\sum_{i,\zeta} \Delta\mathcal{A}_{\zeta}^{\mathcal{W}_i} + \sum_i \Delta A_i^{\text{bias}}$ (visualized in Fig. 3.1.(c)). This relationship enables the formulation of an optimization problem that maximizes the aggregate contribution score subject to memory budget constraints. The evolutionary algorithm explores the combinatorial space of layer-channel selections and bias depths, iteratively refining configurations to identify pareto-optimal solutions that balance accuracy gains with memory efficiency. An example of such SU scheme is presented in Fig. 3.2.(b), where the bias is updated until layer 22, four layers in the middle of the network are fully updated due to their low memory cost and positive contribution to accuracy and two layers are sparsely updated near the network output to trade off accuracy gain and higher memory footprint.

The resulting SU scheme specifies precisely which layers should be updated, at what channel ratios, and which bias parameters should be enabled. While this systematic approach produces effective configurations for the analyzed network-budget combinations, it suffers from significant practical limitations.
First, the methodology requires extensive contribution analysis, with the number of exploratory training runs scaling multiplicatively with network complexity. Second, as demonstrated by Lee et al. [2022], different downstream tasks require distinct update schemes tailored to their specific characteristics. However, Lin et al. [2022]'s approach performs contribution analysis on a single downstream task under the

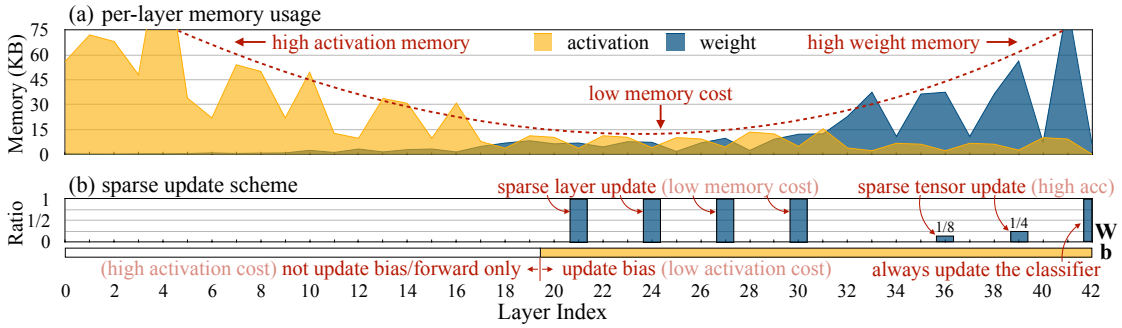


Figure 3.2: (a) represents the weight and activation memory cost of updating *each* layer of MCUNet. (b) corresponds to an example of SU scheme. Credits to Lin et al. [2022].

implicit assumption that findings will generalize to other tasks - an assumption that contradicts empirical evidence of task-dependent layer importance. Third, the framework necessitates computationally expensive evolutionary searches for each memory budget under consideration, making it impractical to adapt to varying resource constraints. Finally, the approach assumes that fine-tuning benefits from static channel updates throughout training, ignoring the potential advantages of adaptive parameter selection based on evolving learning dynamics.

TinyTrain: Task-Adaptive Selection. Kwon et al. [2024]'s TinyTrain addresses some limitations of SU by introducing task-adaptive sparse update policies. Rather than employing fixed configurations, TinyTrain computes Fisher information on activation maps to rank layers and channels and adequately select SU schemes for each target task.

The method introduces a multi-objective criterion that balances accuracy contributions with computational and memory costs. For the i-th layer, this criterion is defined as:

$$s_i = \frac{P_i}{\frac{|\mathcal{W}_i|}{\max\limits_{l \in L}(|\mathcal{W}_l|)} \times \frac{M_i}{\max\limits_{l \in L}(M_l)}}, \tag{3.4}$$

where P_i represents the Fisher potential, $\|\mathcal{W}_i\|$ and M_i represent the number of parameters and multiply-accumulate (MAC) operations of the i-th layer and are normalised by the respective maximum values $\max\limits_{l \in L}(|\mathcal{W}_l|)$ and $\max\limits_{l \in L}(M_l)$ across all layers L of the model.

Another innovation introduced by Kwon et al. [2024] to address the data scarcity challenge inherent to edge deployment is to incorporate meta-learning principles into their framework. The method employs offline meta-training on a source dataset (typically Vinyals et al. [2016]'s Mini-ImageNet) to learn generalizable feature representations that can rapidly adapt to new downstream tasks with minimal data requirements (Snell et al. [2017]). This meta-learning foundation enables effective

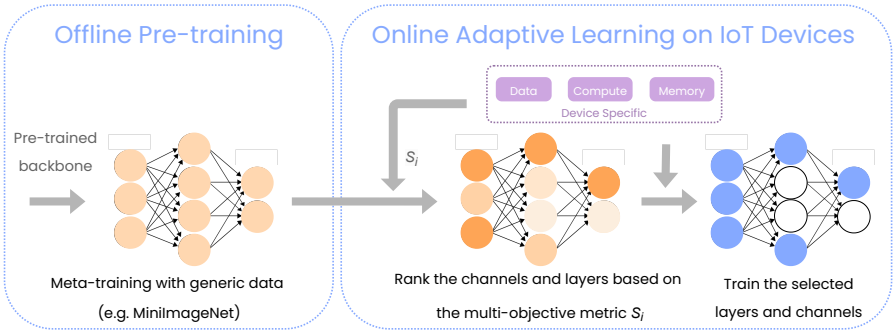


Figure 3.3: Overview of TinyTrain. It consists of (1) the offline pre-training and (2) the online adaptive learning stages. In (1), TinyTrain pre-trains and meta-trains DNNs to improve the attainable accuracy when only a few data are available for adaptation. Then, in (2), TinyTrain performs task-adaptive sparse update based on the multi-objective criterion and dynamic layer/channel selection that co-optimises both memory and computations. Credits to Kwon et al. [2024].

few-shot adaptation while maintaining the computational efficiency required for on-device deployment.
While TinyTrain demonstrates improved adaptability compared to SU, it inherits several limitations from static selection paradigms. First, as second-order derivatives, computing Fisher information for all network channels requires more memory than the gradient computation itself, creating a contradiction with memory-constrained objectives. Second, despite initial task adaptation, the selected sub-network remains static throughout the training process, preventing adaptation to eventual changing parameter importance during fine-tuning.

3.1.3 Details of a Dynamic Neuron Selection Strategy

The Neurons at Equilibrium (NEq) (Bragagnolo et al. [2022]) framework introduces a fundamentally different perspective on parameter selection by focusing on the learning dynamics of individual neurons throughout training. Originally designed to accelerate full network training from scratch rather than enable on-device learning, NEq targets energy consumption reduction and training speedup without performance degradation. Unlike the sparse update strategies discussed in the previous section that focus on fine-tuning pre-trained models, NEq operates in the context of complete training procedures where networks learn task representations from random initialization. The method identifies neurons that have stabilized their input-output relationships and can be safely frozen without affecting final performance, progressively reducing computational requirements as training advances.

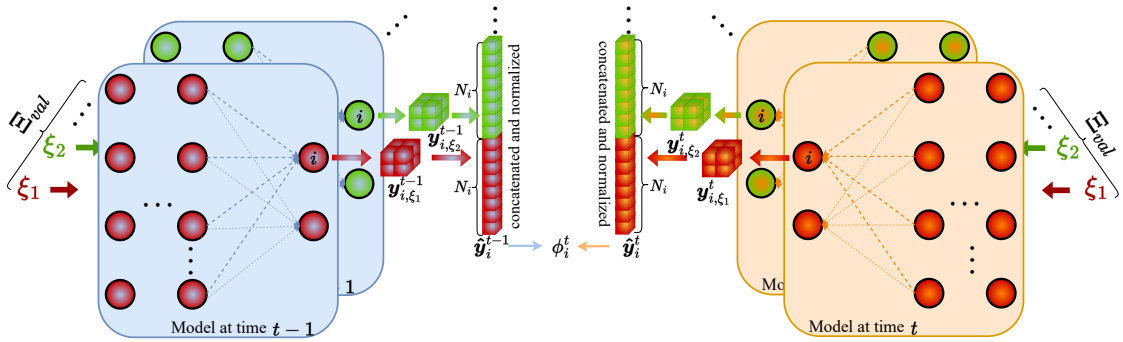


Figure 3.4: For a given epoch t the model (either in blue or orange) receives samples from the validation set (in red or green). The output of the i-th neuron (whose cardinality is N_i) depends on both the model's parameters and the specific sample on the validation set. These outputs are squeezed, concatenated and the obtained vector (of size $N_i \cdot \|\Xi_{val}\|_0$, being $\|\Xi_{val}\|_0$ the cardinality of the validation set) is then normalized, obtaining $\hat{\boldsymbol{y}}^t_i$. Credits to Bragagnolo et al. [2022].

The NEq framework quantifies neuron learning dynamics through systematic comparison of neuron outputs across consecutive training epochs. Let $\boldsymbol{y}^t_{i,\boldsymbol{x}}$ denote the output vector of the i-th neuron when input $\boldsymbol{x}$ is processed by the network after t training epochs. The cosine similarity ϕ^t_i between all outputs of the i-th neuron at epochs t and $t-1$ across the entire validation set Ξ_{val} is computed as:

$$\phi^t_i = \sum_{\boldsymbol{x}\in\Xi_{val}} \sum_{n=1}^{N_{i,\boldsymbol{x}}} \hat{y}^t_{i,\boldsymbol{x},n} \cdot \hat{y}^{t-1}_{i,\boldsymbol{x},n}, \tag{3.5}$$

where $N_{i,\boldsymbol{x}}$ represents the dimensionality of the i-th neuron's output when processing input $\boldsymbol{x}$, and $\hat{y}$ denotes normalized output values. The temporal variations in ϕ_i provide insights into the stability of the input-output mapping learned by each neuron.
To quantify inter-epoch variations, NEq computes the relative change in similarity:

$$\Delta\phi^t_i = \phi^t_i - \phi^{t-1}_i. \tag{3.6}$$

Fig. 3.4 represents the process that allows to compute similarity between neurons outputs. The concept of neuronal equilibrium corresponds to consistent similarity values across epochs, indicating that the considered neuron has converged to a stable input-output relationship. To capture temporal trends and provide momentum-based smoothing, the framework introduces a velocity metric:

$$v^t_i = \Delta\phi^t_i - \mu_{\text{eq}} v^{t-1}_i, \tag{3.7}$$

where μ_{eq} serves as a momentum coefficient that incorporates historical velocity information while maintaining sensitivity to recent changes (in practice $\mu_{\text{eq}} = 0.5$).

A neuron achieves equilibrium status when its velocity magnitude falls below a predefined threshold ε:

$$\left|v_i^t\right| < \varepsilon, \quad \varepsilon \geq 0. \tag{3.8}$$

The equilibrium detection mechanism operates on the principle that decreasing learning rates and converging loss landscapes naturally lead to reduced neuron velocities. During early training phases, most neurons exhibit high velocities as the network explores different regions of the loss surface. As optimization progresses and the network stabilizes, increasing numbers of neurons satisfy the equilibrium criterion and can be frozen without compromising final performance. It is worth noting that due to changes in previous layers, the velocity of a frozen neuron can increase resulting in its unfreezing. This progressive and reversible freezing enables computational savings through reduced gradient computations while preserving the learned representations that contribute to task performance.

NEq offers several advantages over static selection methods. It operates dynamically, adapting the set of trainable parameters based on current learning status rather than predetermined configurations. The method requires no prior knowledge of target datasets, allowing for high flexibility. Additionally, NEq can theoretically adapt to different architectures without architecture-specific tuning.

However, NEq suffers from a critical limitation that prevents direct application to memory-constrained scenarios. In early training epochs, most neurons exhibit high velocities, leading to memory requirements that exceed device constraints. The threshold-based selection mechanism provides no guarantees regarding memory usage, as the number of selected neurons varies unpredictably throughout training. This limitation motivated our development of a budget-constrained framework that combines the dynamic adaptation principles of NEq with explicit memory management mechanisms suitable for on-device deployment.

3.2 Adaptation of Velocity to Memory Constraints

Building upon the theoretical foundations established in the previous section, we develop a practical framework for dynamic neuron selection under strict memory constraints. Our approach extends on NEq's framework by introducing a budget-constrained selection mechanism coupled with a velocity-based refined importance metric that accounts for parameter efficiency. Unlike NEq's threshold-based approach, which provides no memory guarantees, we propose a budget-constrained framework that explicitly limits the number of trainable parameters. Contrary to NEq that addresses energy reduction in training from scratch, typical memory-constrained setups leverage pre-trained architectures. The premise underlying our approach stems from the observation that fine-tuning pre-trained networks requires substantially fewer parameter updates compared to training from scratch.

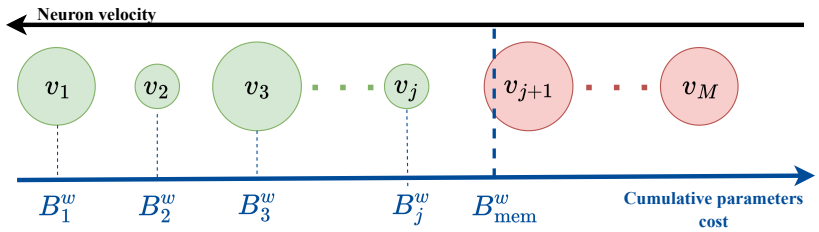


Figure 3.5: Selection of neurons to update given a network of M neurons and a budget B_{mem}^w. The cost C_i^w is proportional to the size of the circle which represents the i-th neuron. The neurons selected for the update are in green, while those frozen in red. Credits to Quélennec et al. [2024].

While Eq. (3.7) effectively identifies neurons undergoing rapid functional changes, it exhibits bias toward neurons with larger parameter counts, as these naturally produce higher absolute velocity values. To address this parameter-count bias and ensure equitable selection across neurons of varying sizes, we introduce a normalized importance metric that quantifies velocity per parameter:

$$\tilde{v}_i = \frac{v_i}{C_i^{\text{w}}}, \tag{3.9}$$

with C_i^w the parameter count of neuron i. This reweighted velocity metric prioritizes neurons demonstrating high learning activity relative to their parameter cost, effectively integrating memory efficiency considerations into the selection process while ensuring that smaller neurons with significant learning dynamics receive appropriate priority alongside their larger counterparts.
Let B_{max}^w denote the maximum number of parameters that can be updated within the available memory budget. Our method consists in ranking all neurons in the network according to our velocity-based metric $\tilde{v}$, and selecting the k fastest neurons such that including the $(k+1)$-th neuron would exceed the budget constraint. The k selected neurons are updated during the subsequent epoch while all other neurons are frozen thus creating a sparse update graph that evolves between epochs. This greedy selection strategy ensures strict adherence to memory limits while maximizing the utilization of available resources.
Formally we write our problem as:

$$\max_j \{B_j^w\} \quad \text{subject to} \quad B_j^w \leq B_{max}^w, \tag{3.10}$$

where $B_j^{\text{w}} = \sum_{i=1}^{j} C_i^w$ represents the cumulative parameter count for the j fastest neurons according to our revised velocity metric. Fig. 3.5 provides a representation of our approach.

Our current methodology encounters a fundamental bootstrapping challenge during the initial training epoch. Since velocity computation requires comparing neuron outputs before and after training iterations, the first epoch lacks historical information necessary for informed selection decisions. While the original NEq framework addresses this by initially training the complete network, our memory-constrained scenario prohibits such an approach. We resolve this initialization problem through random neuron selection within the budget constraint B^w_{max}, providing an unbiased starting point that respects memory limitations while enabling subsequent velocity-based decisions.
This random selection strategy also serves as a control baseline throughout our experimental evaluation, enabling direct assessment of the benefits provided by velocity-based importance estimation compared to purely stochastic parameter selection under equivalent memory constraints.
The complete on-device learning pipeline encompasses model pre-training on upstream tasks followed by memory-constrained adaptation on target devices. This process follows a systematic three-phase cycle that accommodates both static and dynamic selection strategies:

1. *Initial Subnetwork Selection*: Determine the parameter subset for first-epoch updates using either knowledge-driven approaches (evolutionary search as in SU) or heuristic methods (random selection), ensuring compliance with memory budget B^w_{max}.

2. *Training Iteration*: Execute one epoch of training on the downstream dataset $\mathcal{D}_{\text{train}}$ using the selected parameter subset.

3. *Adaptive Selection*: Evaluate and potentially modify the parameter subset for subsequent epochs based on the employed strategy, either maintaining fixed selection for static approaches (SU) or updating selection based on learned dynamics (velocity-based or random reselection).

This modular framework enables systematic comparison of different selection strategies under identical memory constraints and training conditions. The subsequent experimental section leverages this framework to evaluate static selection (SU), velocity-based selection, and random baseline approaches across diverse benchmarks and architectural configurations.

3.3 Experimental Results

We conduct comprehensive experiments to validate our velocity-based selection approach and establish its effectiveness compared to SU and our random neuron

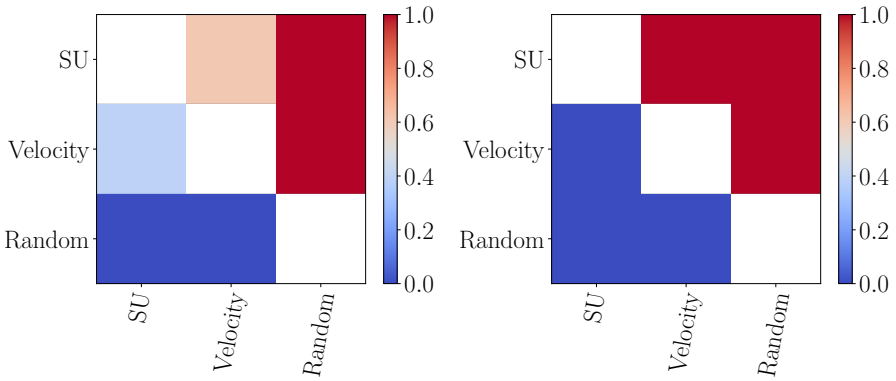
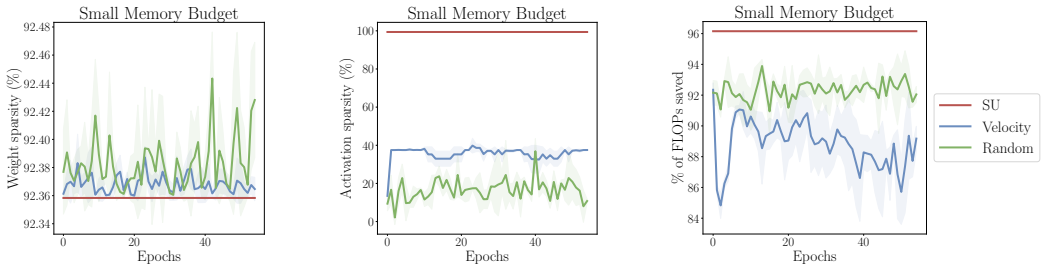


Figure 3.6: T-test comparisons of average final train (3.6a) and test (3.6b) accuracies across multiple experimental dimensions.

selection baseline. Our experimental framework deviates from our initial publication (Quélennec et al. [2024]) to maintain consistency with the methodology established by Lin et al. [2022]. This framework is more consistent with our later publications and will remain consistent throughout this part. We present the detailed experimental configuration in section 3.3.1 and analyze the resulting performance characteristics in Sec. 3.3.2. We do not include comparisons with Kwon et al. [2024]'s TinyTrain framework, as their implementation remains unavailable to the research community and their publication lacks sufficient implementation details regarding both the meta-training procedure and the stopping criteria associated with their proposed importance metric, preventing reliable reproduction of their methodology.

3.3.1 Experimental Configuration and Methodology

Network Architectures and Implementation. Our evaluation employs three state-of-the-art efficient architectures, all initialized with ImageNet (Deng et al. [2009]) pre-trained weights: MobileNetV2 (Sandler et al. [2018]), ProxylessNAS (Cai et al. [2019]), and MCUNet (Lin et al. [2020]). These architectures are selected to maintain direct correspondence with the models utilized in Lin et al. [2022]'s work, ensuring experimental consistency and enabling meaningful performance comparisons. All implementations are executed on Nvidia Tesla V100 SXM2 hardware using PyTorch 2.0.0 with Python backend. Throughout all experimental configurations, we consistently maintain classifier layer training irrespective of the underlying parameter selection strategy, as freezing the final classification layer would compromise adaptation to downstream tasks.



(a) Weight sparsity evolution during training. (b) Activation sparsity evolution during training. (c) Computational savings in weight derivative FLOPs.

Figure 3.7: Efficiency metrics comparison across channel selection strategies during MobileNetV2 fine-tuning on Food dataset under the smallest considered weight memory constraint. Results show evolution of sparsity levels and computational savings throughout training.

Benchmark Datasets and Training Protocol. We evaluate our methodology across seven downstream classification tasks: CIFAR-10 (Krizhevsky et al. [2009]), CIFAR-100 (Krizhevsky et al. [2009]), CUB (Welinder et al. [2010]), Flowers (Nilsback and Zisserman [2008]), Food (Bossard et al. [2014]), Pets (Parkhi et al. [2012]), and VWW (Chowdhery et al. [2019]).[1] We reproduce Lin et al. [2022]'s training policy utilizing cosine annealing learning rate schedules with 5-epoch warm-up periods (Goyal et al. [2017]). Training duration is adapted to dataset complexity: 200 epochs for smaller datasets (CUB, Pets, Flowers), 100 epochs for CIFAR-100, and 50 epochs for larger datasets (CIFAR-10, Food, VWW). The learning rate schedule decays from an initial value of 0.125 down to 0 without applying weight decay or dropout regularization to isolate the effects of parameter selection strategies. Statistical significance is established through triple-seed evaluation protocols, with all reported performance metrics representing means and standard deviations computed across independent runs.

Memory Constraint Framework. To ensure rigorous comparative evaluation, we implement identical memory limitation protocols to those established in Lin et al. [2022]'s SU framework. Our experimental design incorporates three progressively restrictive memory budget tiers for each evaluated architecture, where each budget constraint B^w_{mem} specifies the maximum number of updatable parameters.

3.3.2 Performance Analysis and Comparative Results

Comparative Performance Analysis. Our experimental evaluation systematically compares the three memory-constrained subnetwork selection strategies

[1] Pets: https://creativecommons.org/licenses/by-sa/4.0/, CC BY-SA 4.0 license; ImageNet: https://image-net.org/download.php the ImageNet license; others are not listed.

introduced in Sec. 3.2: SU, our velocity-based selection and our random baseline selection. We conduct comprehensive experiments across the cross-product of three network architectures, seven datasets, three memory budget levels, and three random seeds, yielding 189 individual training runs per selection strategy to ensure statistical robustness.

Statistical Analysis and Results. Fig. 3.6 presents paired t-test results comparing average final training and test top-1 accuracies across all experimental conditions. Each matrix cell represents a statistical hypothesis test evaluating whether the row strategy achieves superior mean test accuracy compared to the column strategy. Comprehensive numerical results are provided in the appendix (Sec. A.3.1).

The results reveal several unexpected patterns in selection strategy performance. While velocity-based selection achieves training accuracy comparable to SU, it demonstrates consistent superiority in test accuracy, suggesting enhanced generalization capabilities. Most surprisingly, our naive random neuron selection baseline surpasses both sophisticated strategies in terms of both training and test performance metrics, challenging conventional assumptions about the necessity of informed parameter selection.

Efficiency Metrics Analysis. Fig. 3.7 illustrates the temporal evolution of key efficiency metrics during MobileNetV2 fine-tuning on the Food dataset under the most restrictive of considered weight memory constraint. These results showcase patterns that remain consistent across different network-dataset-budget combinations, with complete training metrics available in supplementary materials.

We define weight sparsity as the fraction of weight parameters frozen at each epoch due to subnetwork selection decisions. Activation sparsity corresponds to the proportion of forward-pass activations that can be excluded from memory storage, as they are unnecessary for computing gradients of corresponding frozen parameters.

We observe that both velocity-based and random selection achieve weight memory efficiency comparable to SU, maintaining weight sparsity levels exceeding 97%. However, these approaches demonstrate significant shortcomings in activation memory management, the more critical bottleneck in memory-constrained scenarios. Both strategies inherit NEq's neuron-centric selection approach, focusing on individual output channels rather than structured layer components.

This fine-grained selection creates a fundamental architectural mismatch with convolutional memory requirements. Updating weights for any output channel necessitates preserving all activations from the preceding layer throughout the forward pass for subsequent gradient computation. Consequently, selecting even a single neuron within a layer requires storing the complete activation memory footprint of the previous layer, negating potential memory savings. This limitation

is particularly pronounced in early network layers where spatial dimensions are large and activation memory dominates overall consumption.
In contrast, SU's focus on input channel selection provides superior activation memory management, as will be detailed in Sec. 4.2.2, enabling more effective memory utilization under equivalent constraint budgets.
Regarding computational overhead, Fig. 3.7c quantifies the percentage reduction in weight derivative FLOPs achieved through parameter freezing during gradient computation (Eq. (3.2)). While velocity-based and random selection both achieve substantial computational reductions (approximately 88% and 92% respectively), they are exceeded by SU's 96% reduction in weight derivative computation requirements.
These efficiency comparisons highlight the trade-offs inherent in different selection paradigms and motivate the development of more sophisticated approaches that can achieve the generalization benefits observed with dynamic selection while addressing the activation memory limitations that constrain practical deployment.

3.4 Limitations and Conclusions

While our velocity-based approach demonstrates promising performance as a dynamic subnetwork selection strategy, several fundamental limitations constrain its effectiveness and practical applicability, necessitating further research developments.

1. **Activation Memory Limitations.** Our experimental analysis reveals that neuron-level selection, while achieving effective weight sparsity, fails to address the critical bottleneck of activation memory consumption. By selecting individual output channels, our approach requires storing complete activation maps from preceding layers even when updating minimal parameters within each layer. This granular selection paradigm creates a fundamental mismatch with convolutional memory requirements, where activation costs often dominate total memory consumption, particularly in early network layers with large spatial dimensions. Future solutions should target structured selection approaches that enable simultaneous reduction of both weight and activation memory.

2. **Computational Overhead.** The dynamic selection mechanism introduces computational overhead through velocity computation and ranking for all network neurons between each training epoch. While this overhead typically remains modest relative to gradient computation costs, it may become prohibitive in extremely resource-constrained scenarios where even marginal

additional computation compromises deployment feasibility.

3. **Validation Set Dependency.** Our reweighted velocity metric requires access to validation data for similarity computation, creating additional constraints in data-scarce edge environments. Although empirical evidence suggests that minimal validation samples suffice for effective selection (we use 10 inputs in practice), this requirement necessitates reserving downstream data for evaluation rather than training, while also demanding on-device storage that further constrains already limited memory resources. This dependency conflicts with the goal of maximizing training data utilization in resource-constrained scenarios.

4. **Velocity Representativity and Selection Bias.** Unlike the NEq framework where initial full-network training ensures all neurons exhibit meaningful initial velocities, our random initialization strategy creates representativity issues. When no neurons are selected in layers closest to the input during initial epochs, their velocities remain permanently zero due to absence of parameter updates, creating an irreversible selection bias. Even partial selection within early layers leaves neighboring neurons permanently frozen, as zero velocities prevent future selection consideration. This phenomenon produces a cascading effect where velocities in early layers progressively diminish, concentrating updates in later layers. The bias is further amplified by the propagation of early layer changes throughout subsequent network components, causing velocity accumulation in deeper layers while early layers become increasingly underrepresented. This behavior fundamentally contradicts our metric's intended purpose of identifying important neurons throughout the network based on their potential contribution to performance improvements.

5. **Computational Efficiency Considerations.** While our framework prioritizes memory constraints as the primary limiting factor for on-device deployment, our experimental results demonstrate that computational efficiency cannot be neglected. The 96% FLOP reduction achieved by SU compared to our 88% reduction highlights that effective edge deployment requires joint optimization of both memory and computational resources. Future approaches must balance memory efficiency with computational overhead to ensure practical feasibility across diverse edge computing platforms with varying processing capabilities.

Despite these limitations, both our velocity-based and random selection approaches establish important principles for dynamic parameter selection in memory-constrained

environments. The strength of our methods resides in their task-agnostic nature: they can be deployed across any combination of network architecture, dataset, and memory budget without requiring prior computational analysis or domain knowledge. The surprising superior performance of random neuron selection challenges conventional assumptions about parameter importance and provides the foundational insight for developing our TraDy framework. In the following chapters, we will develop activation-aware dynamic subnetwork selection strategies that preserve these desirable properties while incorporating deeper understanding of fine-tuning dynamics and addressing the structural limitations identified in this work.

Chapter 4

Study of Gradient Dynamics during Fine-Tuning

"Oh, we're halfway there"

Bon Jovi

Contents

Chapter 4 Abstract

This chapter presents TraDy (Training Dynamics), a theoretically-grounded framework for dynamic subnetwork selection in memory-constrained transfer learning. Building upon the comprehensive analysis and limitations of existing approaches presented in Chapter 3, we develop a principled methodology that leverages gradient dynamics analysis to inform parameter selection strategies. Our approach addresses the theoretical gaps and practical shortcomings identified in the experimental evaluation of current methods, moving beyond empirical observations toward mathematically-founded selection criteria. Our key contributions include establishing the heavy-tailed nature of stochastic gradients during fine-tuning, demonstrating the architectural consistency of layer importance across downstream tasks, and introducing a dynamic channel sampling approach that maintains strict memory constraints while approximating full gradient information. Through comprehensive empirical validation, we show that TraDy achieves superior performance compared to the static and heuristic approaches analyzed in Chapter 3 across multiple architectures and datasets.

4.1 Foundations and Motivations

The velocity and neuron-based strategies explored in Chapter 3 featured critical limitations with respect to memory-constrained transfer learning scenarios. Rather than continuing to refine these empirically-driven methods, this chapter adopts a fundamentally different paradigm: we seek to understand the underlying gradient dynamics during fine-tuning to develop theoretically-informed selection strategies. The central hypothesis driving this work is that gradient behavior during transfer learning exhibits inherent structure that can be systematically exploited for efficient parameter selection. By analyzing how gradients evolve across network layers and channels during fine-tuning, we can move beyond ad-hoc selection criteria toward principled approaches that respect both the optimization dynamics and memory constraints of edge deployment scenarios.

4.1.1 Limitations of Previous Approaches

The approaches studied in Chapter 3, while pioneering in their demonstration of ultra-low memory fine-tuning, suffer from several critical limitations that motivate the development of more principled alternatives. First, the reliance on Fisher information matrices for parameter importance estimation introduces a fundamental computational contradiction: computing Fisher information for all network

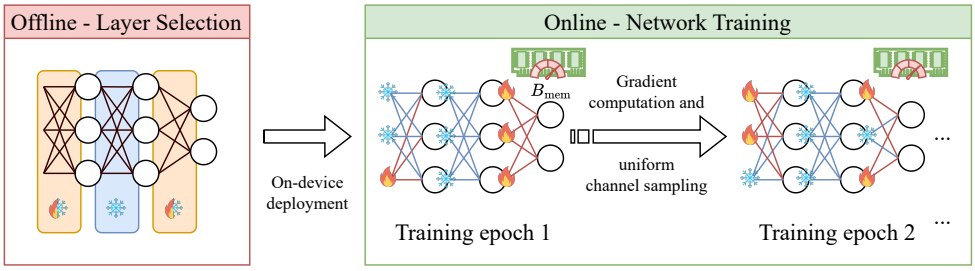


Figure 4.1: TRady dynamically reselects the subgraph to update within the memory budget B_{mem}. Credits to Quélennec et al. [2025a].

parameters requires more memory than the gradient computation these methods seek to optimize. This creates an inherent scalability bottleneck that contradicts the core efficiency objectives.
Second, the static nature of methods such as SU or TinyTrain fails to adapt to the evolving optimization landscape during training. Once parameters are selected for updating, they remain frozen throughout the entire fine-tuning process, potentially missing opportunities to explore more beneficial parameter configurations as the loss landscape evolves. This rigidity becomes particularly problematic when initial importance estimates prove suboptimal or when the relative significance of different network components shifts during adaptation to downstream tasks.
Third, current approaches lack theoretical grounding in optimization theory, relying instead on empirical observations that may not generalize across different architectures, datasets, or training regimes. The absence of principled theoretical frameworks makes it difficult to predict method performance in new scenarios or to understand the principles underlying why certain parameter selection strategies succeed while others fail. This theoretical gap hinders both the development of more effective algorithms and the establishment of rigorous guidelines for practitioners.

4.1.2 Towards Theory-Grounded Subnetwork Selection

This chapter introduces a paradigm shift toward theoretically-informed subnetwork selection by developing the TraDy framework. Our approach is built upon three fundamental insights that emerge from methodical analysis of gradient behavior during transfer learning.
We begin by establishing in Sec. 4.2.1 that stochastic gradients exhibit heavy-tailed behavior during fine-tuning, creating natural sparsity patterns that can be systematically exploited for efficient parameter selection. We follow up by demonstrating in Sec. 4.2.2 how transitioning from neurons to input channels as our finest level of granularity results in both activation and weight sparsity while maintaining

computational coherence. In Sec. 4.2.3 we show that while channel importance exhibits task-dependent variability, layer importance rankings remain remarkably consistent across different downstream tasks and throughout the training process, being primarily determined by network architecture rather than task-specific characteristics. This architectural dependency enables a priori layer selection, reducing the computational overhead associated with dynamic importance estimation while maintaining selection quality.
The synthesis of these insights leads to TraDy, a dynamic sampling framework that pre-selects layers based on architectural analysis and then implements stochastic channel resampling within these layers between training epochs. This approach effectively approximates full gradient information while maintaining computational and memory efficiency, representing a significant advancement over existing empirical approaches as formulated in Sec. 4.2.4.
Sec. 4.3 presents comprehensive experimental validation across multiple architectures and datasets, including novel results on transformer architectures. Finally, we conclude with a discussion of the implications and future research directions.

4.2 Building TraDy Framework

The development of TraDy requires careful consideration of multiple interconnected factors: the theoretical foundations for understanding gradient behavior, the appropriate granularity for parameter selection, and the practical constraints imposed by memory-limited environments. This section systematically addresses each of these aspects, building toward a comprehensive framework that bridges theory and practice.

4.2.1 Heavy-Tailed Theory for Gradient-Based Selection

Recent theoretical advances have revealed that stochastic gradient descent exhibits fundamentally different statistical properties than traditionally assumed, with profound implications for understanding neural network training dynamics and developing principled compression strategies.
Heavy-Tailed Gradient Noise in SGD. Traditional analyses of SGD assume that gradient noise follows Gaussian distributions by invoking the Classical Central Limit Theorem. Consider the standard gradient noise defined as:

$$U_k(\mathcal{W}) = \Delta\tilde{\mathcal{W}}_k - \Delta\mathcal{W}, \tag{4.1}$$

where $\Delta\tilde{\mathcal{W}}_k$ represents the stochastic gradient from a mini-batch and $\Delta\mathcal{W}$ is the full-batch gradient. Classical theory assumes $U_k(\mathcal{W}) \sim \mathcal{N}(0, \sigma^2 I)$. However, Simsekli et al. [2019] demonstrated that gradient noise in deep neural networks actually

exhibits heavy-tailed behavior rather than Gaussian properties, following instead a symmetric α-stable distribution $U_k \sim \mathcal{S}\alpha\mathcal{S}(\sigma)$.
Symmetric α-stable distributions constitute a special class of stable distributions that are symmetric around the origin, characterized by their characteristic function:

$$X \sim \mathcal{S}\alpha\mathcal{S}(\sigma) \iff \mathbb{E}[\exp(i\omega X)] = \exp(-|\sigma\omega|^{\alpha}), \tag{4.2}$$

where $\sigma > 0$ serves as a scale parameter controlling the distribution's spread around zero, and $\alpha \in (0, 2]$ is the stability parameter that determines the tail behavior. These distributions are characterized by probability densities with power-law tail decay proportional to $1/|x|^{\alpha+1}$. When $\alpha = 2$, the distribution reduces to the Gaussian case with finite variance, but for $\alpha < 2$, the distribution exhibits infinite variance and heavy-tailed characteristics. The heavy-tailed nature becomes more pronounced as α decreases, with smaller values indicating thicker tails and more extreme outliers.
From Eq. (4.1), we observe that the full gradient $\Delta\mathcal{W}$ equals the stochastic gradient $\Delta\tilde{\mathcal{W}}_k$ plus noise $U_k(\mathcal{W})$ following a symmetric α-stable distribution. This distributional framework is particularly relevant for understanding gradient dynamics because symmetric α-stable distributions arise naturally as the limit of sums of independent random variables, making them the appropriate mathematical tool for analyzing the aggregate behavior of stochastic gradient estimates across mini-batches.
The heavy-tailed nature of gradient noise has several important implications for neural network optimization. First, SGD under heavy-tailed noise can be analyzed as a discretization of stochastic differential equations driven by Lévy processes rather than Brownian motion, enabling discontinuous jumps between different regions of the loss landscape. This provides theoretical justification for SGD's ability to escape narrow minima and converge to wider basins, as the exit time from local minima depends primarily on basin width rather than depth when $\alpha < 2$.
Implicit Compressibility Through Heavy-Tailed Training. Wan et al. [2023] established a direct connection between heavy-tailed optimization dynamics and neural network compressibility. They demonstrate that injecting additional heavy-tailed noise into SGD iterations causes weight matrix columns to follow multivariate independent and identically distributed heavy-tailed distributions. Consequently, the norm distribution becomes highly skewed: a small subset of columns exhibits disproportionately large norms while the majority remain relatively small. This concentration phenomenon means that the overall weight matrix norm is predominantly determined by a few dominant columns, creating an implicit sparse structure that aligns naturally with selective update requirements.
For overparametrized networks trained with heavy-tailed noise injection, a "propagation of chaos" phenomenon emerges in the mean-field limit: the columns of the weight matrix become independent and identically distributed with a common

heavy-tailed marginal distribution. This independence property is crucial because it enables the application of existing compressibility results for i.i.d. heavy-tailed sequences Gribonval et al. [2012], which establish that such sequences can be compressed with vanishing relative error by retaining only the largest components in magnitude. Specifically, for any desired compression ratio $\kappa \in (0, 1)$, there exists a level of overparametrization such that network weights can be compressed to retain only the top κ fraction of parameters while maintaining performance with high probability.

Application to Transfer Learning. We extend this theoretical framework to gradient distributions during transfer learning. From Eq. (4.1) we thus naturally have that the gradient with respect to the weights $\Delta\mathcal{W}$ is equal to the stochastic gradients $\Delta\tilde{\mathcal{W}}_k$ plus some noise $U_k(\mathcal{W})$ following a symmetric α-stable distribution. Since gradients are naturally composed of stochastic gradient estimates subject to heavy-tailed noise, they inherit the same implicit compressibility properties as the weight matrices analyzed by Wan et al. [2023]. This creates natural sparsity patterns in gradient distributions that can be systematically exploited for parameter selection, providing a principled theoretical basis for understanding why and how gradient-based pruning strategies can be effective in memory-constrained fine-tuning scenarios.

The heavy-tailed nature of gradients suggests that effective approximations of the full gradient can be achieved by focusing computational resources on the high-magnitude components that dominate the gradient norm distribution. This mathematical insight motivates the development of importance metrics based on gradient magnitudes, which we formalize in the following subsection.

4.2.2 From Neurons to Input Channels

Having established the theoretical foundation for gradient-based selection, we now address the critical question of granularity: at what level should we perform parameter selection to efficiently exploit the sparsity patterns revealed by heavy-tailed analysis? In Chapter 3, we followed NEq's neuron-level selection, and while it effectively allowed for significant weight sparsity, it introduced high levels of activation memory that contradict memory efficiency objectives. Inspired by Lin et al. [2022]'s usage of input channels in their SU framework, we propose to transition to input channels as our finest level of granularity for subnetwork selection.

To formalize this analysis, consider the gradient computation for a convolutional layer i with respect to its weight tensor $\mathcal{W}_i$. The partial derivative with respect to a specific weight element can be expressed as:

$$\left[\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i}\right]_{c',c,k,l} = \sum_{b=1}^{B}\sum_{h'=1}^{H'}\sum_{w'=1}^{W'} [\mathcal{A}_i^p]_{b,c,h,w} \left[\frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}}\right]_{b,c',h',w'}, \tag{4.3}$$

where $h = h' \times \text{stride} + k \times \text{dilation}$, $w = w' \times \text{stride} + l \times \text{dilation}$ and $\mathcal{A}_i^p$ is the padded input. As with unstructured pruning, where removing individual parameters does not yield significant computational and memory benefits, freezing individual parameters would create inefficient unstructured sparse tensors. A more effective approach is to freeze along specific weight dimensions, enabling efficient tensor cutting and creating structured gradient sparsity. When considering this approach, we have four potential dimensions for selective freezing: input channels, output channels, and the two kernel dimensions.

From Eq. 4.3, we can observe that updating an input channel c only requires storing in memory the corresponding activation values. In other words, when freezing weight tensors along the input channel dimension, we can generate both weight and activation sparsity. The gradient components corresponding to each input channel form natural groupings that can be treated as independent units for the purpose of exploiting gradient sparsity.

Based on Eq. (4.3), we can derive analytical expressions for both the memory requirements and computational complexity associated with updating a single input channel c within layer i for a single data input. Let $\mathcal{C}_c^{\mathcal{W}_i} = C' \times D \times D$ represent the weight memory cost and $\mathcal{C}_c^{\mathcal{A}_i} = H \times W$ represent the activation memory cost for channel c. The total space complexity $(\Theta_{\text{space}})_c$ and time complexity $(\Theta_{\text{FLOPs}})_c$ are then given by:

$$\left(\Theta_{\text{space}}\right)_c = \mathcal{C}_c^{\mathcal{W}_i} + \mathcal{C}_c^{\mathcal{A}_i}, \tag{4.4}$$

$$\left(\Theta_{\text{FLOPs}}\right)_c = D^2 C' H' W'. \tag{4.5}$$

These expressions demonstrate that input channel-level selection provides fine-grained control over both memory and computational consumption while maintaining the structural coherence necessary for effectively exploiting the heavy-tailed sparsity patterns identified in gradient distributions.

Taking inspiration from Wan et al. [2023]'s use of weight norm magnitude with respect to the column dimension for pruning, we exploit the gradient norm for input channel c in layer i to characterize its importance in our study of channel behavior throughout training:

$$\left\| \left(\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i} \right)_c \right\|_2 = \sqrt{\sum_{c',k,l} \left[\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i} \right]^2_{c',c,k,l}}. \tag{4.6}$$

While the gradient norm directly captures the magnitude information that characterizes the concentration of information in a few channels, it fails to account for the memory constraints that are central to our deployment scenario. Channels with larger memory footprints naturally exhibit higher gradient norms due to their increased dimensionality, creating a bias that can lead to inefficient memory

utilization and suboptimal exploitation of the available sparsity.
To address this limitation and better align with memory-constrained optimization objectives, we introduce the Reweighted Gradient Norm (RGN), which incorporates both gradient magnitude information and memory efficiency considerations:

$$\mathrm{RGN}_c = \frac{\left\| \left(\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i} \right)_c \right\|_2}{\mathcal{C}_c^{\mathcal{W}_i} + \mathcal{C}_c^{\mathcal{A}_i}}. \tag{4.7}$$

This reweighting scheme counteracts the bias toward memory-intensive parameters while preserving the gradient magnitude information essential for exploiting the sparse structure of stochastic gradients. The RGN metric thus provides a principled approach to balancing the insights from heavy-tailed analysis with the practical requirements of memory-constrained deployment, enabling more effective parameter selection under resource constraints.

4.2.3 Channel and Layer Analysis

As gradients are composed of convolutions of both activation maps and activation derivatives (Eq. (4.3)), both of which are heavily dependent on the task data distribution, it becomes evident that the value distributions of weight derivatives are primarily determined by the downstream task. The fundamental purpose of learning is to modify network weights according to input data and the target objective to capture representations and features relevant to the considered dataset. As such, we formalize:

Proposition 4.2.1 *The distribution of channel gradient norms varies between datasets.*

The activation map represents features extracted at each layer of the network and reflects how the network characterizes the input data. The activation derivatives shape the loss landscape and vary according to the specific downstream task. Both components are fundamentally shaped by the task at hand, justifying our statement that channel gradient norms vary between downstream tasks. We provide empirical validation of this phenomenon in Sec. 4.3.1.
We now extend our analysis to understand layer-level behavior during fine-tuning. We define the layer-level reweighted gradient norm as the sum of individual channel RGN values:

$$\left\| \frac{\partial \mathcal{L}}{\partial \mathcal{W}_i} \right\|_{\mathrm{RGN}} = \sum_{c=1}^{C} \mathrm{RGN}_c = \frac{1}{(\Theta_{\mathrm{space}})_i} \sum_{c=1}^{C} \left\| \left(\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i} \right)_c \right\|_2. \tag{4.8}$$

This formulation provides a memory-normalized measure of layer importance that enables meaningful comparisons across layers with different architectural

characteristics.
Network architecture determines which layers consistently exhibit higher gradient norms than others. This architectural dependency becomes particularly evident in networks with residual connections, where skip connections mitigate gradient vanishing by effectively reducing the virtual depth of the network for certain computational paths. As a result, we typically observe a characteristic pattern in gradient norm distribution: the first layer of each residual block generally displays significantly higher gradient norms than subsequent layers within the same block. More specifically, let us consider the case of R convolutional layers having identical dimensions, intercepted by ReLU activations, where a skip connection re-injects the input of the first layer into the final output $\mathcal{Y}$, such that $\mathcal{Y} = \mathcal{A}_{i+R-1} + \mathcal{A}_i = \mathcal{C}_{\mathcal{W}_{i+R-1}} \circ \cdots \circ \mathcal{C}_{\mathcal{W}_i}(\mathcal{A}_i) + \mathcal{A}_i$. We denote $\mathcal{Z}_i$ as the i-th layer pre-activation, $\mathbf{1}$ as the indicator function, and $\odot$ as the Hadamard product operator. Following Eq. (4.3), the weight derivatives can be expressed as:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{W}_{i+R-1}} = \text{conv}\left(\mathcal{A}_{i+R-1}, \left[\frac{\partial \mathcal{L}}{\partial \mathcal{Y}} \odot \mathbf{1}_{\mathcal{Z}_{i+R-1}>0}\right]\right). \tag{4.9}$$

For all $k \in [i, i+R-2]$, we define $\mathcal{J}$ through the following recursive expression:

$$\begin{cases} \mathcal{J}(i+R-1) = \left[\frac{\partial \mathcal{L}}{\partial \mathcal{Y}} \odot \mathbf{1}_{\mathcal{Z}_{i+R-1}>0}\right] \\ \mathcal{J}(k) = \left(\text{conv}\left(\mathcal{J}(k+1), \mathcal{W}_{k+1}^{\top}\right) \odot \mathbf{1}_{\mathcal{Z}_k>0}\right) \end{cases}. \tag{4.10}$$

Subsequently, for all $k \in [i, i+R-2]$:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{W}_k} = \text{conv}\left(\mathcal{A}_k, \mathcal{J}(k)\right). \tag{4.11}$$

Moreover, the input derivative can be written as:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{A}_i} = \frac{\partial \mathcal{L}}{\partial \mathcal{Y}} \cdot \left[I + \frac{\partial(\mathcal{C}_{\mathcal{W}_{i+R-1}} \circ \cdots \circ \mathcal{C}_{\mathcal{W}_{i+1}} \circ \mathcal{C}_{\mathcal{W}_i})}{\partial \mathcal{A}_i}\right], \tag{4.12}$$

where I denotes the identity tensor.
Given that we work with pre-trained networks, we can exploit specific properties established during their initial training phase. Pre-trained deep neural networks typically undergo regularization through weight decay and gradient clipping, which constrains weight norms to generally remain below one. Simultaneously, the inclusion of batch normalization layers during pre-training ensures that activation norms are similarly bounded. When fine-tuning on downstream tasks that share reasonable similarity with the pre-training domain, these weight and activation properties tend to be preserved, since the magnitude of weight adjustments remains

relatively small.
For instance, when $\|\mathcal{W}_{k+1}\|_2 \leq 1$ for all $k \in [i, i+R-2]$ and $\|\mathcal{A}_i\|_2 \leq 1$, we obtain:

$$\left\| \frac{\partial \mathcal{L}}{\partial \mathcal{W}_i} \right\|_2 \leq \left\| \frac{\partial \mathcal{L}}{\partial \mathcal{W}_{i+1}} \right\|_2 \leq \cdots \leq \left\| \frac{\partial \mathcal{L}}{\partial \mathcal{W}_{i+R-1}} \right\|_2 . \tag{4.13}$$

While exceptions may occur with certain layers occasionally exhibiting weight or activation norms exceeding one, these instances reflect inherent properties of the pre-trained network rather than task-specific adaptations. The fundamental insight is that layers consistently maintain their relative gradient norm rankings across diverse downstream tasks. This pattern becomes even more pronounced when using our reweighted gradient norm metric, since both channel weight and activation memory costs are architecture-dependent constants within each layer. We provide empirical validation of this behavior in Sec. 4.3.1. We formalize this observation as:

Proposition 4.2.2 *The relative ranking of layers according to their reweighted gradient norm remains largely invariant over time during training and across different downstream tasks. This ranking is primarily determined by the network architecture rather than dataset-specific characteristics.*

Based on these observations, we can strategically restrict parameter updates to the subset of layers that naturally receive higher gradients. Recent literature supports this approach, with multiple studies demonstrating that selectively updating certain layers provides significant contributions to model optimization on downstream tasks Kaplun et al. [2023]; Lee et al. [2022]; Zhang and Bottou [2024]. The practical implication is substantial: depending on the similarity between pre-training and downstream tasks, updating only a carefully selected subset of layers can maintain performance comparable to full fine-tuning while significantly reducing memory requirements.

4.2.4 TraDy Algorithm Formulation

The synthesis of our theoretical analysis leads to the complete TraDy algorithm, which operates through a two-stage selection process. The first stage involves architecture-based layer pre-selection using the RGN-based importance ranking developed in the previous section. This pre-selection can be performed offline using any relevant downstream task, as the layer rankings exhibit consistency across tasks (Proposition 4.2.2). The second stage implements dynamic channel sampling within the pre-selected layers. Contrary to layers, channel selection cannot rely on predetermined importance rankings due to the task-dependent nature of RGN distributions established in Proposition 4.2.1. This presents a fundamental computational dilemma: edge deployment scenarios typically lack access to complete task

Algorithm 1: TraDy. Credits to Quélennec et al. [2025a].

Require: Pre-trained backbone weights $\mathcal{W}$, number of epochs n, training data D_{train}, test data D_{test}, memory budget B_{mem}, set of relevant layers $\{L_K\}$.

for *epochs* $= 1$ ***to*** n **do**
  Randomly sample channels $\{C^t\}$ within the set of relevant layers $\{L_K\}$ following uniform distribution until memory budget B_{mem} is met.
  Update weights of selected channels using D_{train}.
Evaluate fine-tuned backbone using D_{test}.

datasets for offline analysis, while stringent memory budgets prevent full gradient computation during online operation. Attempting to directly compute channel-wise full RGN distribution would violate the very memory constraints our framework aims to address, creating a contradiction between optimal selection criteria and practical deployment requirements.

To address this dilemma, we introduce TraDy, a stochastic sampling framework designed to operate within stringent memory limitations. Drawing inspiration from the surprisingly effective performance of random neuron selection demonstrated in Chapter 3, we propose a dynamic channel resampling mechanism that selects input channels for parameter updates exclusively within the architecturally-determined layer subset, with channel selection renewed between successive training epochs. This methodology guarantees that combined memory consumption for both weights and activations never exceeds the imposed budget constraints, achieving an optimal balance between parameter space exploration and hardware deployment limitations.

The layer pre-selection methodology established in Sec. 4.2.3 enables us to concentrate computational resources on the subset containing the majority of gradient information, consistent with the theoretical predictions of heavy-tailed distributions outlined in Sec. 4.2.1. Through epoch-wise stochastic channel resampling, the expected value of our selected gradient converges to the non-null gradient components within the restricted layer subset, since layers with gradient of negligible magnitudes have been systematically excluded through our architectural selection process. This probabilistic approximation strategy maintains fidelity to the full optimization objective while respecting hardware constraints.

Let $\Delta\tilde{\mathcal{W}}_t$ denote the non-null gradient at epoch t and $\Delta\mathcal{W}_{\{C^t\}}$ denote the sparse gradient composed of the subset $\{C^t\}$ of randomly selected channels at epoch t within $\{L_K\}$, the subset of K pre-selected layers. Following the principle that stochastic gradient expectation equals full gradient expectation, and given that our layer selection excludes low-magnitude gradients while the stochastic channel

selection follows a uniform distribution, we have:

$$\mathbb{E}_t\left[\Delta\tilde{\mathcal{W}}_t\right] \simeq \mathbb{E}_t\left[\Delta\mathcal{W}_{\{C^t\}}\right]. \tag{4.14}$$

Furthermore, the algorithmic complexity of selecting k channels from n candidates scales as $\mathcal{O}(k\log(n))$, which represents negligible computational overhead relative to gradient evaluation costs in typical network architectures, ensuring our methodology introduces minimal additional computational burden.

The complete algorithm depicted in Fig. 4.1 proceeds as follows. We first perform offline layer ranking using a small subset of task data to establish the top-K layers $\{L_K\}$ according to their RGN values. These layers capture the majority of gradient information (typically over 90% based on our empirical analysis) while comprising a small fraction of the total network parameters.

Alg. 1 provides an overview of the online component. Given a pre-trained network, a downstream task, and a memory budget, during each training epoch, we randomly sample input channels from within the pre-selected layers until the memory budget is exhausted (line 2). The sampling is performed uniformly across available channels, ensuring unbiased exploration of the parameter space over time. After each epoch, the channel selection is resampled, allowing the algorithm to explore different parameter configurations while maintaining focus on architecturally important layers.

This approach effectively addresses the limitations of previous methods by combining theoretical rigor with practical efficiency. The architecture-based layer selection eliminates the computational overhead of online importance estimation, while the dynamic channel resampling ensures adaptation to task-specific requirements without violating memory constraints. At the end of training, we evaluate our model's performance on the test dataset (line 4). In the next section, we present our experimental results.

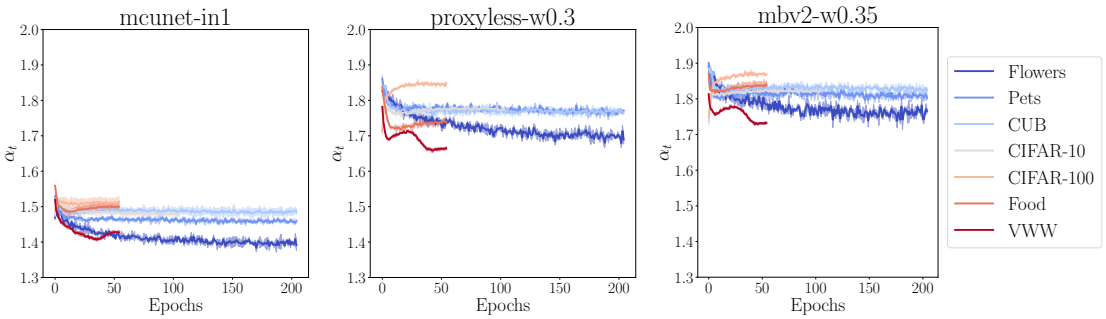


Figure 4.2: Evolution of stochastic gradient heavy-tailed index α. Credits to Quélennec et al. [2025a].

4.3 Experimental Results

The experimental validation of TraDy employs the same comprehensive framework introduced in Sec. 3.3.1, but extends it to account for both weight and activation memory constraints, a critical consideration for realistic deployment scenarios. Additionally, we expand our evaluation to include transformer architectures, demonstrating the generalizability of our approach beyond convolutional networks.

4.3.1 Preliminary Experiments

Heavy-Tailed Validation. We provide empirical confirmation of the heavy-tailed properties of stochastic gradients during fine-tuning, as theoretically established in Sec. 4.2.1. Reproducing the methodology proposed by Simsekli et al. [2019], we employ Mohammadi et al. [2015]'s estimator for α-stable distributions. At each fine-tuning epoch t, we aggregate stochastic gradients from all P trainable parameters over S training steps, forming a $P \times S$ matrix. The estimator processes this matrix to yield α_t, quantifying the heavy-tailed index of the gradient distribution at epoch t. Fig. 4.2 demonstrates the temporal evolution of α_t across our three network architectures when adapted to each downstream tasks. Across all experimental configurations, α_t consistently remains below two, substantiating the heavy-tailed nature of stochastic gradients during transfer learning. Notably, MCUNet displays markedly more pronounced heavy-tailed behavior relative to other architectures. We attribute this phenomenon to its compact architectural design, which achieves greater parameter efficiency compared to alternative networks. This enhanced efficiency likely creates more concentrated gradient information within fewer parameters, amplifying the heavy-tailed characteristics of the resulting gradient distribution.

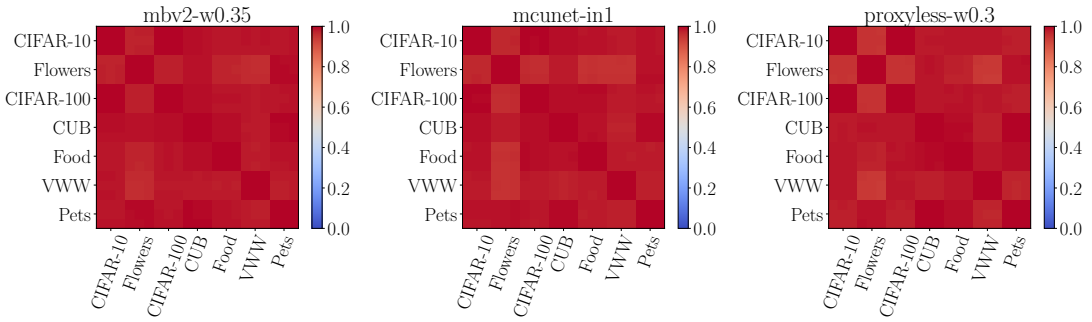


Figure 4.3: Spearman correlation of layer gradient norm across seeds and datasets. Credits to Quélennec et al. [2025a].

Channel and Layer Validation. To validate our theoretical predictions about layer and channel behavior formalized in Propositions 4.2.1 and 4.2.2, we examine the consistency of importance rankings across different experimental conditions through comprehensive statistical analysis.

For layer analysis, we characterize each fine-tuning experiment through its "layer importance profile", a vector containing the aggregate gradient norm of each layer accumulated across all training epochs. To assess the stability of these profiles, we calculate Spearman rank correlation coefficients between all possible pairs of fine-tuning runs across our seven downstream datasets, employing three random seeds per dataset. This approach generates a comprehensive 21×21 correlation matrix for each network architecture, providing a systematic view of ranking consistency as illustrated in Fig. 4.3. The consistently high Spearman correlation coefficients (above 0.8 across all tested conditions) provide strong empirical support for Proposition 4.2.2, confirming that layer importance ranking is primarily determined by architectural characteristics rather than task-specific factors. This consistency validates our approach of performing offline layer ranking using limited task data, as the relative ordering remains stable across diverse experimental scenarios.

For channel analysis, we adopt a parallel methodology by constructing vectors where each element captures the cumulative gradient norm of a specific channel across all training epochs. This representation encodes channel importance patterns for each fine-tuning experiment, enabling systematic comparison across different experimental conditions. To determine whether these channel importance patterns exhibit significant variation between tasks, we apply Student's t-test for all pairwise comparisons, with statistical outcomes presented in Fig. 4.4. The predominantly null in all cross-dataset comparisons indicate statistically significant differences in channel importance distributions across tasks, providing empirical validation of Proposition 4.2.1. This analysis demonstrates that channel-level importance cannot

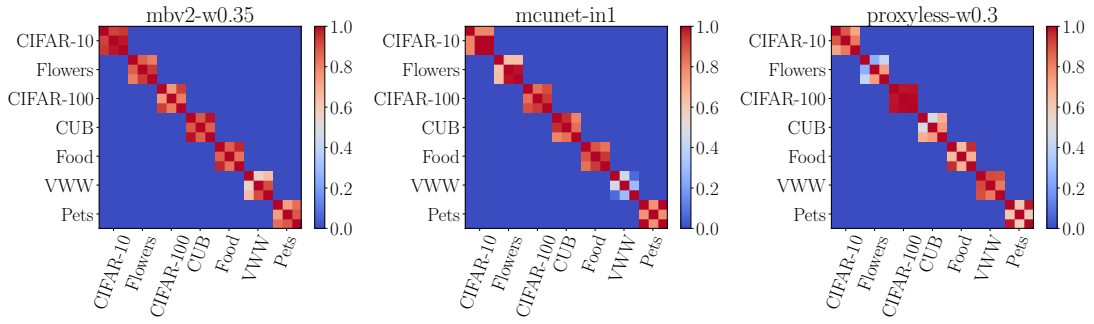


Figure 4.4: T-test of channel gradient norm across seeds and datasets. Credits to Quélennec et al. [2025a].

be predetermined based solely on architectural considerations, unlike layer-level behavior. The statistical evidence supports our theoretical analysis and justifies the necessity for dynamic channel sampling approaches that can adapt to task-specific gradient patterns while maintaining computational efficiency within memory constraints.

Layer Selection Analysis. Having validated the architectural consistency of layer importance rankings, we now investigate the practical question of determining the optimal number K of highest-ranked layers to include in our sampling framework. To examine this parameter's influence on performance, we conduct a systematic study using transfer learning under the most restrictive memory budget B_{mem} on CIFAR-10. We establish layer rankings a priori using RGN values and vary K to evaluate different layer subset sizes. Within each selected layer subset, we compare three distinct dynamic channel selection approaches:

1. Random channel selection until memory budget B_{mem} is satisfied (Random).
2. Oracle-based deterministic selection of channels with maximum RGN values using complete pre-computed gradient information until B_{mem} is satisfied (Det RGN).
3. Oracle-based deterministic selection using raw gradient norm values with complete pre-computed gradient information until B_{mem} is satisfied (Det Raw).

Fig. 4.5 illustrates the outcomes of this investigation. For random selection, removing the least significant layers initially enhances training performance by focusing the sampling process on more informative channels. Performance reaches its optimum in the 35 to 40 layer range before degrading as crucial layers are excluded, demonstrating their essential contribution to convergence. Remarkably, these top

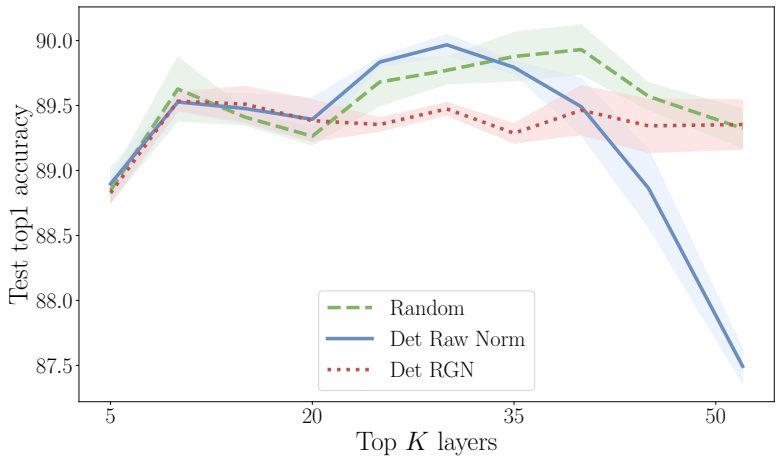


Figure 4.5: Final test top1 accuracies depending on the number of top K layers for different dynamic channel selection strategies. Credits to Quélennec et al. [2025a].

35 layers encompass 97% of the network's total RGN. We adopt this 97% threshold for our remaining architectures, resulting in 27 layers for MCUNet and 43 layers for ProxylessNAS.

The RGN-based deterministic approach features stable performance across varying K values except when decreasing from the top 10 to top 5 layers, causing performance degradation. This pattern supports our theoretical expectation that gradient significance concentrates within a limited layer subset, as established by the heavy-tailed distribution analysis in Sec. 4.2.1.

The Raw Norm strategy exhibits more intricate behavior patterns. When operating across the complete network, it produces moderate performance but demonstrates marked improvement as less significant layers are removed. This trend indicates that lower-ranked layers contain channels with significant absolute gradient magnitudes but suboptimal gradient-to-memory efficiency ratios, which our reweighting methodology successfully identifies and deprioritizes.

Notably, Raw Norm selection attains peak accuracy above RGN deterministic and similar to Random selection at $K = 30$, suggesting potential alternative approaches for balancing gradient magnitude against memory efficiency beyond our current framework. However, as discussed in our method formulation, utilizing gradient norm information for channel selection introduces computational overhead that contradicts the efficiency objectives of random selection. Beyond the 20-layer threshold, all approaches converge to comparable performance levels, likely because channel selection becomes sufficiently dense within the search space, causing all methods to select similar channel configurations.

4.3.2 Main Results

Comparative Performance Analysis. To rigorously validate the claims presented in Sec. 4.2, we conduct a comprehensive empirical evaluation comparing TraDy against established baselines and alternative strategies. Building upon the channel selection approaches introduced in our preliminary analysis and the Random baseline examined in Chapter 3, we systematically evaluate three distinct memory-constrained selection strategies. For each method, we investigate both Static and Dynamic variants: Static approaches perform channel selection once at initialization and maintain the same channels throughout training, while Dynamic approaches reapply the selection criterion after each epoch, enabling different channels to be updated over time.

1. *Full Random.* This baseline strategy, previously evaluated with output channels in Chapter 3, randomly selects channels from the entire network architecture without layer-based prioritization. It serves as a control to assess the impact of our layer selection methodology.

2. *Det RGN.* As presented in our layer selection analysis above, this oracle-based approach computes the complete gradient at each epoch without weight updates, then deterministically selects channels with maximum RGN values within the predetermined subset of top-K layers. While computationally impractical for deployment due to full gradient computation requirements, it provides an upper-bound performance reference.

3. *Top-K Random.* This strategy randomly samples channels exclusively from within the predetermined subset of top-K layers identified through our architectural analysis. In its dynamic variant, this constitutes our proposed TraDy algorithm.

Following the experimental framework established in Chapter 3, we benchmark our approaches against the SU baseline and employ the same memory budgets B_{mem} with three constraint levels per architecture, now accounting for both parameter and activation memory consumption.

Statistical Analysis and Results. Using the same experimental design as Chapter 3 (three architectures, seven datasets, three memory levels, three seeds), we conduct 189 training runs per strategy. Fig. 4.6 presents statistical comparisons of final test accuracies, with complete table of results in the appendix.

The results reveal several key findings regarding our proposed strategies. Each dynamic variant (D-prefix) consistently outperforms its static counterpart (S-prefix), confirming the benefits of adaptive channel resampling. While S-Full Random produces the poorest results, our TraDy algorithm—combining layer pre-selection

with dynamic channel sampling—achieves optimal performance. Notably, D-RGN Deterministic now surpasses SU performance, demonstrating the effectiveness of our RGN importance metric regarding channel selection.
Interestingly, D-Full Random now achieves performance equivalent to SU, contrasting with Chapter 3 results where Full Random with output channels outperformed SU. This performance shift highlights the impact of transitioning from output to input channels on selection strategy effectiveness, suggesting that architectural granularity choices significantly influence comparative algorithm performance.
We attribute TraDy's superior performance to its balanced approach to gradient approximation. Under severely constrained memory budgets, D-RGN Deterministic's strategy of consistently selecting maximum RGN channels effectively freezes numerous channels with moderate but meaningful RGN values. This deterministic approach may drive optimization toward steep gradient directions, potentially converging to suboptimal local minima. Conversely, TraDy maintains the same expected gradient direction as the complete non-null gradient while introducing beneficial stochasticity through dynamic resampling among architecturally significant layers. The poor performance of S-TopK Random (second-worst strategy) further emphasizes that dynamic channel reselection is essential for effective memory-constrained fine-tuning. Full experimental results tables are available in Sec. A.3.2 of the appendix.

Efficiency Metrics Analysis. Fig. 4.7 presents the same efficiency analysis as Fig. 3.7 (MobileNetV2 fine-tuning on the Food dataset under the smallest memory budget) but applied to the selection strategies examined in this chapter. Complete training metric curves are available in supplementary materials.
All our proposed methods achieve comparable levels of weight and activation sparsity, ranging from 93% to 99% and 97.5% to 99.5% respectively. SU achieves extremely low activation memory consumption at the expense of higher weight memory usage, potentially linked to its evolutionary search process that implicitly maximizes the number of updated parameters. In con-

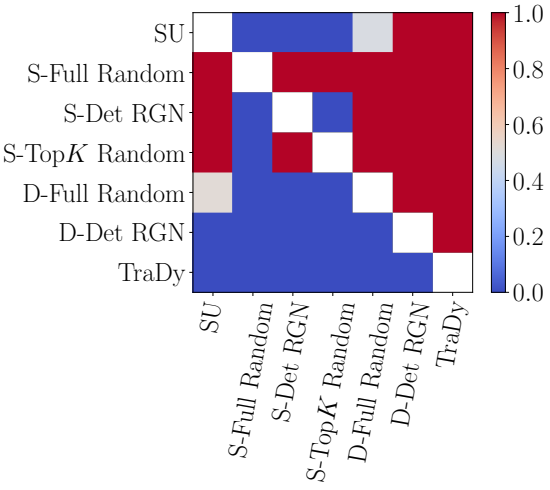
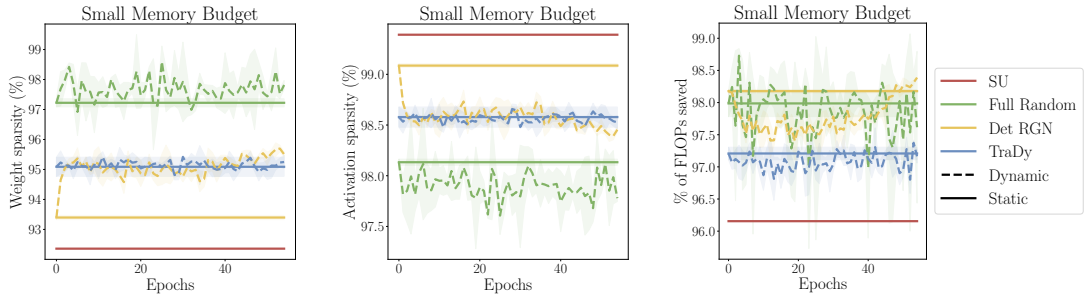


Figure 4.6: T-test comparisons of average final test accuracies across multiple experimental dimensions. Credits to Quélennec et al. [2025a].



(a) Weight sparsity evolution during training.

(b) Activation sparsity evolution during training.

(c) Computational savings in weight derivative FLOPs.

Figure 4.7: Efficiency metrics comparison across channel selection strategies during MobileNetV2 fine-tuning on Food dataset under memory constraint. Results show evolution of sparsity levels and computational savings throughout training. Credits to Quélennec et al. [2025a].

trast, TraDy demonstrates a more balanced memory trade-off, suggesting that maximizing weight updates does not necessarily translate to improved task performance.
Furthermore, TraDy consistently requires fewer FLOPs for weight derivative computation compared to SU. This computational advantage arises from the prominence of depthwise convolution layers among our top-ranked layers, which inherently exhibit low computational costs during both forward and backward passes.
While citelin2022device observed that depthwise layers contribute minimally to accuracy when updated in isolation, Zhang et al. [2019] demonstrated that layer contributions cannot be evaluated independently, as they depend critically on which other layers remain active or frozen. Our findings suggest that coordinated updating of depthwise layers alongside their corresponding pointwise layers within each block creates synergistic effects that promote efficient convergence. This hypothesis is supported by the consistently high RGN values observed for these layer combinations, indicating their collective importance for gradient-based optimization under memory constraints.

4.3.3 Results on Transformer Architectures

While TraDy was originally developed for on-device learning scenarios, its theoretical foundations should in principle extend to larger architectures operating under memory or energy constraints. To confirm scalability, we evaluate our approach on transformer models across both computer vision and natural language processing domains. For vision tasks, we employ a SwinT model Liu et al. [2021] pre-trained

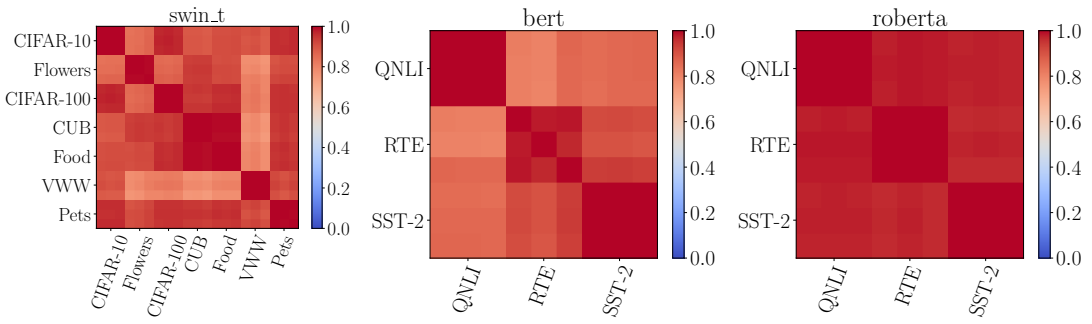


Figure 4.8: Spearman correlation of layer gradient norm across seeds and datasets. Credits to Quélennec et al. [2025a].

on ImageNet and adapted to our seven downstream tasks. For NLP evaluation, we utilize BERT Kenton and Toutanova [2019] and RoBERTa Liu [2019] as representative transformer architectures, fine-tuning them on three established tasks: QNLI Demszky et al. [2018], RTE Poliak [2020], and SST2 Socher et al. [2013].

Proposition Validation on Transformers. We replicate our layer and channel behavior analysis from Sec. 4.3.1 to validate whether Propositions 4.2.2 and 4.2.1 hold for transformer architectures. Figures 4.8 and 4.9 present the layer gradient norm Spearman correlations and channel gradient norm t-tests for these three architectures.

The Spearman correlation analysis in Fig. 4.8 confirms that transformer architectures also exhibit layer importance consistency across downstream tasks, validating Proposition 4.2.2 beyond convolutional networks. Correspondingly, Fig. 4.9 demonstrates significant variation in channel importance distributions between datasets, supporting Proposition 4.2.1.

However, transformers reveal an intriguing behavioral distinction from convolutional architectures. While CNNs maintained consistent channel importance patterns across different random seeds for identical datasets (Fig. 4.4), transformers exhibit substantial variability in channel configurations even within the same task. We attribute this phenomenon to transformers' superior representational capacity, which enables multiple distinct parameter configurations to achieve equivalent performance on identical problems. This architectural flexibility results in less deterministic channel importance patterns compared to their convolutional counterparts, suggesting that transformer optimization landscapes offer multiple viable pathways to task solutions.

Performance Evaluation Under Different Memory Constraints. Given the substantial scale difference between transformer and CNN architectures—SwinT models are approximately 26-30× larger while BERT/RoBERTa models are 77-90×

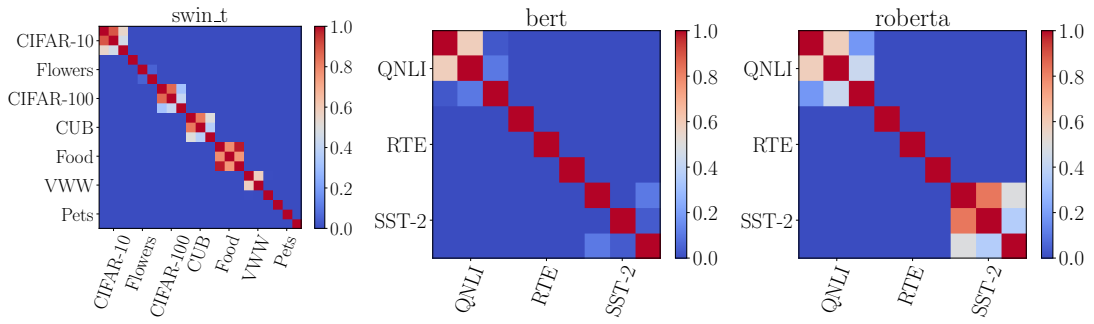


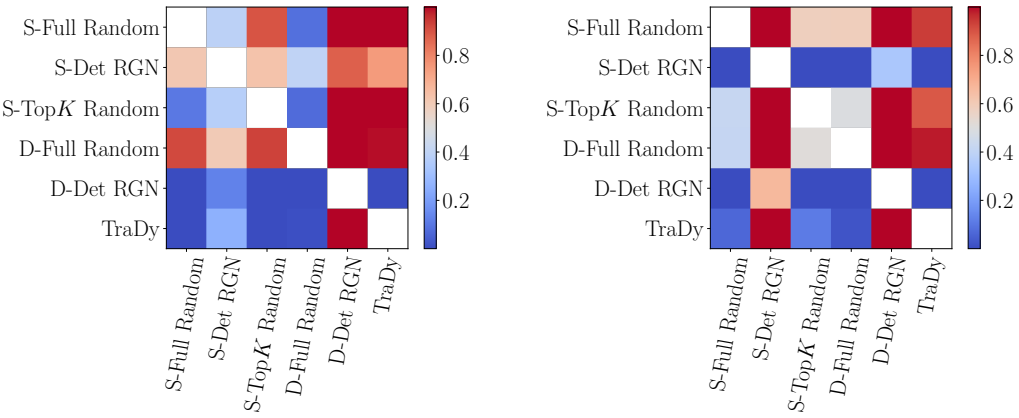
Figure 4.9: T-test of channel gradient norm across seeds and datasets. Credits to Quélennec et al. [2025a].

larger in terms of combined memory requirements—we adopt a dual-constraint evaluation strategy:

1. *Absolute Budget Matching*: Applying identical memory budgets as CNN experiments, creating extreme constraint scenarios where budgets represent less than 0.1% of total network memory.

2. *Proportional Budget Matching*: Scaling budgets to maintain equivalent network proportions as CNN experiments, enabling more substantial updates.

This dual approach enables comprehensive assessment of our method's effectiveness across varying constraint severity while providing insights into transformer fine-tuning behavior under different resource limitations.
Remarkably, all channel selection methods achieve superior accuracy on SwinT compared to CNN architectures, even under the most restrictive memory constraints. Both vision and NLP transformers demonstrate smaller accuracy degradation between constrained budgets and full fine-tuning baselines compared to CNNs, despite operating under proportionally more severe memory limitations. This resilience highlights transformers' ability to learn robust, transferable representations during pre-training that remain effective with minimal parameter adjustments during downstream adaptation. The complete result tables are available in Sec. A.3.2 of the appendix.
Statistical Analysis and Method Comparison. Fig. 4.10 extends our statistical analysis from the main experiments to transformer architectures. For SwinT (Fig. 4.10a), we observe consistent strategy rankings with D-RGN Deterministic achieving optimal performance, followed by TraDy. However, BERT-family models present more complex patterns, where both dynamic approaches appear to be outperformed by S-RGN Deterministic, though this observation carries greater uncertainty due to limited experimental sample size.



(a) SwinT model

(b) BERTs models

Figure 4.10: T-test comparisons of average final test accuracies across multiple experimental dimensions for each group of transformer architectures. Credits to Quélennec et al. [2025a].

The substantial architectural complexity of BERT models suggests that our top-K layer selection methodology, while highly effective for CNNs and moderately sized transformers, may require more sophisticated calibration techniques for architectures of this magnitude. This finding indicates potential directions for extending our framework to accommodate the unique characteristics of very large transformer models.

4.4 Limitations and Conclusions

This chapter has presented TraDy, a theoretically-grounded framework that addresses the limitations of empirical approaches by leveraging heavy-tailed gradient analysis for dynamic subnetwork selection. Our contributions establish three key insights: the heavy-tailed nature of gradients during transfer learning creates exploitable sparsity patterns, layer importance rankings remain architecturally consistent across tasks, while channel importance distributions are inherently task-dependent.

The experimental validation demonstrates that TraDy achieves superior performance compared to existing static approaches by combining principled layer preselection with stochastic channel resampling. The framework successfully bridges theoretical understanding with practical deployment constraints, showing consistent improvements across both convolutional and transformer architectures while maintaining strict memory budgets.

However, several limitations highlight opportunities for further advancement. First,

our layer selection methodology relies on a fixed 97% RGN threshold, which may not optimize resource allocation across different memory budgets or architectural families. Second, while uniform random channel sampling provides computational efficiency, it does not exploit the task-specific channel importance patterns we have identified. The framework treats all channels within selected layers equally, potentially missing opportunities to leverage gradient magnitude information for more informed selection decisions. Finally, while our method proves efficient in CNN scenarios, the results obtained on transformer architectures are underwhelming.
These limitations motivate the development of more adaptive approaches that can dynamically adjust both layer selection and channel sampling strategies based on available resources and task requirements. The theoretical foundations established in this chapter—particularly the understanding of gradient sparsity patterns and architectural consistency—provide the necessary groundwork for such extensions. In the following chapter, we present MeDyate, an enhanced framework that addresses these limitations by introducing adaptive layer selection based on memory budgets and incorporating gradient-informed channel sampling while maintaining the computational efficiency principles established here. This progression demonstrates how theoretical insights can be systematically refined into increasingly sophisticated practical solutions for memory-constrained learning.

Chapter 5

MeDyate, an Adapative Channel Selection Strategy

"They did not know it was impossible so they did it"

Mark Twain

Contents

Chapter 5 Abstract

This chapter presents MeDyate (Memory-constrained Dynamic subnetwork update), an advanced framework extending TraDy's dynamic subnetwork selection for memory-efficient fine-tuning. We introduce the LaRa metric for holistic layer characterization and budget-adaptive layer selection that scales with memory constraints. Exploiting the temporal stability of channel importance distributions, MeDyate employs probabilistic channel sampling with gradient-informed probability distributions. By transitioning to raw gradient norms within pre-selected layers, our approach reduces computational overhead while preserving importance ranking information. Experimental validation demonstrates that MeDyate consistently achieves superior performance while maintaining strict memory constraints.

5.1 TraDy Framework Extension

The TraDy algorithm introduced in Chapter 4 established a solid foundation for dynamic subnetwork selection in memory-constrained environments by leveraging three key insights: the heavy-tailed behavior of stochastic gradients during transfer learning, the architectural consistency of layer importance across downstream tasks, and the task-dependent variability of channel importance distributions. Through dynamic channel resampling within pre-selected layers, TraDy demonstrated significant improvements over static selection approaches, achieving state-of-the-art performance while respecting strict memory constraints for both parameters and activations.
However, our comprehensive analysis of TraDy's performance characteristics revealed several limitations that warranted further investigation and methodological refinement. Building upon these insights, we present in this chapter our **Me**mory-constrained **Dy**namic subnetwork upd**ate** algorithm (MeDyate), a comprehensive extension of the TraDy framework that addresses these identified limitations while preserving the core theoretical foundations that made the original approach successful.

5.1.1 TraDy Limitations

Suboptimal Layer Ranking and Selection Methodology. As described in Sec. 4.2.3 of the previous chapter, our TraDy approach characterizes layer importance through the sum of its channels' RGN values (Eq. (4.8)). While this metric embodies an relevant approach to characterizing layers with respect to both

gradient magnitude and memory cost, it remains fundamentally channel-centric, thus failing to capture holistic layer behavior. This granularity mismatch becomes problematic given the binary nature of layer selection, where layers are either completely frozen or made available for channel sampling. To address this limitation, we introduce **LaRa**, a novel metric for **La**yer **Ra**nking that enables improved layer characterization while maintaining the crucial property of stability across downstream tasks.

Furthermore, TraDy employs a static threshold approach for layer selection, determining the number K of layers to include in the update space by applying a fixed criterion that captures 97% of the cumulative RGN across all layers, regardless of the available memory budget. This approach fails to adapt to varying memory constraints and does not optimize the trade-off between search space exploration and memory utilization. When memory budgets are particularly constrained, maintaining an excessively large search space dilutes the available memory resources across numerous layers, thereby reducing the algorithm's capacity to sufficiently explore channels within each layer and potentially degrading on-task performance. In this chapter, we propose an adaptive layer selection strategy that scales the number of selected layers according to the magnitude of the available memory budget.

Incomplete Channel Characterization and Selection. In Sec. 4.2.2, we characterized channel importance through RGN values (Eq. (4.7)) and demonstrated that channel importance distributions are task-dependent and thus cannot be predetermined, unlike layer rankings. Given that memory constraints prevent computing the complete channel importance distribution simultaneously, TraDy resorts to random channel selection within the preselected set of important layers. While this approach proved effective in demonstrating the advantages of dynamic channel selection over static methods, it remains conceptually limited as it fails to exploit available channel importance information. Building upon the insights regarding channel importance developed in the previous chapter, MeDyate advances beyond random selection toward task-adaptive channel selection strategies that leverage partial importance information to guide the selection process.

5.1.2 Introducing MeDyate

Building upon these observations, this chapter introduces **MeDyate**, a comprehensive extension of the TraDy framework that addresses the identified limitations through several key theoretical and algorithmic innovations.

Layer-Wise Improvements. We propose the Layer Ranking (LaRa) methodology that replaces TraDy's channel-centric RGN summation approach with a holistic layer-level metric. LaRa incorporates both the Euclidean norm of the

entire layer gradient tensor and its complete memory cost, enabling more accurate layer importance assessment that aligns with the binary nature of layer selection decisions.

Moving beyond TraDy's fixed layer selection approach, MeDyate introduces a principled framework for adaptive layer selection through the parameter β_K, which represents the ratio between the total memory budget and the memory footprint of the K selected layers. This formulation enables dynamic adjustment of the search space based on available memory constraints, ensuring that our sampling strategy operates within an appropriately sized parameter space that optimally balances comprehensive channel exploration with practical memory limitations.

Channel-Wise Improvements. Since LaRa-based layer exclusion eliminates inefficient high-memory layers a priori, MeDyate can safely transition from memory-reweighted metrics to raw gradient norms as channel importance indicators within the selected layer subset. This simplification reduces computational overhead while preserving the essential importance ranking information required for effective channel selection.

Furthermore, we demonstrate that while channel importance distributions vary across different downstream tasks, they exhibit significant stability within a given dataset following a brief initial stabilization phase. This stability insight enables MeDyate to develop a sophisticated dynamic channel sampling strategy that incorporates stochasticity with gradient-informed probability distributions, facilitating more effective exploration of the channel space while maintaining strict memory constraints. Rather than TraDy's uniform random selection, MeDyate implements a probabilistic channel selection mechanism where update probabilities are proportional to channel gradient magnitudes. This approach ensures comprehensive coverage of all channels within selected layers while naturally prioritizing channels with higher importance, ultimately leading to superior task adaptation performance.

5.1.3 Chapter Organization

The remainder of this chapter is structured to provide a comprehensive development of the MeDyate framework, from theoretical foundations to experimental validation. Sec. 5.2 presents the methodological construction of MeDyate through four complementary components. We begin in Sec. 5.2.1 by introducing the LaRa methodology for improved layer ranking and our principled approach to adaptive layer selection based on memory budget constraints. Sec. 5.2.2 then analyzes the stability properties of channel importance distributions within downstream tasks, providing the theoretical foundation for our enhanced channel selection strategy. Building upon these insights, Sec. 5.2.3 presents the complete algorithmic formulation of MeDyate, detailing the integration of layer and channel selection

mechanisms into a unified framework. Sec. 5.2.4 concludes the methodological development with an analysis of the algorithm's convergence properties along a study of its computational complexity.
Sec. 5.3 provides extensive experimental validation of MeDyate across multiple network architectures, datasets, and memory budget configurations. Through systematic comparisons with existing approaches, we demonstrate the consistent performance improvements achieved by our methodological innovations while maintaining strict memory constraints. Finally, Sec. 5.4 summarizes the key contributions of this work and outlines promising directions for future research in memory-efficient dynamic subnetwork selection.

5.2 Building MeDyate

This section develops the core methodological innovations that distinguish MeDyate from its TraDy predecessor. We systematically address each identified limitation through targeted improvements: introducing the LaRa metric for holistic layer characterization and budget-adaptive selection, exploiting channel importance stability for informed sampling strategies, and integrating these components into a unified algorithmic framework with proven theoretical properties.

5.2.1 LaRa and Adaptive Layer Selection

The theoretical framework established in Chapter 4 demonstrated that heavy-tailed gradient distributions during transfer learning enable effective layer importance ranking (see Sec. 4.2.3). TraDy's approach to layer characterization relied on aggregating individual channel RGN values within each layer (Eq. (4.8)). While this summation-based metric incorporates both gradient magnitude and memory efficiency considerations, it suffers from several conceptual limitations that compromise its effectiveness for holistic layer assessment.
The fundamental issue lies in the metric's sensitivity to outlier channels with exceptionally high gradient norms, which can dominate the layer's overall score regardless of the broader channel distribution within that layer. Since all channels within a given layer share identical spatial dimensions, the memory reweighting factors become uniform across channels and can be factored out of the summation (Eq. (4.8)). This property reveals that the metric primarily reflects the behavior of few channels within the layer, scaled by a constant channel-specific memory factor, rather than capturing genuine layer-wide behavioral patterns.
Such characterization becomes particularly problematic given the binary decision-

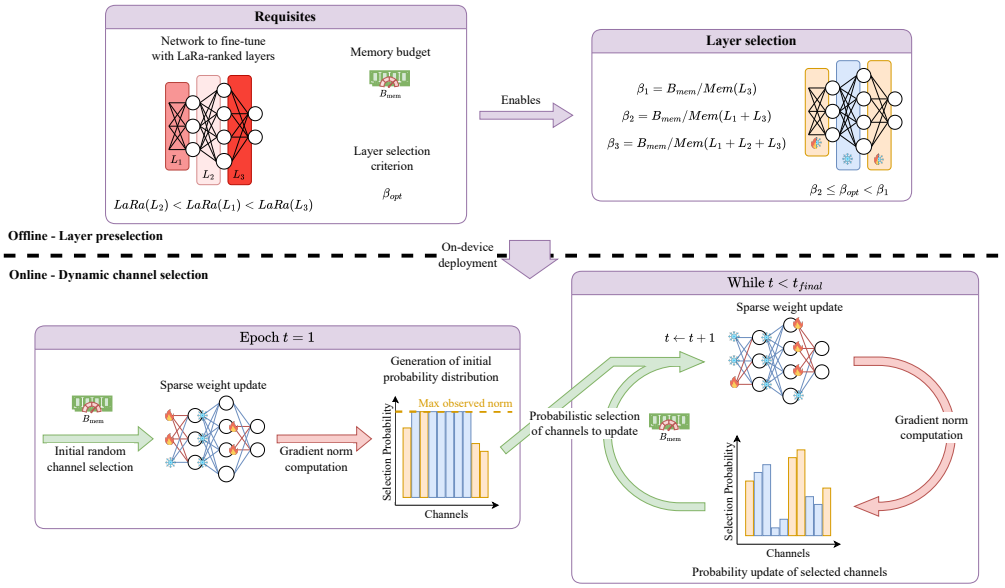


Figure 5.1: Overview of the MeDyate framework. The offline phase (top) shows layer preselection based on LaRa rankings and memory budget constraints, while the online phase (bottom) illustrates dynamic channel sampling with importance-weighted probabilities during training epochs. Credits to Quélennec et al. [2025b].

making process inherent to layer selection strategies. In our framework, layers are either completely excluded from updates throughout training or made fully available for stochastic channel sampling. The mismatch between this binary treatment and a metric that fundamentally reflects individual channel characteristics within layers creates suboptimal layer selection decisions that fail to account for the comprehensive utilization patterns expected under dynamic sampling strategies.

LaRa Metric Formulation. To address this limitation, we propose LaRa (**La**yer **Ra**nking), a holistic layer-level metric, corresponding to the Euclidian norm of the entire layer gradient tensor, reweighted by its complete memory cost:

$$\text{LaRa}_i = \frac{\left\| \left(\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i} \right)_c \right\|_2}{\sum_{c=1}^{C} \left(\mathcal{C}_c^{\mathcal{W}_i} + \mathcal{C}_c^{\mathcal{A}_i} \right)}, \tag{5.1}$$

where $\left\| \frac{\partial L}{\partial \mathcal{W}_i} \right\|_2$ represents the Euclidean norm of the entire layer gradient tensor. This metric preserves the architectural consistency property established in Sec. 4.2.3, as the theoretical inequalities for layer ranking stability are based on Euclidean norms. Empirical validation confirms that LaRa maintains high Spearman correlation

coefficients across downstream tasks, thus enabling reliable a priori layer selection (Sec. 5.3.1).

Budget-Adaptive Layer Selection. The layer selection policy developed for TraDy, as described in Sec. 4.3.1, employed a fixed layer count for each architecture regardless of memory constraints. To address this limitation, we introduce a memory budget-adaptive layer selection strategy that dynamically adjusts the number of selected layers based on available resources.

We define the selection parameter β_K as the ratio between the total memory budget B_{mem} and the memory footprint M_K of the top K layers ranked in decreasing order according to their LaRa values:

$$\beta_K = \frac{B_{\text{mem}}}{M_K} \tag{5.2}$$

This formulation enables principled adjustment of the search space based on memory constraints, addressing the fundamental trade-off between exploration breadth and memory utilization efficiency. When memory budgets are particularly constrained, maintaining an excessively large search space dilutes available resources across numerous layers, thereby reducing the algorithm's capacity to sufficiently explore channels within each individual layer.

In practice, we select K_{opt}, the optimal number of layers such that $\beta_{K_{\text{opt}}} \leq \beta_{\text{opt}} < \beta_{K_{\text{opt}}-1}$, where β_{opt} serves as the selection boundary hyperparameter. The hyperparameter search process for determining an appropriate value of β_{opt} is detailed in Sec. 5.3.1. Combined with our LaRa ranking methodology, this adaptive approach enables the selection of fewer layers compared to TraDy's fixed strategy while maintaining superior performance. The synergistic effects between this reduced layer count and our task-adaptive channel selection algorithm are further explored in Sec. 5.2.4.

5.2.2 Channel Importance Stability

While Chapter 4 established that layer importance rankings remain architecturally consistent across downstream tasks , the situation differs significantly at the channel level (Sec. 4.2.3). As demonstrated, channel importance distributions exhibit substantial task-dependent variations that prevent reliable *a priori* channel selection across different downstream tasks. This task-specificity stems from the fundamental composition of weight derivatives, where both activation maps and activation derivatives are inherently shaped by task-specific data characteristics and loss landscape properties (Eq. (4.3)).

Since memory constraints prevent full gradient computation, our TraDy framework initially assumed that obtaining channel importance distributions on-device was impractical. However, subsequent analysis of channel importance evolution over

time revealed a critical insight that enables effective importance-adaptive channel selection within memory-constrained environments: channel importance distributions exhibit significant temporal stability within a given downstream task throughout the fine-tuning process. More precisely, the evolution of channel importance during fine-tuning unfolds through two distinct phases:

- **Rapid Stabilization Phase:** During the initial training epochs, channel importance distributions undergo rapid changes as the network adapts its representations to the new task requirements. This period corresponds to the most significant weight modifications, where high loss and gradient magnitudes drive substantial parameter adjustments. The duration of this phase varies based on the similarity between pre-training and downstream task domains, typically spanning the first few epochs for closely related tasks.

- **Distributional Consistency Phase:** Following the initial adaptation period, channel topology stabilizes and maintains structural consistency throughout the remainder of the fine-tuning process. While absolute gradient magnitudes typically decrease as the network converges toward a local minimum, the relative importance rankings among channels remain largely preserved.

This temporal behavior can be understood through the dynamics of fine-tuning pre-trained architectures. As established in Sec. 2.2, the fundamental assumption underlying transfer learning is that downstream task features exhibit sufficient similarity to pre-training representations, necessitating only targeted adaptations rather than complete relearning. In the early stages of fine-tuning, the network must adapt to the unseen features of the new task, particularly at the classifier level which is typically randomly initialized. This adaptation process results in relatively high loss values that propagate through the network, leading to high gradient magnitudes which in turn drive significant weight changes.
As the network rapidly converges and weight modifications become more incremental, this stability propagates through the computational graph: stabilized weights $\mathcal{W}_i$ combined with fixed input datasets lead to stable activations $\mathcal{A}_i$, which cascade to stable activation derivatives and ultimately to stable weight derivatives according to the chain rule formulation established in Sec. 3.1.1.
The addition of temporal stability to our understanding of channel behavior allows us to overcome TraDy's limitation that channel importance cannot be reliably estimated on-device. During the distributional consistency phase, the preserved relative rankings among channels mean that partial gradient information collected through memory-constrained sampling can provide reliable estimates of the complete channel topology. This theoretical insight bridges the gap between TraDy's

uniform random selection and computationally prohibitive full gradient computation.
Building on the established superiority of dynamic selection approaches demonstrated in Sec. 4.3.2, MeDyate leverages this stability insight to develop gradient-informed probabilistic channel selection mechanisms. The visualization of channel importance evolution throughout training is provided in Sec. 5.3.1.

5.2.3 MeDyate Algorithm Formulation

Algorithm 2: MeDyate. Credits to Quélennec et al. [2025b].

Require: Pre-trained backbone weights $\mathcal{W}$, LaRa ranked layers L, initial empty channel norm vector N, number of epochs n, Train data D_{train}, Test data D_{test}, memory budget B_{mem}.
Hyper-Parameters: Top K layers selection criterion β_{opt}.
Offline Layer Selection: Select the set of top K_{opt} layers $\{L_{K_{\text{opt}}}\}$ such that $\beta_{K_{\text{opt}}} \leq \beta_{\text{opt}} < \beta_{K_{\text{opt}}-1}$.
Online Channel Selection:
for *epochs* $t = 1$ ***to*** n **do**
  if $t = 1$ **then**
    Randomly sample channels $\{C^t\}$ within $\{L_{K_{\text{opt}}}\}$ along uniform probability distribution until the memory budget B_{mem} is met.
  else if $t > 1$ **then**
    Compute gradient norm $\left\| \left(\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i} \right)_c \right\|_2$ of channels selected at previous epoch.
    Update the corresponding norm values in N.
    if $t = 2$ **then**
      Set the norm value for non-selected channels $\{\bar{C}^t\}$ to be the maximum norm value observed, $\max_c \left[\left\| \left(\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i} \right)_c \right\|_2 \right]$.
    Resample channels $\{C^t\}$ along probability distribution proportional to N until the memory budget B_{mem} is met.
  Update the weights of the selected channels $\{C^t\}$ using D_{train}.
Evaluate the fine-tuned backbone using D_{test}

The previously established LaRa-based layer ranking, budget-adaptive layer selection, and channel importance stability converge to enable the design of MeDyate's dynamic channel sampling algorithm. This section presents the complete algorithmic formulation that integrates these theoretical foundations into a unified

framework for memory-constrained fine-tuning.
Drawing upon the established temporal stability of channel importance within downstream tasks, MeDyate's core innovation lies in leveraging partial gradient information collected during training to construct and maintain probability distributions that reflect channel importance, while ensuring comprehensive exploration of the channel space within memory constraints.
Instead of relying on the RGN metric introduced in TraDy's framework to quantify channel importance, MeDyate employs raw gradient norms as the fundamental importance measure. This algorithmic design choice exploits the fact that memory cost reweighting becomes redundant when operating within pre-selected efficient layers, allowing raw channel norms to serve as direct measures of importance. The transition is further motivated by empirical evidence from Fig. 4.5, which demonstrates that deterministic raw norm selection can outperform its RGN counterpart when the least efficient layers are excluded from consideration. Additionally, as established by Wan et al. [2023], raw gradient norms effectively capture the concentration of information in channels due to the heavy-tailed nature of stochastic gradients demonstrated in Sec. 4.2.1. This transition from memory-reweighted metrics to raw gradient norms reduces computational overhead while preserving essential importance ranking information.
The core innovation of MeDyate compared to TraDy lies in the implementation of a probabilistic selection strategy that exploits observed channel importance patterns. MeDyate begins with uniform random sampling of channels, but after each update cycle, selected channels receive new selection probabilities proportional to their computed gradient norms. This mechanism ensures comprehensive coverage of all channels within selected layers while naturally prioritizing those demonstrating higher importance through their gradient magnitudes. This stochastic approach balances exploration of the channel space with exploitation of discovered importance patterns.
Algorithm 2 provides the complete description of our proposed MeDyate strategy. The algorithm assumes pre-computed LaRa metrics obtained through a preliminary fine-tuning run on available downstream task data. This offline pre-computation exploits the demonstrated stability of layer rankings across tasks, necessitating a single brief training session to establish the layer importance hierarchy.
MeDyate begins offline with the adaptive layer selection mechanism determining the optimal number of layers K_{opt} such that $\beta_{K_{\text{opt}}} \leq \beta_{\text{opt}} < \beta_{K_{\text{opt}}-1}$, where β_{opt} represents the selection boundary hyperparameter established in Sec. 5.2.1 (line 1). The online training process follows a structured progression through distinct phases. Initially, channels undergo uniform random sampling within the predetermined layer set $L_{K_{\text{opt}}}$ while respecting memory budget constraint B_{mem} (line 5). During the second epoch, gradient norms are computed for previously selected channels

(line 8), enabling the construction of an informed sampling distribution. The algorithm assigns observed norm values to previously sampled channels while allocating the maximum observed norm value to unselected channels (line 10). This maximum norm assignment strategy capitalizes on the typical decreasing trend of gradient magnitudes during fine-tuning, as analyzed in Sec. 5.2.2. By providing unobserved channels with maximum selection probability, the algorithm encourages rapid exploration of the complete channel space.
Following distribution construction, channels undergo resampling according to probabilities proportional to the gradient norm vector while maintaining memory budget compliance (line 11), and selected channels receive weight updates (line 12). From the third epoch forward, the algorithm omits the maximum norm assignment procedure (line 10) and enters an iterative cycle of channel resampling and updating based on the evolving probability distribution derived from the observed gradient norms. Training concludes with model performance evaluation on the test dataset (line 13).
Fig. 5.1 illustrates the operational flow of our MeDyate algorithm across its two distinct phases.

5.2.4 Algorithm Properties Study

This section examines two key properties of the MeDyate algorithm that validate its practical applicability in memory-constrained environments. We first analyze the computational overhead introduced by gradient norm computation and probabilistic channel selection, demonstrating that these additional operations remain negligible compared to standard gradient computation operation. Subsequently, we establish theoretical bounds on the algorithm's convergence behavior in terms of search space exploration, showing that MeDyate achieves comprehensive channel coverage within relatively few training epochs.

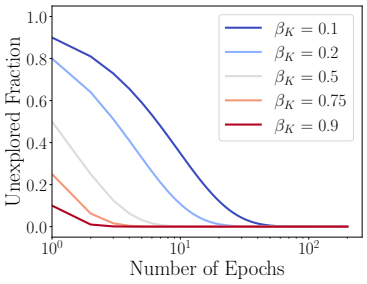


Figure 5.2: Evolution of u_e as a function of epochs for different β values. Credits to Quélennec et al. [2025b].

Computational Overhead Analysis. MeDyate introduces additional online computational complexity compared to approaches like TraDy's random selection or Lin et al. [2022]'s pre-computed SU schemes, primarily due to the requirement of computing gradient norms between epochs and executing probabilistic sampling procedures. For a given epoch t, let γ_t denote the set of selected channels with cardinality $|\gamma_e|$. The computational complexity of computing gradient norm metrics for selected channels

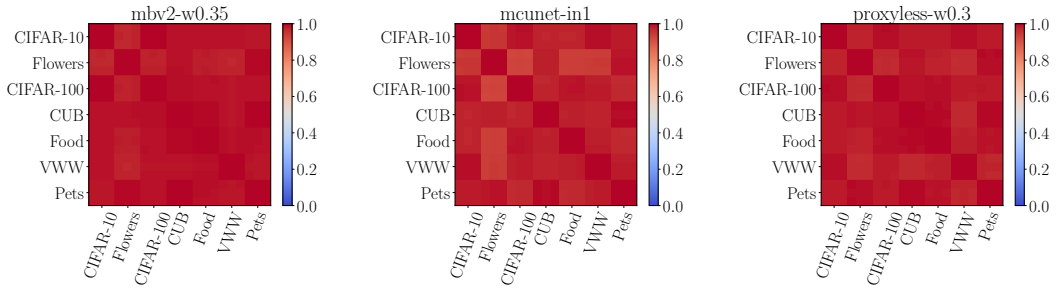


Figure 5.3: Spearman correlation of layer cumulated LaRa metric across seeds and datasets. Credits to Quélennec et al. [2025b].

between epochs, assuming uniform channel dimensions, is:

$$(\Theta_{\text{time}})_t^{RN} = |\gamma_t| C' D^2. \tag{5.3}$$

The complexity of probabilistic sampling within a distribution of length N is $\mathcal{O}(\log(N))$ per channel, making the total computational complexity of MeDyate for epoch t:

$$(\mathcal{O}_{\text{time}})_t^{\text{Med}} = |\gamma_e|(C' D^2 + \log(N)). \tag{5.4}$$

To contextualize this overhead, we compare it to the computational cost of weight derivative computation during backpropagation. Let M denote the number of steps per epoch. The computational complexity for gradient computation during epoch t is:

$$(\Theta_{\text{time}})_t^{\text{grad}} = M|\gamma_t| D^2 C' H' W'. \tag{5.5}$$

Additionally, forward propagation and loss backpropagation demonstrate comparable computational complexity while operating on the entire network architecture rather than solely on the updated channel subset. Considering the typical orders of magnitude characterizing these variables, MeDyate's computational overhead remains negligible relative to the overall backpropagation cost. We anticipate a modest increase in inter-epoch latency, establishing a trade-off between enhanced performance and computational overhead.

Search Space Exploration Convergence. Due to the maximum probability assignment for unexplored channels, MeDyate is expected to demonstrate enhanced search space exploration compared to uniform random selection approaches. We employ uniform random sampling as a conservative baseline for analyzing MeDyate's convergence characteristics.

At each epoch, channels are selected according to the memory budget, which by design represents a fraction $\beta_{K_{\text{opt}}}$ of the total memory within the selected layer search space. If u_t denotes the proportion of unexplored search space at epoch t, we can write $u_t = (1 - \beta_{K_{\text{opt}}})^t$ since, with random selection, the sampled channels

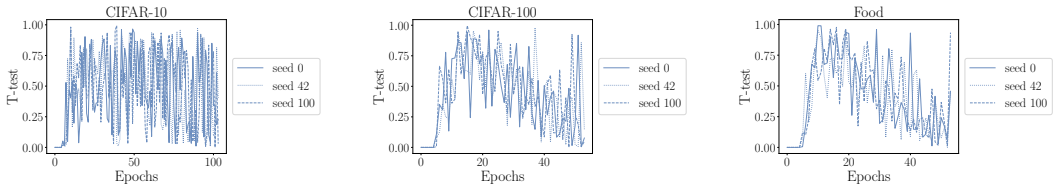


Figure 5.4: Evolution of channel gradient norm T-test over time, MobileNetV2 fine-tuned on 3 downstream tasks. Credits to Quélennec et al. [2025b].

at each epoch are independent of each other. This expression clearly highlights the interdependence between the number of selected layers and the memory budget as key factors determining our algorithm's capacity to explore the available search space. Fig. 5.2 illustrates this convergence behavior, demonstrating that MeDyate can construct comprehensive gradient representations within relatively few epochs depending on the chosen β value.
In practice, we have $\beta_{K_{\text{opt}}} \simeq \beta_{\text{opt}}$, and our hyperparameter search detailed in Sec. 5.3.1 yields $\beta_{\text{opt}} = 0.2$. Under random search conditions, this relationship indicates that 80% of channels are observed within 8 epochs, 90% within 11 epochs, and 95% within 14 epochs. These values represent upper bounds since MeDyate is expected to outperform random approaches. These rapid convergence rates prove essential, as they demonstrate that within just a few epochs, MeDyate can construct comprehensive gradient representations, enabling performance comparable to methods with complete gradient knowledge.

5.3 Experimental Results

The experimental evaluation of MeDyate follows the methodological framework established in Chapters 3 and 4. We employ the same network architectures, downstream datasets, memory budget configurations, and evaluation protocols to ensure consistent comparison across our subnetwork selection approaches. This standardized experimental context enables direct assessment of MeDyate's contributions relative to the baseline methods previously evaluated.

5.3.1 Properties Validation

This subsection provides empirical validation of the properties established in the previous sections. We systematically verify the architectural consistency of our LaRa metric, demonstrate the temporal stability of channel importance distributions, and validate the effectiveness of our proposed modifications through controlled

experiments. These validations establish the empirical foundations that justify MeDyate's design choices and theoretical claims.

LaRa Metric Architectural Consistency. Our first validation examines whether the architectural consistency of layer importance rankings, established for RGN-based metrics in Sec. 4.2.3, extends to our proposed LaRa metric from Sec. 5.2.1. Following the methodology employed in Sec. 4.3.1, we construct layer rankings based on LaRa values computed according to Eq. (5.1). Each fine-tuning experiment generates accumulated LaRa values across training epochs, creating a "layer topology" profile that captures the relative importance hierarchy within the network architecture.

The evaluation methodology mirrors the approach used for RGN validation: we compute Spearman rank correlation coefficients across all pairwise combinations of fine-tuning experiments, generating comparison matrices analogous to those presented in Fig. 4.3.

As demonstrated in Fig. 5.3, our LaRa metric successfully maintains the architectural consistency property observed with RGN-based rankings. The correlation analysis reveals that across all evaluated network architectures, correlation coefficients between downstream task pairs consistently exceed 0.8, with most comparisons achieving correlations above 0.9. These results establish that our LaRa metric preserves the essential characteristic that layer importance hierarchies remain stable across diverse downstream tasks, regardless of variations in data distributions and task-specific requirements. The observed consistency supports the practical approach of performing layer ranking through preliminary fine-tuning on any available downstream task data, then applying the derived LaRa-based hierarchy to the target task.

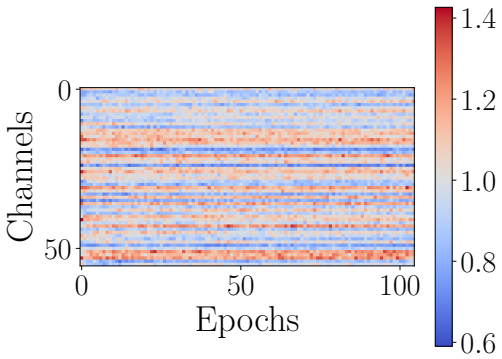
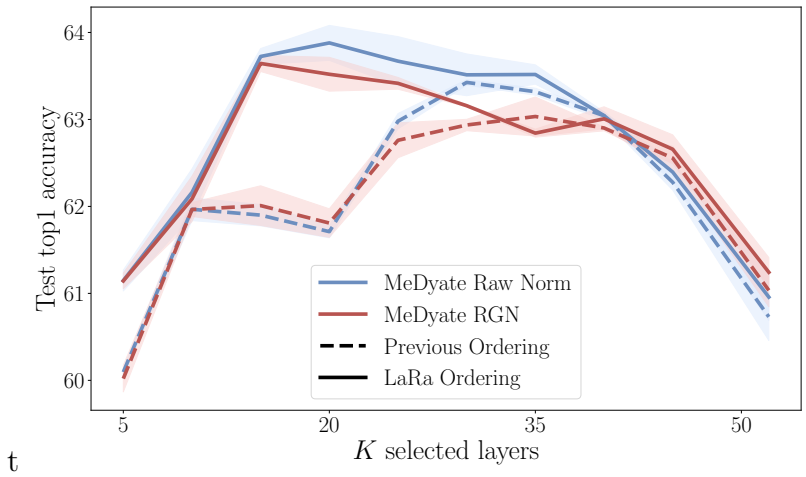


Figure 5.5: Evolution of channel gradient norm over time within a specific layer of a MobileNetV2 fine-tuned on CIFAR-10. Results are normalized per epoch for visualization. Credits to Quélennec et al. [2025b].

Channel Importance Temporal Stability. Our second validation focuses on the empirical verification of channel importance stability throughout training, extending the theoretical analysis from Sec. 5.2.2. We investigate the temporal dynamics of channel gradient distributions during transfer learning by employing statistical tests that compare channel gradient norms between successive epochs,



t

Figure 5.6: Performance comparison of layer ranking methods (LaRa vs. Previous ordering) and channel importance metrics (raw gradient norm vs. RGN) when fine-tuning MobileNetV2 on Food dataset under memory constraints. Results show test top-1 accuracy across different numbers of selected layers K. Credits to Quélennec et al. [2025b].

enabling assessment of distributional consistency over time.
Fig. 5.4 presents the statistical analysis results, which exhibit a characteristic two-phase evolution pattern aligned with our theoretical framework. The initial training phase demonstrates p-values consistently approaching zero, signifying substantial modifications in gradient norm distributions as the network undergoes adaptation to novel task characteristics. Subsequently, the p-values are consistently non-null, thus representing the failure to reject the distributional similarity hypothesis. These statistical findings provide empirical validation for the temporal stability assumptions underlying MeDyate's probabilistic sampling framework.
Complementing the statistical analysis, Fig. 5.5 provides visual evidence of temporal consistency in relative channel importance rankings. The visualization tracks channel gradient norm evolution within a given layer during MobileNetV2 fine-tuning on CIFAR-10, employing epoch-wise normalization to account for gradient magnitude decay and improve visual interpretation. The resulting trajectory patterns demonstrate that channels exhibiting consistently elevated or diminished gradient norms maintain their relative positions throughout training, forming distinct and persistent "streams" across the fine-tuning duration. This visual analysis reinforces our statistical findings by demonstrating that relative gradient norm hierarchies among channels exhibit temporal consistency, thereby providing additional empirical validation for the theoretical framework presented in Sec. 5.2.2.

LaRa and Raw Gradient Norm Validation. Our third validation assesses the

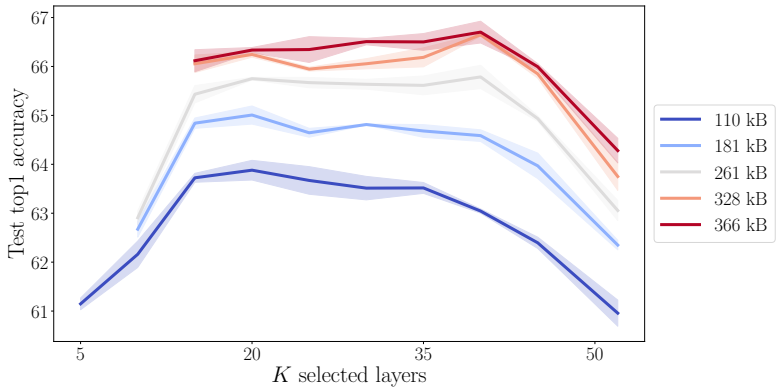


Figure 5.7: Final MeDyate test top1 accuracies depending on the number of top K layers for different memory budgets. Credits to Quélennec et al. [2025b].

performance improvements achieved through two methodological innovations: the LaRa-based layer ordering compared to the RGN-based approach from Sec. 4.2.3, and the adoption of raw gradient norms as channel importance indicators. We conduct a controlled comparison using MobileNetV2 fine-tuned on the Food dataset under the most restrictive memory budget, evaluating four experimental configurations that combine our LaRa ranking against the original TraDy layer hierarchy, with each pairing tested using both raw gradient norm and RGN metrics for channel importance assessment.

The comparative analysis presented in Fig. 5.6 displays final test top-1 accuracies across different numbers of selected layers K. The experimental evidence establishes the distinct advantage of raw gradient norms over RGN metrics across both layer ranking methodologies, corroborating our claim that memory cost reweighting provides diminishing returns when applied within pre-selected efficient layer subsets. Additionally, the LaRa-based layer hierarchy demonstrates consistent superior peak performance compared to the previous ranking system while achieving optimal results with reduced layer requirements. These combined benefits validate both the efficacy of our comprehensive layer assessment and its practical advantage for memory-constrained sampling applications.

We attribute this performance improvement to the exclusion of depthwise layers from the best-ranked layers with LaRa. In our TraDy framework, these layers were standing as the best-ranked layers, even though Lin et al. [2022] observed that they did not contribute to accuracy gain (Fig. 3.1). These layers were taking priority in our previous layer ranking mostly due to their comparatively low channel memory cost. Their effect can be observed in Fig. 5.6 as the accuracy does not improve between the top 10 and top 20 layers, showcasing how the addition of these layers does not add important channels for update. Comparatively, our LaRa ordering

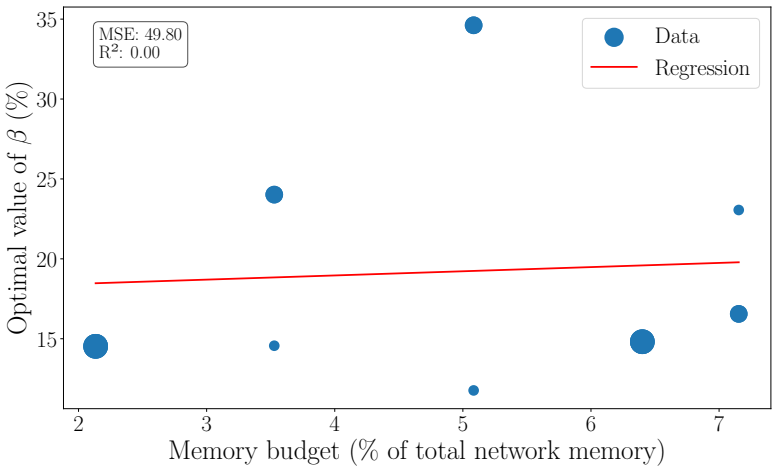


Figure 5.8: Regression on the optimal β_{opt} obtained for different memory budgets and seeds. Credits to Quélennec et al. [2025b].

results in higher performance from the 5 layers mark and consistently increases as the first level of layers are added.

Budget-Adaptive Layer Selection. The performance analysis in Fig. 5.6 reveals that MeDyate's effectiveness varies with the number of selected layers K, achieving optimal results at $K_{\text{opt}} = 20$. This observation confirms the need for an investigation into the relationship between memory constraints and the optimal layer count K_{opt}.

Fig. 5.7 illustrates how final test top-1 accuracy evolves with layer count K across different memory budgets during MobileNetV2 fine-tuning on the Food dataset. The accuracy curves exhibit rightward shifts as budgets increase, supporting our hypothesis that optimal search space dimensions correlate with available memory constraints, as theorized in Sec. 5.2.4. Performance degradation occurs with both overly restrictive and permissive layer selection: insufficient layers exclude crucial parameters, while excessive inclusion dilutes memory resources and impairs convergence.

To establish the quantitative relationship between B_{mem} and β_{opt}, we extract optimal layer counts from Fig. 5.7 and compute corresponding β_K values via Eq. (5.2). Fig. 5.8 displays these β_K values in a scatter plot format against memory budgets (represented as percentages of total network memory), with point dimensions reflecting the frequency of each configuration. While regression analysis demonstrates high variance, the results in Fig. 5.7 indicate that larger memory budgets achieve relatively stable peak accuracy across moderate K ranges, providing flexibility in β_{opt} selection.

Considering the accuracy degradation associated with insufficient layer selection

(corresponding to elevated β_K values), we determine $\beta_{\text{opt}} = 0.2$, positioned slightly above the observed regression trend. This selection accommodates the discrete characteristics of layer selection, where we choose K_{opt} such that $\beta_{K_{\text{opt}}} \leq \beta_{\text{opt}} < \beta_{K_{\text{opt}}-1}$, thereby ensuring adaptive layer selection that optimizes MeDyate performance across varying memory constraints.

5.3.2 Main Results

This subsection presents the comprehensive performance evaluation of MeDyate across our complete experimental framework. The experimental methodology follows the design established in Sec. 4.3.2, where detailed experimental configurations are provided. Our analysis encompasses 189 individual training runs representing the full cross-product of three network architectures, seven downstream datasets, three memory budget levels, and three random seeds.

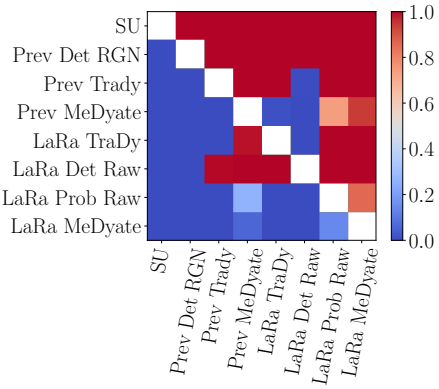


Figure 5.9: T-test comparisons of average final test accuracies across multiple experimental dimensions. Credits to Quélennec et al. [2025b].

Comparative Performance Analysis. Our experimental evaluation encompasses both static and dynamic selection methodologies. The comparative framework includes Lin et al. [2022]'s SU method as the representative static baseline, while all other evaluated strategies employ dynamic channel resampling between epochs, consistent with the demonstrated superiority of dynamic approaches hinted at in Chapter 3 and further established in Chapter 4.

The strategy nomenclature differentiates between layer ranking methodologies: approaches with "Prev" prefixes utilize the original TraDy layer ordering with fixed layer counts, while "LaRa" prefixed strategies employ our proposed layer ranking with budget-adaptive layer selection. Our evaluation examines several key strategies from the TraDy framework, including deterministic RGN selection that leverages gradient pre-computation to select channels maximizing RGN within memory constraints, and the original TraDy approach for random channel sampling within pre-selected layers.

To isolate the contributions of our methodological components, we implement cross-combinations of layer ranking approaches with different channel selection strategies. This includes deterministic raw gradient norm selection within our LaRa framework, MeDyate applied with previous layer selection methodology, and TraDy adapted to our layer ranking system. Additionally, we evaluate a probabilistic

gradient norm strategy that pre-computes channel gradient norms and converts them to sampling probabilities, serving as a theoretical upper bound for MeDyate's performance by providing complete gradient knowledge during channel selection.

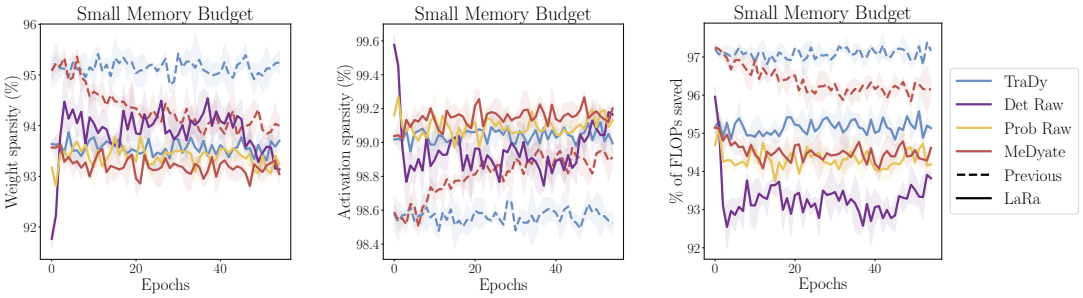
Statistical Analysis and Results. Fig. 5.9 presents the statistical analysis results through paired T-tests comparing average final test top-1 accuracies across all experimental conditions. Each matrix cell represents a statistical T-test examining whether the row strategy achieves superior mean accuracy compared to the column strategy. Full results tables are available in Sec. A.3.3.

The results strongly support our methodological design choices. MeDyate achieves the highest performance across all evaluated strategies, with results that remarkably appear to exceed those of the probabilistic gradient norm strategy despite operating with incomplete gradient information, indicating highly effective convergence under memory constraints. The third-ranked performance of Prev MeDyate demonstrates that the core algorithmic principles remain effective even when combined with suboptimal layer ordering and selection, highlighting the robustness of our approach. The strong performance of LaRa TraDy further validates our layer ranking methodology, showing that improved layer selection can enhance existing dynamic strategies. This approach presents a compelling alternative when computational overhead considerations favor simpler sampling policies over MeDyate's gradient norm computation requirements. The consistent underperformance of raw gradient norm deterministic selection compared to its probabilistic counterparts reinforces the established advantage of stochastic sampling strategies in memory-constrained environments.

Efficiency Metrics Analysis. Our algorithm achieves these performance gains while simultaneously reducing computational overhead and maintaining high sparsity levels. Fig. 5.10 illustrates the evolution of key efficiency metrics when fine-tuning MobileNetV2 on the Food dataset under the smallest memory budget across representative channel selection strategies.

The sparsity analysis reveals the existence of trade-offs between different approaches. While all methods achieve comparable overall sparsity levels with weight sparsity ranging from 92% to 96% and activation sparsity from 98.4% to 99.6%, LaRa MeDyate exhibits a distinctive pattern of trading lower weight sparsity for higher activation sparsity. This behavior reflects the selection of channels with higher weight-to-activation memory ratios within the LaRa-selected layers. Conversely, TraDy with the previous layer policy demonstrates the opposite tendency, favoring channels with lower weight-to-activation memory ratios.

Regarding computational efficiency, the analysis presents the percentage of weight derivative FLOPs saved through channel freezing during gradient computation. Methods utilizing the previous layer policy achieve higher FLOP savings, typically one to two percentage points more saved FLOPs compared to their LaRa-based



(a) Weight sparsity evolution during training. (b) Activation sparsity evolution during training. (c) Computational savings in weight derivative FLOPs.

Figure 5.10: Efficiency metrics comparison across channel selection strategies during MobileNetV2 fine-tuning on Food dataset under memory constraint. Results show evolution of sparsity levels and computational savings throughout training. Credits to Quélennec et al. [2025b].

counterparts. This difference highlights a potential trade-off between accuracy optimization and computational efficiency, which may be relevant in scenarios where computational cost takes precedence over performance gains. These efficiency characteristics remain consistent across experimental conditions, with complete training metrics provided in supplementary materials.

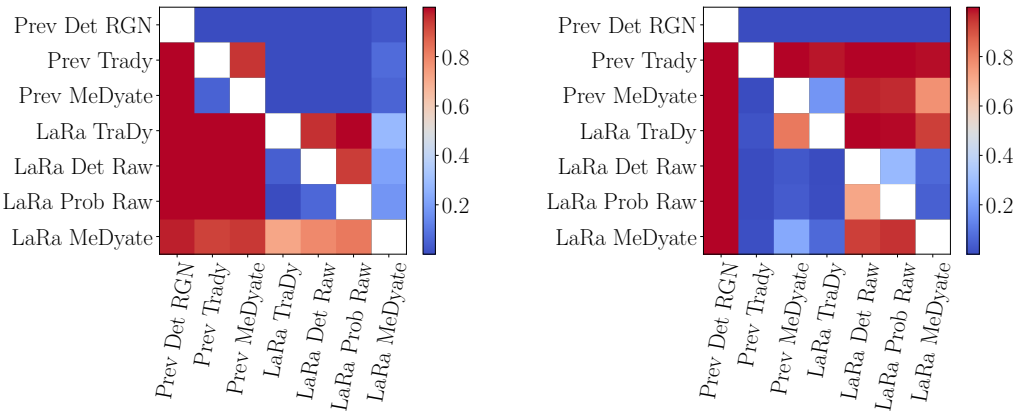
5.3.3 Transformer Results

Following the evaluation framework established in Sec. 4.3.3, we assess the performance of MeDyate across transformer architectures, employing the same comparative methodology used for convolutional networks in Sec. 5.3.2. Fig. 5.11 presents the statistical comparison of final mean test top-1 accuracies through paired T-tests, maintaining consistency with the analysis presented in Fig. 4.10. Full results tables are available in Sec. A.3.3.

The results reveal unexpected patterns that diverge significantly from the convolutional network observations. Across both transformer families, the deterministic channel selection approach using RGN metrics within the fixed layer subset demonstrates superior performance compared to all other evaluated strategies. This finding contrasts with the CNN results where MeDyate consistently achieved the highest performance.

For the SwinT architecture (Fig. 5.11a), the LaRa-based MeDyate approach exhibits the poorest performance among all evaluated strategies. Conversely, its counterpart employing the fixed RGN-ranked layer set with MeDyate channel selection emerges as the second-best strategy and the most effective practically deployable approach.

This performance hierarchy suggests that the LaRa methodology, which proved beneficial for CNNs, appears detrimental when applied to vision transformer architectures. The adaptive layer selection that enhanced CNN performance seem to introduce suboptimal behavior in the context of attention-based vision models.
In the case of BERT architectures (Fig. 5.11b), while the deterministic RGN approach maintains its superiority, the performance gap between strategies narrows considerably. The LaRa ranking methodology demonstrates more competitive results in this NLP context, with both deterministic and stochastic raw gradient norm-based approaches yielding satisfactory performance. This suggests that the LaRa framework may exhibit task-dependent effectiveness, performing adequately for language models while struggling with vision transformers.
The substantial performance discrepancy across architectural families—CNNs, SwinT, and BERTs—highlights fundamental differences in fine-tuning dynamics that warrant deeper investigation. Several hypotheses may explain these observations. First, the attention mechanism's global connectivity patterns in transformers may alter the gradient flow characteristics that informed our LaRa design, potentially invalidating assumptions derived from CNN architectures where gradient information propagates through local convolutions. Second, the architectural differences between vision and language transformers suggest that task-specific features influence the effectiveness of layer ranking strategies. Vision transformers process spatial hierarchies through attention mechanisms, while language models handle sequential dependencies, potentially requiring distinct approaches to layer importance characterization.
These results underscore the need for enhanced understanding of transfer learning dynamics in transformer architectures, with particular attention to the distinct characteristics of vision and NLP tasks. Future work should investigate whether modifications to the LaRa metric or adaptive layer selection criteria can better accommodate the unique properties of attention-based architectures, potentially through attention-specific importance measures or task-aware layer ranking strategies.



(a) SwinT model

(b) BERTs models

Figure 5.11: T-test comparisons of average final test accuracies across multiple experimental dimensions for each group of transformer architectures. Credits to Quélennec et al. [2025b].

5.4 Limitations, Conclusions and Future Works

This chapter presented MeDyate, a comprehensive extension of the TRaDy framework that addresses key limitations in memory-constrained dynamic subnetwork selection. Our main contributions include the LaRa metric for holistic layer characterization with budget-adaptive selection, the exploitation of channel importance temporal stability for probabilistic sampling strategies, and the transition to raw gradient norms within pre-selected layers for computational efficiency. Experimental validation on convolutional architectures demonstrates that MeDyate consistently outperforms existing approaches, often exceeding theoretical upper bounds while maintaining strict memory constraints.

However, our evaluation reveals significant limitations in the generalizability of these findings. The transformer architecture experiments presented in Sec. 5.3.3 demonstrate that MeDyate's effectiveness varies substantially across architectural families. While the LaRa methodology proves beneficial for CNNs, it exhibits suboptimal performance on vision transformers (SwinT) and shows task-dependent effectiveness on language models (BERTs). These discrepancies suggest that the assumptions underlying our layer ranking and channel selection strategies may not extend uniformly to attention-based architectures, highlighting the need for architecture-specific or task-aware adaptations of the framework.

Furthermore, a critical limitation of this work lies in the absence of actual on-device implementation and deployment validation. While our algorithmic innovations demonstrate promise in controlled experimental settings, the practical feasibility of

dynamic channel reselection between epochs on resource-constrained edge devices remains unvalidated. The computational overhead of gradient norm computation and probabilistic sampling, though theoretically manageable, has not been evaluated under real-world hardware constraints. Without empirical evidence from on-device deployment, we cannot definitively assess the energy consumption, latency characteristics, or memory access patterns that would determine the practical viability of MeDyate in production edge AI systems.

While subnetwork selection effectively addresses which parameters to update, the memory bottleneck in backpropagation extends beyond parameter selection to activation storage. The next part of this thesis explores an orthogonal approach that directly targets this activation memory bottleneck through tensor decomposition-based compression techniques, offering a complementary perspective on memory-efficient deep learning.

Part II

Activation Map Compression

Chapter 6

Tensor Decomposition for Activation Compression

"Why Not Help One Another On This Lonely Journey?"

Solaire of Astora

Contents

Chapter 6 Abstract

This chapter investigates the application of tensor decomposition techniques to address the memory bottleneck associated with activation storage during neural network training. While the previous part demonstrated effective parameter selection strategies, activation tensors remain a substantial memory burden regardless of which parameters are selected for update. We present a comprehensive approach leveraging Higher-Order Singular Value Decomposition (HOSVD) to compress activation maps while maintaining the ability to perform exact gradient computations. Our theoretical analysis establishes convergence guarantees and error bounds, while experimental validation demonstrates memory reductions exceeding 95% with minimal accuracy degradation. Despite these promising compression capabilities, the computational overhead of HOSVD limits practical deployment, motivating the efficient alternatives developed in Chapter 7.

6.1 Introduction and Setup

The deployment of neural networks on resource-constrained devices demands sophisticated strategies to address memory limitations during training. As established in the previous chapters, careful parameter selection can significantly reduce the memory footprint of trainable components. However, even with optimal parameter selection, the storage requirements for activation tensors during backpropagation remain a fundamental constraint that affects all network components, regardless of their selection status.

6.1.1 Motivations

The memory and computational challenges in neural network training stem from multiple sources, each requiring targeted solutions. Parameter storage and gradient computation represents one dimension of this challenge, addressed through the subnetwork selection strategies developed in Part I. However, activation tensor storage during the forward pass constitutes an orthogonal memory burden that persists independently of parameter selection decisions.

During standard backpropagation, intermediate activation tensors must be retained throughout the forward pass to enable gradient computation during the backward pass as was described in Sec. 3.1.1. For convolutional neural networks processing high-resolution inputs with large batch sizes, these activation tensors can consume memory resources that dwarf parameter storage requirements by orders of magnitude as was described by Cai et al. [2020]. This memory bottleneck becomes

particularly acute in resource-constrained environments where every kilobyte of memory usage must be carefully managed. Despite this prevalence of activation in memory consumption during backpropagation, the activation compression domain remains relatively unexplored compared to weight compression, presenting opportunities for substantial memory efficiency gains.

As was introduced in Sec. 2.4, tensor decomposition techniques offer a principled mathematical foundation for addressing activation compression. Unlike heuristic compression methods that may compromise gradient quality, tensor decomposition provides controllable approximation error with theoretical guarantees. Furthermore, the low-rank structure inherent in many neural network activations suggests that significant compression may be achievable without substantial information loss.

6.1.2 Chapter Organization

This chapter develops and analyzes a tensor decomposition approach for activation compression. Sec. 6.2 establishes the theoretical foundation, beginning with the mathematical formulation of HOSVD and its application to activation tensors (Sec. 6.2.1). We derive in Sec. 6.2.2 the modified backpropagation equations required to compute gradients directly from compressed representations without full reconstruction. Sec. 6.2.3 provides comprehensive performance analysis, covering compression error bounds, memory footprint reduction, and computational complexity implications. We establish theoretical guarantees for convergence and characterize the trade-offs between compression ratio and approximation quality. Sec. 6.3 presents experimental validation across diverse architectures and datasets, demonstrating the practical effectiveness of our approach and its comparison with existing compression techniques. Finally, we conclude with discussion of limitations and future directions, setting the stage for the efficient

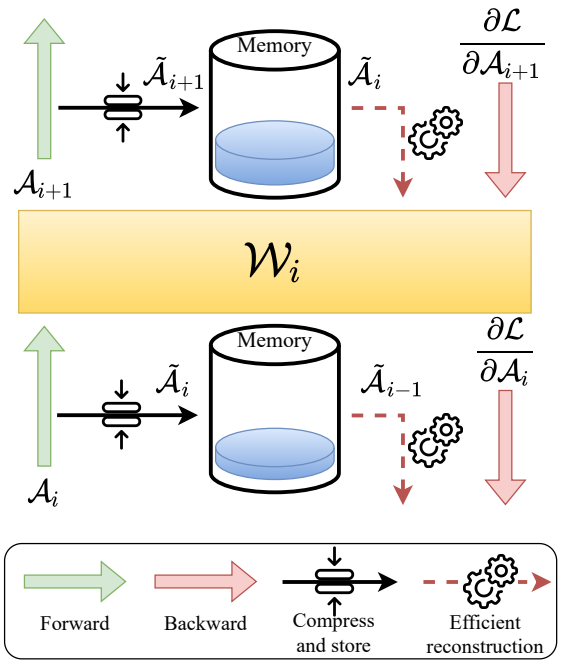


Figure 6.1: Forward and backward pass with activation compression during network training. Credits to Nguyen et al. [2024].

approximation strategies developed in Chapter 7.

6.2 Applying HOSVD Decomposition to Activation Maps

The application of tensor decomposition to activation compression requires careful consideration of both the mathematical properties of the decomposition and the computational requirements of gradient computation. This section develops the theoretical framework for HOSVD-based activation compression and derives the necessary modifications to standard backpropagation algorithms.

6.2.1 The Decomposition Process

We establish here the theoretical framework for tensor decomposition techniques applied to neural network activation compression. Following the computational setup introduced in Sec. 3.1.1, we focus on compressing activation tensors $\mathcal{A}_i \in \mathbb{R}^{B\times C_i\times H_i\times W_i}$.

SVD Baseline. As our baseline approach, we present Singular Value Decomposition (SVD) for activation compression. Given activation tensor $\mathcal{A}_i$, we perform a matricization operation to transform it into a two-dimensional representation $A_i \in \mathbb{R}^{B\times(C_iH_iW_i)}$ and subsequently apply the standard SVD factorization:

$$A_i = U_i\Sigma_i V_i^T, \qquad U_i \in \mathbb{R}^{B\times B}, \quad \Sigma_i \in \mathbb{R}^{B\times(C_iH_iW_i)}, \quad V_i \in \mathbb{R}^{(C_iH_iW_i)\times(C_iH_iW_i)}. \tag{6.1}$$

In this factorization, Σ_i represents a rectangular diagonal matrix containing r_i singular values $s_{k\in[1,r_i]}$ where r_i denotes the rank of matrix A_i. The proportion of total variance captured by the j^{th} singular component can be computed as $\sigma_{i,j}^2 = s_j^2/\sum_k s_k^2$.

Considering that singular values within Σ_i are arranged in decreasing order, $s_l \geq s_j$ for all $l \leq j$, we can establish a compression strategy based on variance retention. For a specified cumulative explained variance threshold $\varepsilon \in [0,1]$, we identify the *compression rank* as the smallest value $J_i \in [1, r_i]$ satisfying $\sum_{j=1}^{J_i}\sigma_{i,j}^2 \geq \varepsilon$. The compressed approximation $\tilde{A}_i$ consists in retaining only the first J_i components from each factor matrix:

$$\tilde{A}_i = U_{i,(J_i)}\Sigma_{i,(J_i)}V_{i,(J_i)}^T, \qquad U_{i,(J_i)} \in \mathbb{R}^{B\times J_i}, \quad \Sigma_{i,(J_i)} \in \mathbb{R}^{J_i\times J_i}, \quad V_{i,(J_i)} \in \mathbb{R}^{(C_iH_iW_i)\times J_i}. \tag{6.2}$$

The compressed representation requires storing the matrix products $U_{i,(J_i)} \times \Sigma_{i,(J_i)}$ and $V_{i,(J_i)}^T$, reducing memory requirements from the original $(\Theta_{\text{space}})_i = BC_iH_iW_i$ elements to $(\Theta_{\text{space}})_i = J_i(B + C_iH_iW_i)$ elements. For linear layer activations,

which naturally exist as two-dimensional matrices, this decomposition process is more straightforward.

A fundamental trade-off emerges between approximation quality and memory efficiency: higher explained variance values yield reconstructions $\tilde{A}_i$ that more closely match the original A_i, potentially improving gradient estimation quality during backpropagation. Conversely, this accuracy improvement necessitates larger compression ranks J_i, thereby increasing memory consumption. Our objective becomes optimizing this balance between variance retention and storage reduction. While SVD has been extensively employed for neural network compression since early pioneering work (Zhang et al. [2015]), traditional applications focused primarily on parameter compression (Xinwei et al. [2023]). Our approach extends this methodology to activation tensor compression, establishing SVD as a baseline against which we evaluate our more advanced tensor decomposition strategy.

HOSVD Generalization. Although SVD provides a well-understood compression mechanism, the reshaping operation required for high dimension tensor decomposition potentially disrupts structural relationships inherent in the tensor format (Xinwei et al. [2023]). Higher-Order Singular Value Decomposition (HOSVD) addresses this limitation by operating directly on tensor structures without dimensional flattening.

For a general n-th order tensor $\mathcal{T} \in \mathbb{R}^{M_1 \times M_2 \times \cdots \times M_n}$, the Tucker decomposition expresses the tensor as:

$$\mathcal{T} = \mathcal{S} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 \cdots \times_n U^{(n)}, \tag{6.3}$$

where $\mathcal{S} \in \mathbb{R}^{L_1 \times L_2 \times \cdots \times L_n}$ represents the core tensor containing compressed information from $\mathcal{T}$, and $U^{(k)} \in \mathbb{R}^{M_k \times L_k}$ are factor matrices whose columns correspond to principal components along the k-th mode. The k-mode product $\times_k$ between tensor $\mathcal{G} \in \mathbb{R}^{P_1 \times P_2 \times \cdots \times P_n}$ and matrix $B \in \mathbb{R}^{Q \times P_k}$ produces tensor $\mathcal{R} \in \mathbb{R}^{P_1 \times \cdots \times P_{k-1} \times Q \times P_{k+1} \times \cdots \times P_n}$ defined by:

$$\mathcal{R}_{p_1,\ldots,p_{k-1},q,p_{k+1},\ldots,p_n} = \mathcal{G} \times_k B = \sum_{p_k=1}^{P_k} g_{p_1,p_2,\ldots,p_n} b_{q,p_k}. \tag{6.4}$$

HOSVD constitutes a specific case of Tucker decomposition where all factor matrices $U^{(k)}$ are mutually orthogonal (Vasilescu and Terzopoulos [2003]). In a similar fashion to our SVD baseline, we apply HOSVD to the four-dimensional activation tensors $\mathcal{A}_i$, yielding:

$$\mathcal{A}_i = \mathcal{S}_i \times_1 U_i^{(1)} \times_2 U_i^{(2)} \times_3 U_i^{(3)} \times_4 U_i^{(4)}, \tag{6.5}$$

where the core tensor $\mathcal{S}_i \in \mathbb{R}^{L_{i,1} \times L_{i,2} \times L_{i,3} \times L_{i,4}}$ and factor matrices $U_i^{(k)} \in \mathbb{R}^{M_{i,k} \times L_{i,k}}$ capture the principal components across batch, channel, height, and width dimensions respectively. Compression occurs through truncation based on explained

variance thresholds. For each mode k, we determine the minimum rank $J_{i,k}$ such that the cumulative explained variance exceeds threshold ε, yielding the compressed representation:

$$\tilde{\mathcal{A}}_i = \hat{\mathcal{S}}_i \times_1 U^{(1)}_{i,J_{i,1}} \times_2 U^{(2)}_{i,J_{i,2}} \times_3 U^{(3)}_{i,J_{i,3}} \times_4 U^{(4)}_{i,J_{i,4}}, \tag{6.6}$$

where $U^{(k)}_{i,J_{i,k}} \in \mathbb{R}^{M_{i,k} \times J_{i,k}}$ contains the first $J_{i,k}$ columns of $U^{(k)}_i$ and $\hat{\mathcal{S}}_i \in \mathbb{R}^{J_{i,1} \times J_{i,2} \times J_{i,3} \times J_{i,4}}$ represents the corresponding truncated core tensor.

6.2.2 Backpropagation with the Decomposed Tensors

One of the key innovations towards enabling practical deployment of activation compression lies in performing gradient computations directly within the compressed representation space. Our approach draws inspiration from prior research (Hameed et al. [2022]; Kim et al. [2015]), which established the computational feasibility of forward operations using decomposed weight tensors while circumventing expensive recomposition procedures. The work of Kim et al. [2015] provides particularly relevant insights through their application of Tucker decomposition to convolutional weight tensors across two modes.

Our methodology extends these principles to the activation compression domain. Their approach is particularly interesting as they apply Tucker decomposition to weight tensors across two modes in convolutional networks. As such, our work constitutes a variation of theirs, since as described in Eq. (3.2), weight derivatives are obtained through a convolution operation between activations and activation derivatives, we decompose along four modes instead of two and HOSVD is a specific case of Tucker.

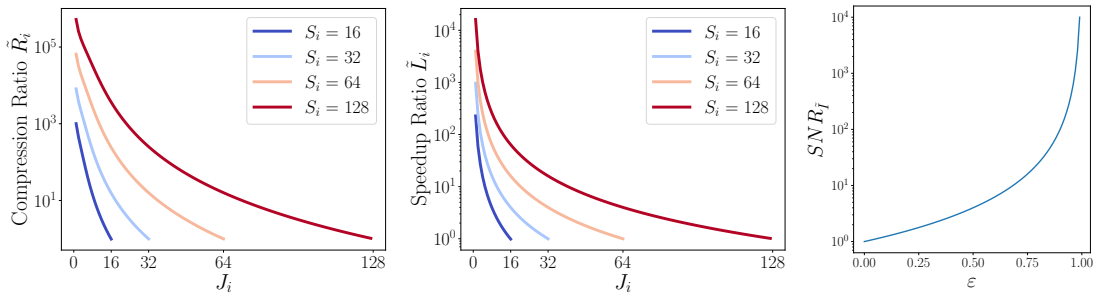
When injecting the decomposed activations $\tilde{\mathcal{A}}_i$ from Eq. (6.6) into the weight derivatives computation, Eq. (3.2) becomes:

$$\frac{\partial \tilde{\mathcal{L}}}{\partial \mathcal{W}_i} = \text{conv}_{1\times1}\left\{\text{conv}_*\left[\text{conv}_{1\times1}\left(\text{conv}_{1\times1}\left(\hat{\mathcal{S}}_i, \underline{U}^{(3)}_{i,J_{i,3}}\right), \underline{U}^{(4)}_{i,J_{i,4}}\right), \text{conv}_{1\times1}\left(\frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}}, U^{(1)}_{i,J_{i,1}}\right)\right], U^{(2)}_{i,J_{i,2}}\right\}, \tag{6.7}$$

where conv_* is a 2D convolution with a specific kernel size as defined in Eq. (A.7), and $\underline{U}^{(k)}_{i,J_{i,k}}$ is the vertically padded version of $U^{(k)}_{i,J_{i,k}}$ as defined in Eq. (A.3). Complete mathematical derivations are provided in Sec. A.1 of the Appendix.

This formulation enables approximated weight gradient computation through the successive computation of lower-dimensional convolution operations, eliminating the need for explicit activation reconstruction. This approach preserves the memory advantages of compression while enabling exact gradient computation up to the approximation error introduced by truncation.

Fig. 6.1 illustrates the application of our method during training. During the forward propagation phase, computations proceed through the standard pathway



(a) (b) (c)

Figure 6.2: **(a)** and **(b)** respectively illustrate the predicted changes in compression rate $\tilde{R}_i$ and latency speedup ratio $\tilde{L}_i$ as a function of J_i, when comparing HOSVD with standard training. **(c)** shows the evolution of the $SNR_{\tilde{I}}$ with retained variance ε.

with activations flowing naturally between layers. However, rather than retaining complete activation tensors in memory for subsequent backward pass requirements, our system stores only the compressed principal components derived from the HOSVD factorization process. When the backward pass commences, these stored principal components are retrieved and utilized in the modified gradient computation sequence described by Eq. (6.7), enabling memory-efficient training.

6.2.3 Performance Analysis

Let us now examine the performance of our proposed method through multiple dimensions: approximation error introduced by compression, memory footprint reduction achieved, and computational implications of the modified backpropagation procedure.

Compression Error. We build our compression error analysis upon signal processing principles applied to the neural network training context. We establish the relationship between explained variance retention and gradient quality through frequency domain analysis.

We adopt here simplified notations where $\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i}$ becomes $\Delta\mathcal{W}$, the activation tensor $\mathcal{A}_i$ is represented as $\mathcal{I}$, and the upstream gradient $\frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}}$ is denoted $\Delta\mathcal{Y}$. Our analysis focuses on characterizing the approximation error introduced when factorizing and truncating activation tensor $\mathcal{I}$ to preserve the components corresponding to explained variance level $\varepsilon \in [0, 1]$.

We establish that the energy content within the computed weight gradients directly corresponds to the energy retained within the compressed activation $\tilde{\mathcal{I}}$. Through frequency domain analysis, we represent the input activation, weight derivative, and output activation derivative as I, ΔW, and ΔY respectively, with $I[u, v]$

denoting the spectral amplitude at frequency coordinates (u, v). The discrete Fourier transform of the compressed activation yields the relationship $\tilde{I} = \varepsilon I$, enabling computation of the signal-to-noise ratio for the compressed representation:

$$SNR_{\tilde{I}} = \frac{\sum_{(u,v)} I[u,v]^2}{\sum_{(u,v)} (I[u,v] - \varepsilon I[u,v])^2} = \frac{1}{(1-\varepsilon)^2}. \tag{6.8}$$

Leveraging the fundamental property that convolution operations in the spatial domain correspond to element-wise multiplication in the frequency domain, the compressed weight gradient becomes $\Delta\tilde{W} = \tilde{I}\Delta Y = \varepsilon I \Delta Y$. Applying the same analytical framework as in Eq. (6.8), we derive $SNR_{\Delta\tilde{W}} = (1-\varepsilon)^{-2} = SNR_{\tilde{I}}$. This relationship, illustrated graphically in Fig. 6.2c, analytically confirms that signal energy transfer from compressed activations to weight gradients scales quadratically with explained variance retention. For instance, preserving $\varepsilon = 0.8$ of the original variance yields $SNR_{\Delta\tilde{W}} = 25$, indicating substantial signal preservation despite significant compression.

A crucial property of our compression approach lies in the absence of error propagation. Since our methodology exclusively compresses activation tensors while leaving weight parameters unmodified, and considering the gradient computation relationships established in Eq. (3.2) and Eq. (3.3), approximation errors affect only the weight gradients at individual layer level. The activation gradients that propagate between network layers are computed exactly through uncompressed pathways, as these calculations depend solely on uncompressed weight tensors and activation derivatives. This error isolation property ensures that compression-induced approximations do not accumulate across network depth, maintaining training stability regardless of model complexity.

Memory Footprint. The memory footprint analysis establishes a quantitative comparison between storage requirements for compressed representations and their uncompressed counterparts. Standard activation storage necessitates $(\Theta_{\text{space}})_i = BC_iH_iW_i$ elements for each activation tensor $\mathcal{A}_i$. In contrast, HOSVD compression stores only the truncated core tensor $\hat{\mathcal{S}}_i \in \mathbb{R}^{J_{i,1} \times J_{i,2} \times J_{i,3} \times J_{i,4}}$ and the corresponding factor matrices $U^{(k)}_{i,J_{i,k}} \in \mathbb{R}^{M_{i,k} \times J_{i,k}}$ for $k = 1, 2, 3, 4$, resulting in a total storage requirement of $(\Theta_{\text{space}})_i = J_{i,1}J_{i,2}J_{i,3}J_{i,4} + BJ_{i,1} + C_iJ_{i,2} + H_iJ_{i,3} + W_iJ_{i,4}$. The compression ratio for layer i therefore becomes:

$$R_i = \frac{BC_iH_iW_i}{J_{i,1}J_{i,2}J_{i,3}J_{i,4} + BJ_{i,1} + C_iJ_{i,2} + H_iJ_{i,3} + W_iJ_{i,4}}. \tag{6.9}$$

To estimate the order of magnitude of achievable compression ratios, we consider a simplified scenario where $B = C_i = H_i = W_i$ and $J_{i,1} = J_{i,2} = J_{i,3} = J_{i,4}$, denoted respectively as S_i and J_i. Under these assumptions, the compression ratio simplifies

to:

$$\tilde{R}_i = \frac{S_i^4}{J_i^4 + 4S_iJ_i}. \tag{6.10}$$

Fig. 6.2a illustrates the variation of $\tilde{R}_i$ as a function of J_i for representative values of S_i. The memory efficiency demonstrates particularly pronounced benefits for large spatial dimensions, where the quadratic scaling of the compression ratio yield substantial storage reductions. Empirical validation in Sec. 6.3 reveals that high explained variance thresholds remain achievable even with modest truncation values J_i, thereby enabling extreme compression rates while maintaining minimal performance degradation.

Computational Complexity. We analyze the computational implications of the two primary modifications introduced by HOSVD activation decomposition relative to standard training procedures. The forward pass incurs substantial additional computational overhead due to the tensor decomposition operations required for compression. Given that SVD computation for a matrix of dimensions $m \times n$ (with $m \geq n$) exhibits complexity $\Theta_{\text{FLOPs}} = m^2n$, and considering that HOSVD performs SVD operations across each tensor mode, the computational burden becomes significant. For a tensor with dimensions $B \times C_i \times H_i \times W_i$, HOSVD necessitates SVD computations on matrices of sizes $B \times C_iH_iW_i$, $C_i \times BH_iW_i$, $H_i \times BC_iW_i$, and $W_i \times BC_iH_i$. Consequently, the computational complexity for the decomposition phase during forward propagation becomes:

$$\begin{aligned}(\Theta_{\text{FLOPs}})_i[\text{HOSVD}] = \; & \max(B, C_iH_iW_i)^2 \times \min(B, C_iH_iW_i) + \max(C_i, BH_iW_i)^2 \times \min(C_i, BH_iW_i) + \\ & \max(H_i, BC_iW_i)^2 \times \min(H_i, BC_iW_i) + \max(W_i, BC_iH_i)^2 \times \min(W_i, BC_iH_i).\end{aligned} \tag{6.11}$$

For comparative analysis, the computational complexity of standard forward propagation is characterized by:

$$(\Theta_{\text{FLOPs}})_i[\text{Forward}] = D_i^2C_iC_i'BH_iW_i. \tag{6.12}$$

Applying the same dimensional simplification used in Eq. (6.10), where all dimensions equal S_i, the complexities become:

$$(\mathcal{O}_{\text{time}})_i[\text{HOSVD}] = 4S_i^7, \tag{6.13}$$

$$(\mathcal{O}_{\text{time}})_i[\text{Forward}] = S_i^7. \tag{6.14}$$

This analysis reveals that incorporating HOSVD activation decomposition effectively quintuples the computational cost, equivalent to performing four additional forward passes. Such overhead clearly presents unrealistic latency and energy consumption characteristics for practical on-device deployment scenarios.

Conversely, the backward pass exhibits computational benefits through reduced complexity operations on compressed representations. The modified gradient

computation procedure outlined in Eq. 6.7 employs a sequence of lower-dimensional operations rather than full convolutions on original tensor dimensions.
Standard weight derivative computation $\Delta\mathcal{W}$ requires complexity $(\Theta_{\text{FLOPs}})[\Delta\mathcal{W}_\rangle] = D^2 C_i C_i' B H_i' W_i'$, whereas the intermediate term computations detailed in Eq. A.8 of the appendix exhibit the following complexities:

$$\begin{aligned}
(\Theta_{\text{FLOPs}})[Z_i^{(1)}] &= J_{i,1} C_i' H_i' W_i' B, \\
(\Theta_{\text{FLOPs}})[Z_i^{(2)}] &= J_{i,1} J_{i,2} H_i J_{i,4} J_{i,3}, \\
(\Theta_{\text{FLOPs}})[Z_i^{(3)}] &= J_{i,1} J_{i,2} H_i W_i J_{i,4}, \\
(\Theta_{\text{FLOPs}})[Z_i^{(4)}] &= C_i' J_{i,2} D_i^2 H_i' W_i' J_{i,1}, \\
(\Theta_{\text{FLOPs}})[\Delta\tilde{\mathcal{W}}_\rangle] &= C_i' C_i D_i^2 J_{i,2}.
\end{aligned}$$

Following the dimensional simplification approach used for compression ratio analysis, we assume uniform truncation thresholds J_i and spatial dimensions S_i. This yields the latency ratio comparing standard to accelerated weight derivative computation L_i:

$$\tilde{L}_i = \frac{S_i^7}{J_i^4 S_i + J_i^3 S_i^2 + J_i^2 S_i^5 + 2 J_i S_i^4} = \frac{S_i^6}{J_i (J_i^3 + J_i^2 S_i + J_i S_i^4 + 2 S_i^3)} \tag{6.15}$$

Fig. 6.2b showcases the variation of $\tilde{L}_i$ as a function of J_i for representative values of S_i. The magnitude of achievable speedup ratios exhibits trends similar to compression ratios, illustrating the potential for our approach to achieve dramatic reductions in both memory consumption and computational energy requirements. Nevertheless, the forward pass computational overhead typically dominates overall training costs, resulting in net latency increases despite substantial backward pass acceleration. This fundamental limitation provides the primary motivation for the efficient approximation strategies developed in Chapter 7.

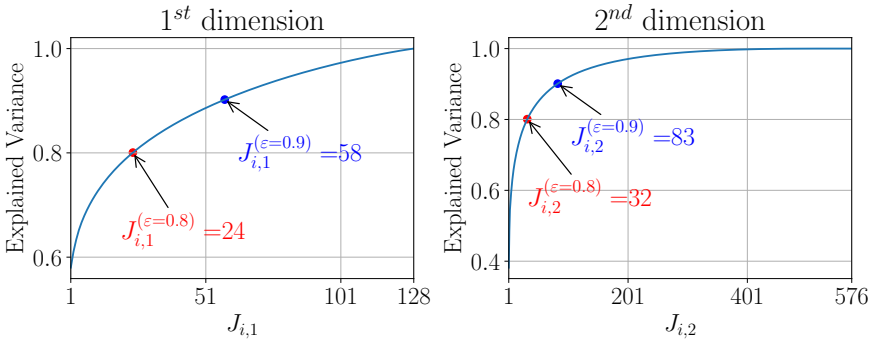


Figure 6.3: Explained variance ε for the first two dimensions of the activation map in the 4^{th} last layer ($i = 39$) when fine-tuning the last four layers of MCUNet using HOSVD on CIFAR-10, following setup A. Credits to Nguyen et al. [2024].

6.3 Experimental Results

This section presents comprehensive experimental validation of our HOSVD-based activation compression approach. First, we establish the setup upon which we base our experiments (Sec. 6.3.1). We then analyze the distribution of energy across different tensor dimensions in HOSVD, providing empirical validation for typical truncation values J_i (Sec. 6.3.2) followed by an extensive performance evaluation across multiple architectures and datasets to characterize the accuracy-memory trade-off landscape (Sec. 6.3.3). Finally, we provide in Sec. 6.3.4 a visualization of latency comparison between HOSVD and standard training. All experiments were conducted using an NVIDIA RTX 3090Ti with PyTorch 1.13.1 as the underlying framework.

6.3.1 Preamble

Our experimental validation encompasses diverse computer vision scenarios across two fundamental tasks: classification and semantic segmentation. This breadth ensures robust evaluation of compression effectiveness across different computational patterns and memory usage profiles. Further details about our experimental setup and hyperparameters are available in Sec. A.2.2 of the Appendix.

Classification Experiments. We investigate two distinct fine-tuning methodologies that reflect different practical deployment scenarios:

- *Full fine-tuning* (designated as *setup A*): Following established transfer learning practices, we initialize models with ImageNet pre-trained weights and perform complete fine-tuning on various downstream classification datasets

including CIFAR-10, CIFAR-100, CUB, Flowers, and Pets. This configuration is mostly similar to one explored in Part I.

- *Half fine-tuning* (designated as *setup B*): Adopting the methodology established by Yang et al. [2023b], we partition each classification dataset (ImageNet, CIFAR-10/100) into two non-independent and identically distributed segments of equal size using the FedAvg McMahan et al. [2017] partitioning strategy. Each partition undergoes an 80%-20% train-validation split. The first partition serves for pre-training, while the second enables fine-tuning evaluation under more constrained data availability conditions.

Semantic Segmentation Experiments. Mirroring the constrained fine-tuning approach, we reproduce the segmentation framework from Yang et al. [2023b]. Models undergo initial training on Cityscapes Cordts et al. [2016] using MMSegmentation Contributors [2020] infrastructure, followed by fine-tuning adaptation on Pascal-VOC12 Chen et al. [2017]. This configuration evaluates compression effectiveness for dense prediction tasks with different spatial resolution and computational requirements.

Memory Measurement Methodology. A critical aspect of our evaluation concerns accurate memory consumption quantification for HOSVD and SVD approaches. Unlike methods that explicitly control compression through rank specification, our approach regulates compression via retained information thresholds, preventing direct memory usage control of principal components. Consequently, all reported results include both peak and average consumption values accompanied by their respective standard deviations, ensuring transparent and reproducible evaluation of compression effectiveness.

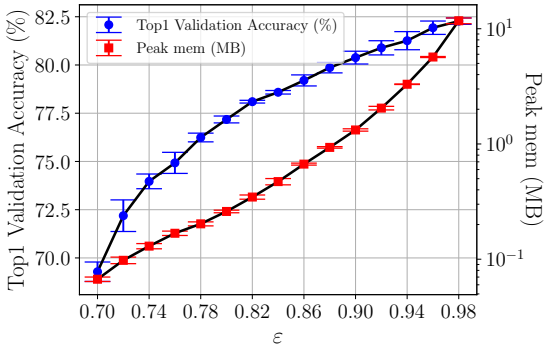


Figure 6.4: Behavior of top1 validation accuracy and peak memory when applying HOSVD with different explained variance thresholds ε and finetuning the last four convolutional layers of an MCUNet model using the CIFAR-10 dataset on setup A. Credits to Nguyen et al. [2024].

6.3.2 Explained Variance Evolution

We conduct targeted experiments to validate the theoretical predictions regarding energy distribution within tensor modes. Using MCUNet Lin et al. [2020] with its

last four layers fine-tuned on CIFAR-10 and decomposed using HOSVD, following Setup A configuration.
Fig. 6.3 illustrates the explained variance evolution across two tensor dimensions k as a function of retained principal components $J_{i,k}$, where $k \in \{1, 2\}$ corresponds to the two largest activation map dimensions (batch size and input channels respectively). We define $J_{i,k}^{(\varepsilon=x)}$ as the minimum number of principal components necessary to retain at least fraction x of the explained variance.
The results validate our theoretical hypothesis from Sec. 6.2.3: fewer than 20% of components capture more than 80% of explained variance across both dimensions. This concentration of information in leading components confirms the effectiveness of low-rank approximation for activation compression. Furthermore, the curves exhibit logarithmic behavior, indicating that high explained variance levels remain achievable with minimal compression ratio degradation. This characteristic proves particularly significant given that SNR transmitted to weight derivatives scales quadratically with explained variance (as demonstrated in Fig. 6.2c). Additional layer-wise analysis appears in Sec. A.2.2.
Fig. 6.4 presents the relationship between different explained variance thresholds ε and resulting performance metrics, averaged across three random seed configurations. For $\varepsilon < 0.8$, increasing variance retention yields substantial accuracy improvements coupled with impressive compression ratios. However, beyond $\varepsilon = 0.8$, accuracy gains diminish while computational costs increase. Above the $\varepsilon = 0.9$ threshold, exponential peak memory growth creates unfavorable accuracy-compression trade-offs. Based on this analysis, subsequent experiments employ ε values of 0.8 and 0.9 as optimal operating points.

6.3.3 Main Results

MCUNet Classification with Setup A. Our primary classification evaluation employs MCUNet pre-trained on ImageNet and fine-tuned on CIFAR-10, systematically varying the number of trained layers from the closest to the output to complete network fine-tuning (1 to 42). We compare standard training, Yang et al. [2023b]'s gradient filtering with patch sizes 2, 4, and 7, SVD, and HOSVD with explained variance threshold 0.8 across different fine-tuning depths.
Fig. 6.5 presents performance curves where the X-axis represents activation memory consumption in kilobytes (logarithmic scale) and the Y-axis indicates peak validation accuracy. Individual markers denote the number of fine-tuned convolutional layers, with progression from minimal (rightmost) to complete network fine-tuning (leftmost). Optimal methods exhibit curves trending toward the upper-left region, indicating superior accuracy with minimal memory consumption.
The results reveal distinct behavioral patterns across compression methods. Gra-

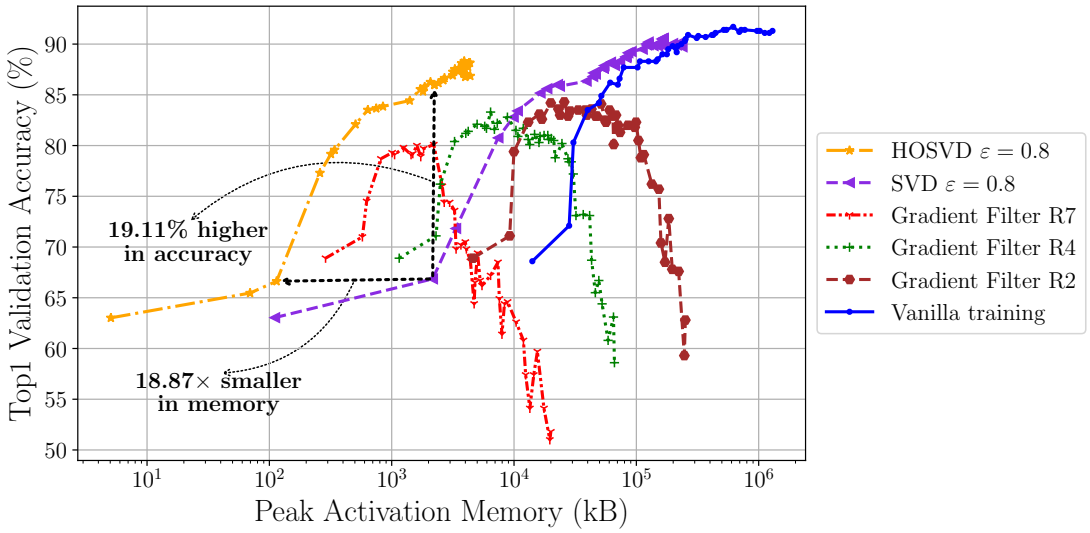


Figure 6.5: Performance curves of an MCUNet pre-trained on ImageNet and finetuned on CIFAR-10 with different activation compression strategies. Credits to Nguyen et al. [2024].

dient filtering accuracy initially improves with additional layers but subsequently deteriorates, suggesting error propagation through the network during training. Conversely, SVD, HOSVD, and standard training maintain consistent accuracy improvement with increased depth, following similar trends. This distinction aligns with our theoretical analysis in Sec. 6.2.3: HOSVD localizes compression errors to individual layers, preventing network-wide degradation propagation.
Comparing equivalent depths, SVD consistently achieves superior accuracy over HOSVD. We attribute this phenomenon to HOSVD performing SVD across all tensor modes, potentially introducing information loss across multiple dimensions, whereas SVD concentrates information loss within a single mode through matricization.
Despite this accuracy trade-off, HOSVD demonstrates substantial memory advantages over SVD, achieving up to 18.87× memory reduction at equivalent accuracy levels. Conversely, for identical memory budgets, HOSVD yields up to 19.11% accuracy improvement over SVD. When compared to gradient filtering and vanilla training at equivalent memory consumption, HOSVD consistently delivers significantly superior accuracy. Remarkably, complete network fine-tuning with HOSVD requires substantially less activation memory than single-layer vanilla training, confirming theoretical compression predictions from Fig. 6.2a. Complete experimental results for Setup A appear in Table A.9 within Sec. A.3.4 of the Appendix.
ImageNet Classification with Setup B. Table 6.1 presents classification perfor-

Table 6.1: Experimental results on ImageNet-1k. "#Layers" refers to the number of fine-tuned convolutional layers (counted from the end of the model). Activation memory consumption is shown in MegaBytes (MB). Credits to Nguyen et al. [2024].

Method	MobileNetV2				Method	ResNet18			
	#Layers	Acc ↑	Peak Mem (MB) ↓	Mean Mem (MB) ↓		#Layers	Acc ↑	Peak Mem (MB) ↓	Mean Mem (MB) ↓
Vanilla training	All	74.0	1651.84	1651.84 ± 0.00	Vanilla training	All	72.8	532.88	532.88 ± 0.00
	2	62.6	15.31	15.31 ± 0.00		2	69.9	12.25	12.25 ± 0.00
	4	65.8	28.71	28.71 ± 0.00		4	71.5	30.63	30.63 ± 0.00
Gradient Filter R2	2	62.6	5.00	5.00 ± 0.00	Gradient Filter R2	2	68.7	4.00	4.00 ± 0.00
	4	65.2	9.38	9.38 ± 0.00		4	69.3	7.00	7.00 ± 0.00
SVD ($\varepsilon = 0.8$)	2	61.7	4.97	4.92 ± 0.08	SVD ($\varepsilon = 0.8$)	2	69.5	7.88	7.71 ± 0.21
	4	65.2	14.76	14.61 ± 0.09		4	71.1	19.98	19.72 ± 0.28
SVD ($\varepsilon = 0.9$)	2	62.3	8.97	8.91 ± 0.08	SVD ($\varepsilon = 0.9$)	2	69.7	9.86	9.77 ± 0.13
	4	65.5	20.35	20.20 ± 0.07		4	71.3	24.81	24.66 ± 0.17
HOSVD ($\varepsilon = 0.8$)	2	61.1	0.15	0.15 ± 0.00	HOSVD ($\varepsilon = 0.8$)	2	69.2	0.97	0.91 ± 0.05
	4	63.9	0.73	0.68 ± 0.03		4	70.5	2.89	2.74 ± 0.12
HOSVD ($\varepsilon = 0.9$)	2	61.8	0.43	0.43 ± 0.01	HOSVD ($\varepsilon = 0.9$)	2	69.5	2.73	2.63 ± 0.10
	4	64.8	1.92	1.76 ± 0.08		4	71.1	7.96	7.66 ± 0.21
Method	**MCUNet**				**Method**	**ResNet34**			
	#Layers	Acc ↑	Peak Mem (MB) ↓	Mean Mem (MB) ↓		#Layers	Acc ↑	Peak Mem (MB) ↓	Mean Mem (MB) ↓
Vanilla training	All	67.4	632.98	632.98 ± 0.00	Vanilla training	All	75.6	839.04	839.04 ± 0.00
	2	62.1	13.78	13.78 ± 0.00		2	69.6	12.25	12.25 ± 0.00
	4	64.7	19.52	19.52 ± 0.00		4	72.2	24.50	24.50 ± 0.00
Gradient Filter R2	2	61.8	4.50	4.50 ± 0.00	Gradient Filter R2	2	68.8	4.00	4.00 ± 0.00
	4	64.4	6.38	6.38 ± 0.00		4	70.9	8.00	8.00 ± 0.00
SVD ($\varepsilon = 0.8$)	2	62.0	7.62	7.51 ± 0.12	SVD ($\varepsilon = 0.8$)	2	69.2	6.70	6.49 ± 0.29
	4	64.5	10.59	10.37 ± 0.20		4	71.8	14.68	14.24 ± 0.50
SVD ($\varepsilon = 0.9$)	2	62.1	10.32	10.26 ± 0.08	SVD ($\varepsilon = 0.9$)	2	69.4	9.10	8.96 ± 0.23
	4	64.6	14.39	14.26 ± 0.13		4	72.0	19.11	18.83 ± 0.37
HOSVD ($\varepsilon = 0.8$)	2	61.7	0.48	0.43 ± 0.04	HOSVD ($\varepsilon = 0.8$)	2	68.7	0.30	0.27 ± 0.02
	4	63.9	0.88	0.78 ± 0.07		4	71.1	1.11	1.02 ± 0.07
HOSVD ($\varepsilon = 0.9$)	2	62.0	1.32	1.27 ± 0.06	HOSVD ($\varepsilon = 0.9$)	2	69.2	0.71	0.65 ± 0.05
	4	64.4	2.52	2.36 ± 0.15		4	71.9	3.24	3.09 ± 0.13

mance and memory consumption across MobileNetV2, ResNet18, and ResNet34 He et al. [2016] architectures using various compression methods including standard training, gradient filtering, HOSVD, and SVD on ImageNet. The results demonstrate that SVD and HOSVD achieve competitive performance with gradient filtering across most configurations while substantially reducing activation memory consumption through HOSVD compression.

Segmentation Performance. Table 6.2 reports segmentation performance and memory consumption across multiple architectures following the Yang et al. [2023b] evaluation framework. The results consistently demonstrate that increasing explained variance from 0.8 to 0.9yields substantial performance improvements with minimal memory overhead. This behavior further validates our theoretical prediction that most explained variance concentrates within leading principal components, enabling effective generalization with aggressive compression rates.

6.3.4 Latency Analysis

In this section we conduct empirical latency analysis comparing computational overhead against standard training. The experimental configuration employs MCUNet

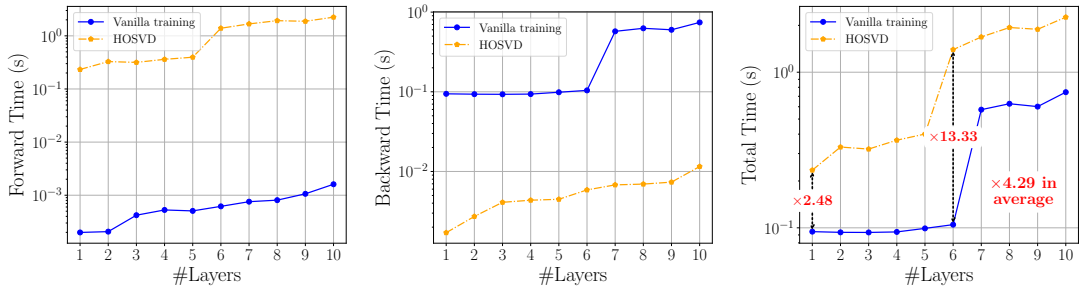
Table 6.2: Experimental results for semantic segmentation. mIoU is the mean Intersection over Union, and mAcc is the micro averaged accuracy. Credits to Nguyen et al. [2024].

Method	PSPNet Zhao et al. [2017]					Method	PSPNet-M Zhao et al. [2017]				
	#Layers	mIoU ↑	mAcc ↑	Peak Mem (MB) ↓	Mean Mem (MB) ↓		#Layers	mIoU ↑	mAcc ↑	Peak Mem (MB) ↓	Mean Mem (MB) ↓
Vanilla training	All	54.97	68.46	920.78	920.78 ± 0.00	Vanilla training	All	48.92	62.11	2622.49	2622.49 ± 0.00
	5	39.36	51.79	128.00	128.00 ± 0.00		5	36.22	46.31	104.00	104.00 ± 0.00
	10	53.17	67.18	352.00	352.00 ± 0.00		10	45.62	58.35	604.00	604.00 ± 0.00
Gradient Filter	5	39.34	51.59	8.00	8.00 ± 0.00	Gradient Filter	5	35.73	45.78	6.50	6.50 ± 0.00
	10	51.2	65.01	22.00	22.00 ± 0.00		10	44.89	57.38	37.75	37.75 ± 0.00
SVD ($\varepsilon = 0.8$)	5	38.97	50.04	90.40	85.64 ± 2.80	SVD ($\varepsilon = 0.8$)	5	34.98	44.48	58.50	57.17 ± 0.77
	10	52.03	65.44	238.20	232.50 ± 4.71		10	44.73	56.83	377.68	369.24 ± 6.93
SVD ($\varepsilon = 0.9$)	5	39.34	50.92	111.20	108.88 ± 1.43	SVD ($\varepsilon = 0.9$)	5	35.51	45.51	78.65	78.65 ± 0.00
	10	52.7	66.39	295.60	292.46 ± 2.50		10	45.79	58.97	482.00	475.17 ± 4.79
HOSVD ($\varepsilon = 0.8$)	5	38.11	49.29	0.47	0.32 ± 0.08	HOSVD ($\varepsilon = 0.8$)	5	33.40	42.50	0.03	0.03 ± 0.00
	10	49.23	62.39	1.40	1.24 ± 0.08		10	40.06	51.79	1.47	1.30 ± 0.06
HOSVD ($\varepsilon = 0.9$)	5	39.03	50.29	2.26	1.70 ± 0.30	HOSVD ($\varepsilon = 0.9$)	5	34.09	43.69	0.07	0.07 ± 0.00
	10	52.11	65.5	8.18	6.44 ± 0.49		10	44	56.9	8.20	6.16 ± 0.62
Method	**DLV3 Chen et al. [2017]**					**Method**	**DLV3-M Chen et al. [2017]**				
	#Layers	mIoU ↑	mAcc ↑	Peak Mem (MB) ↓	Mean Mem (MB) ↓		#Layers	mIoU ↑	mAcc ↑	Peak Mem (MB) ↓	Mean Mem (MB) ↓
Vanilla training	All	58.44	72.03	1128.02	1128.02 ± 0.00	Vanilla training	All	55.87	69.47	2758.01	2758.01 ± 0.00
	5	40.75	52.95	336.00	336.00 ± 0.00		5	38.38	49.61	240.00	240.00 ± 0.00
	10	55.04	69.07	560.00	560.00 ± 0.00		10	47.91	61.67	620.00	620.00 ± 0.00
Gradient Filter	5	32.18	42.93	27.47	27.47 ± 0.00	Gradient Filter	5	35.7	46.71	20.71	20.71 ± 0.00
	10	47.44	60.08	83.47	83.47 ± 0.00		10	45.4	58.97	65.62	65.62 ± 0.00
SVD ($\varepsilon = 0.8$)	5	39.75	51.69	242.30	240.91 ± 0.77	SVD ($\varepsilon = 0.8$)	5	36.78	47.15	169.00	166.87 ± 1.26
	10	52.69	66.2	381.00	376.81 ± 3.02		10	46.22	57.82	374.13	366.76 ± 5.66
SVD ($\varepsilon = 0.9$)	5	40.47	52.09	291.40	291.40 ± 0.00	SVD ($\varepsilon = 0.9$)	5	37.59	48.24	204.50	200.69 ± 1.80
	10	53.86	67.61	472.70	466.52 ± 4.71		10	47.4	59.4	493.88	485.56 ± 5.77
HOSVD ($\varepsilon = 0.8$)	5	38.52	50.14	2.66	2.62 ± 0.01	HOSVD ($\varepsilon = 0.8$)	5	35.55	45.64	0.63	0.55 ± 0.03
	10	50.11	63.14	1.93	1.48 ± 0.14		10	42.68	54.17	1.04	0.96 ± 0.05
HOSVD ($\varepsilon = 0.9$)	5	40.19	52.3	12.64	12.35 ± 0.10	HOSVD ($\varepsilon = 0.9$)	5	37.06	47.73	4.56	4.16 ± 0.15
	10	52.26	65.8	9.23	7.24 ± 0.63		10	45.75	57.61	5.52	4.79 ± 0.29
Method	**FCN Long et al. [2015]**					**Method**	**UPerNet Xiao et al. [2018]**				
	#Layers	mIoU ↑	mAcc ↑	Peak Mem (MB) ↓	Mean Mem (MB) ↓		#Layers	mIoU ↑	mAcc ↑	Peak Mem (MB) ↓	Mean Mem (MB) ↓
Vanilla training	All	45.36	59.53	952.00	952.00 ± 0.00	Vanilla training	All	64.71	77.32	2168.78	2168.78 ± 0.00
	5	27.31	38.21	288.00	288.00 ± 0.00		5	48.05	61.66	1380.00	1380.00 ± 0.00
	10	43.54	57.96	480.00	480.00 ± 0.00		10	48.9	63.1	1436.00	1436.00 ± 0.00
Gradient Filter	5	27.24	38.1	18.00	18.00 ± 0.00	Gradient Filter	5	46.79	60.5	33.00	33.00 ± 0.00
	10	36.91	50.14	120.00	120.00 ± 0.00		10	47.89	62.44	36.50	36.50 ± 0.00
SVD ($\varepsilon = 0.8$)	5	28.62	38.42	196.30	191.41 ± 2.12	SVD ($\varepsilon = 0.8$)	5	46.56	59.46	804.55	784.80 ± 17.28
	10	34.60	45.71	316.70	288.37 ± 13.78		10	47.70	61.02	840.70	829.13 ± 5.39
SVD ($\varepsilon = 0.9$)	5	31.92	42.35	251.50	245.40 ± 1.74	SVD ($\varepsilon = 0.9$)	5	47.22	59.77	1078.38	1056.52 ± 14.42
	10	42.04	54.53	406.20	396.14 ± 4.89		10	48.3	61.16	1126.50	1113.98 ± 11.49
HOSVD ($\varepsilon = 0.8$)	5	26.34	36.14	1.43	1.26 ± 0.08	HOSVD ($\varepsilon = 0.8$)	5	45.28	57.80	1.35	1.32 ± 0.02
	10	30.91	41.19	3.77	2.54 ± 0.72		10	46.44	58.93	1.68	1.63 ± 0.03
HOSVD ($\varepsilon = 0.9$)	5	30.46	41.12	8.10	6.99 ± 0.38	HOSVD ($\varepsilon = 0.9$)	5	46.8	59.29	4.72	4.59 ± 0.10
	10	39.89	52.11	17.57	14.28 ± 1.55		10	47.94	60.7	7.81	7.51 ± 0.12

architecture processing a single CIFAR-10 batch under Setup A conditions, with timing measurements collected over one complete training epoch to isolate the compression-specific computational costs.

Fig. 6.6 presents detailed timing comparisons across forward propagation, backward propagation, and total training duration between vanilla training and HOSVD compression. The results reveal a characteristic trade-off pattern that validates our theoretical predictions from Sec. 6.2.3.

The backward propagation phase demonstrates substantial computational benefits for HOSVD, requiring approximately one to two orders of magnitude less execution time compared to standard training. This acceleration stems from the reduced-dimensional operations enabled by compressed representations, as detailed in our



(a) (b) (c)

Figure 6.6: **(a)**, **(b)**, and **(c)** represent the time in seconds taken by the algorithm to respectively perform forward pass, backward pass, and the total training process, when fine-tuning an MCUNet on one batch of CIFAR-10 with Setup A across 1 to 10 layers. Credits to Nguyen et al. [2024].

complexity analysis (Sec. 6.2.3). However, the forward propagation phase exhibits the opposite behavior, with HOSVD incurring computational overhead reaching three orders of magnitude above vanilla training costs. This dramatic increase results directly from the tensor decomposition operations required for activation compression at each layer.

When aggregating forward and backward timing contributions, HOSVD demonstrates an average slowdown factor of 4.29× relative to standard training. This net performance degradation aligns precisely with our theoretical expectations established in Sec. 6.2.3, where we identified the forward pass decomposition overhead as the primary computational bottleneck limiting practical deployment.

These empirical timing results confirm that while HOSVD achieves remarkable memory compression capabilities, the associated computational overhead presents significant challenges for real-time training scenarios. The substantial forward pass costs effectively overshadow the backward pass acceleration benefits, creating unfavorable overall latency characteristics. This fundamental limitation provides the primary motivation for developing the more computationally efficient approximation strategies presented in Chapter 7, which aim to preserve the memory advantages of tensor decomposition while addressing the computational constraints that limit practical applicability.

6.4 Limitations and Conclusions

This chapter established tensor decomposition as a mathematically principled approach to addressing the activation memory bottleneck in neural network training. Through comprehensive theoretical analysis and empirical validation, we

demonstrated that HOSVD-based compression achieves substantial memory reductions—exceeding 95% in optimal scenarios—while maintaining competitive model accuracy across diverse architectures and tasks.

Our theoretical contributions encompass several key advances. We derived modified backpropagation equations that enable gradient computation directly from compressed representations, eliminating memory-intensive activation reconstruction. The compression error analysis revealed that approximation errors remain localized to individual layers without network-wide propagation, ensuring training stability. Furthermore, we established that gradient signal-to-noise ratio scales quadratically with retained explained variance, providing quantitative guidance for compression parameter selection.

Experimental validation confirmed the practical effectiveness of our approach. HOSVD consistently outperformed existing activation compression methods, achieving superior accuracy-memory trade-offs compared to gradient filtering techniques. Results validated our theoretical predictions regarding energy concentration in leading principal components, with fewer than 20% of components capturing over 80% of explained variance.

However, significant limitations constrain practical deployment. The computational overhead during forward propagation presents the most critical constraint. Our latency analysis revealed that HOSVD incurs an average 4.29× slowdown relative to standard training, with forward pass costs reaching three orders of magnitude above baseline requirements. This overhead stems from performing complete SVD decomposition across all tensor modes at each training iteration.

Additionally, our approach requires manual tuning of explained variance thresholds for different architectures and datasets. While theoretical analysis provides general guidance, practical deployment benefits from empirical validation to determine optimal compression parameters, potentially limiting applicability in scenarios where extensive hyperparameter exploration is infeasible.

Despite these constraints, the theoretical framework establishes tensor decomposition as a promising foundation for activation compression research. The demonstrated memory efficiency, combined with error localization properties, provides clear advantages over heuristic compression approaches and creates opportunities for training larger models within fixed memory budgets.

The computational limitations directly motivate the efficient approximation strategies developed in Chapter 7. The fundamental trade-off between memory efficiency and computational overhead suggests that practical deployment requires methods preserving compression benefits while reducing per-iteration costs. Chapter 7 addresses these challenges through the Activation Subspace Iteration method, leveraging temporal stability of activation patterns during fine-tuning to precompute decomposition components and significantly reduce computational overhead.

Chapter 7

Solving the Latency Issue with ASI

"Sometimes the best way to solve your own problems is to help someone else"

Uncle Iroh

Contents

Chapter 7 Abstract

The High-Order Singular Value Decomposition (HOSVD) approach presented in Chapter 6 demonstrates significant potential for memory-efficient on-device learning through activation tensor compression. However, the substantial computational overhead of tensor decomposition limits real-world applicability along with the lack of control over the memory footprint. This chapter addresses these limitations by introducing Activation Subspace Iteration (ASI), a novel method that leverages the temporal stability of activation patterns during training to achieve comparable compression ratios while dramatically reducing computational costs. The key insight is that activation maps evolve gradually between consecutive training iterations, enabling effective reuse of previously computed decomposition components. ASI employs a single subspace iteration step with warm-start initialization, replacing the computationally expensive HOSVD decomposition while maintaining memory efficiency. Through perplexity-based rank selection adapted specifically for activation compression, ASI provides explicit memory budget control unlike variance threshold approaches. Experimental validation demonstrates that ASI achieves up to 120.09× memory reduction and 1.86× training speedup compared to standard training, while delivering 91.0× faster execution than HOSVD on embedded devices. These results establish ASI as a practical solution for enabling on-device learning with strict computational and memory constraints.

7.1 HOSVD Framework Extension

7.1.1 HOSVD Limitations

While the HOSVD-based approach developed in Chapter 6 demonstrates substantial memory compression capabilities, two fundamental limitations restrict its applicability for on-device learning scenarios.

The first limitation concerns the computational overhead introduced by tensor decomposition operations. Performing HOSVD decomposition at each training iteration requires computing singular value decompositions across all four modes of each activation tensor $\mathcal{A}_i$ and for each training step, resulting in a steep computational overhead (Eq. (6.14)). This overhead becomes particularly problematic for larger activation maps common in modern architectures, where the decomposition cost can significantly exceed the computational savings achieved during backpropagation. Real-world latency measurements reveal that HOSVD can increase forward pass latency by over 100× compared to vanilla training, making it impractical for

resource-constrained environments.
The second limitation relates to memory budget control. The explained variance threshold approach employed by HOSVD provides theoretical guarantees about information preservation but offers limited control over actual memory consumption. Different layers and training phases can result in varying compression ratios for the same variance threshold, making it difficult to maintain consistent memory usage within strict hardware constraints. This variability particularly affects deployment scenarios where precise memory budget adherence is critical for system stability. Additionally, the dynamic nature of HOSVD's resource consumption complicates system resource planning. Unlike methods with fixed computational and memory profiles, HOSVD exhibits peak resource usage that varies throughout training, requiring systems to provision for worst-case scenarios rather than average consumption patterns.

7.1.2 Introducing ASI

ASI addresses the computational limitations of HOSVD by exploiting the temporal stability properties of activation maps during neural network fine-tuning. The method builds upon the key observation that activation maps change gradually between consecutive training iterations due to the incremental nature of gradient-based optimization and the Lipschitz continuity of activation functions.
Our method introduces three core components that enable efficient activation compression. First, a perplexity-based rank selection strategy adapted specifically for activation compression scenarios provides explicit memory budget control. Unlike variance thresholds that offer indirect memory management, this approach directly optimizes rank combinations to satisfy hardware constraints while minimizing approximation error in gradient computations.
Second, the subspace iteration technique proposed by Stewart and Miller [1975] replaces expensive SVD operations with a single iteration step that approximates the dominant subspace directions. This approach leverages mathematical properties from distributed optimization literature, where similar techniques demonstrate effectiveness for gradient compression in communication-limited settings.
Third, the warm-start mechanism reuses subspace approximations from previous iterations as initialization for current decompositions. This reuse strategy exploits the smooth evolution of activation patterns during training, reducing approximation variance and improving convergence properties compared to independent decompositions at each step.
The combination of these components enables ASI to achieve compression ratios comparable to HOSVD while dramatically reducing computational overhead. The method maintains the fundamental advantages of tensor decomposition approaches,

including error localization properties and theoretical memory efficiency guarantees, while addressing the practical deployment challenges that limit HOSVD's applicability.

7.1.3 Chapter Organization

The remainder of this chapter provides comprehensive exposition of the ASI method and its experimental validation. Sec. 7.2 introduces the theoretical foundations of subspace iteration for activation compression and motivates the need for specialized rank selection strategies. The algorithmic details of the subspace iteration process adaptation is detailed in Sec. 7.2.1, followed by a presentation of our perplexity search techniques for activation scenarios in Sec. 7.2.2. Sec. 7.2.3 presents the complete ASI algorithm with computational complexity analysis.
Experimental validation is presented in Sec. 7.3 with exploratory experiments in Sec. 7.3.1 investigating the effects of warm-start mechanisms, perplexity variations, and latency characteristics. Main results in Sec. 7.3.2 demonstrate ASI's performance across diverse architectures and datasets. The chapter concludes with a discussion of future research directions and potential extensions of the ASI framework.

7.2 Activation Subspace Iteration (ASI) Method

The core innovation of ASI lies in replacing the computationally expensive HOSVD decomposition with a more efficient approximation strategy based on subspace iteration. This approach leverages the observation that exact tensor decomposition may be unnecessarily precise for the gradient computation requirements of backpropagation, where approximate low-rank representations can maintain training effectiveness while dramatically reducing computational costs.
Subspace iteration provides a natural solution for computing dominant subspace directions without full SVD computation. The technique performs iterative refinement of subspace approximations through matrix-vector multiplications, offering computational complexity that scales more favorably with target rank dimensions than full decomposition methods. For activation compression, this approach is particularly well-suited since the first few singular values typically capture most tensor energy, making low-rank approximations both effective and sufficient for accurate gradient computation.
However, one of the key challenge of efficient subspace iteration is the truncation rank pre-selection. As demonstrated by Zhang et al. [2023], selecting a single rank for all layers and across all dimensions does not account for the different contributions to model's performance. This motivates the development of a rank se-

lection strategy that optimally balances approximation quality with computational efficiency under explicit memory constraints.

7.2.1 Subspace Iteration

The main source of inspiration for ASI builds upon Vogels et al. [2019] subspace iteration for distributed gradient compression by, which in turn adapts the classical subspace iteration method from Stewart and Miller [1975]. The subspace iteration framework operates by approximating a matrix $M \in \mathbb{R}^{m \times n}$ through its low-rank factorization $\tilde{M} = PQ^T$, where $P \in \mathbb{R}^{m \times r}$ and $Q \in \mathbb{R}^{n \times r}$ represent the factor matrices and $r \in [1, \min(m, n)]$ denotes the target truncation rank. This decomposition strategy achieves computational advantages by focusing exclusively on the most significant subspace directions rather than computing the complete spectral decomposition. The iterative refinement process employs matrix-vector multiplications to progressively improve subspace approximations, resulting in computational complexity that scales favorably with the target rank dimension compared to traditional full decomposition approaches.
Vogels et al. [2019] observe that single-step subspace iteration alone provides insufficient approximation accuracy for practical applications. The limitation stems from the random initialization required at each iteration, which fails to capture the underlying subspace structure effectively within a single computational step. To address this challenge, they introduce a warm-start mechanism that reuses the low-rank approximation computed in the previous iteration as initialization for the current step. This strategy leverages the demonstrated temporal stability of gradient subspaces across consecutive epochs, enabling single-step approximations to achieve accuracy levels comparable to multi-step iterations while maintaining computational efficiency.
Algorithm 3 presents the subspace iteration method with warm-start initiallization. It requires performing two matrix multiplications $P^{(t)} = M^{(t)}Q^{(t)}$, $Q^{(t)} = M^{(t)^T}P^{(t)}$ and an orthogonalization using Gram-Schmidt, respectively corresponding to $2mnr$ and r^3 operations. Therefore, the total computational complexity of the subspace iteration algorithm is:

$$(\Theta_{\text{FLOPs}})[\text{SI}] = 2mnr + r^3. \tag{7.1}$$

The adaptation of subspace iteration to activation tensor compression presents a natural extension of these principles, where activation maps replace gradient matrices as the target for low-rank approximation. The four-dimensional structure of activation tensors necessitates applying the subspace iteration procedure independently to each tensor mode, effectively treating each unfolded matrix representation as a candidate for compression. This mode-wise decomposition preserves the essential structure of activation tensors while enabling significant memory reduction

Algorithm 3: Subspace Iteration with Warm-start at epoch t. Credits to Nguyen et al. [2025].

Require: Epoch index t, gradient matrix $M^{(t)} \in \mathbb{R}^{m \times n}$, target rank $r \in [1, \min(m, n)]$.

if $t = 0$ **then**
  Initialize $Q^{(t)} \in \mathbb{R}^{n \times r}$ from an i.i.d standard normal distribution.
else
  Warm-start initialization, $Q^{(t)} \leftarrow Q^{(t-1)}$.
$P^{(t)} \leftarrow M^{(t)} Q^{(t)}$.
$\hat{P}^{(t)} \leftarrow \text{Orthogonalize}(P^{(t)})$.
$Q^{(t)} \leftarrow {M^{(t)}}^T \hat{P}^{(t)}$.
Return $\hat{P}^{(t)}, Q^{(t)}$.

through rank truncation.
However, the successful application of subspace iteration to activation compression requires careful consideration of rank selection strategies across tensor modes. The exponential growth in possible rank combinations across multiple tensor modes and network layers necessitates sophisticated optimization procedures to identify configurations that minimize compression-induced errors while satisfying hardware constraints. This challenge motivates the development of an efficient rank selection methodology specifically adapted for activation tensor decomposition scenarios.

7.2.2 Perplexity Search and Rank Selection

The tensor decomposition methodologies explored in Chapter 6 employed explained variance thresholds as the primary mechanism for determining appropriate truncation ranks. However, this approach presents fundamental limitations for subspace iteration scenarios where ranks must be predetermined before computation begins. Furthermore, direct rank specification enables precise control over memory consumption targets, addressing the memory budget limitations identified in HOSVD deployments.
To overcome these challenges, we adapt to the domain of activation tensor decomposition Yuan et al. [2023]'s rank selection techniques originally developed for parameter compression. Their Sensitivity-based Truncation Rank Searching (STRS) determines optimal compression levels through perplexity analysis. The core insight involves deriving a layer sensitivity metric using perplexity $\mathcal{P} = e^{\mathcal{L}}$, where $\mathcal{L}$ represents the model loss, to quantify approximation errors when processing previously unseen data.
Traditional perplexity evaluation operates through layer-wise analysis, system-

atically varying compression intensity to identify configurations that maximize compression while minimizing perplexity degradation. The methodology involves evaluating models across diverse compression ratios, defined as the relationship between compressed and original tensor representations. Their analysis reveals inverse proportionality between perplexity changes and compression ratio modifications, with each layer exhibiting distinct sensitivity characteristics. Layers demonstrating substantial perplexity increases under reduced compression ratios are classified as highly sensitive and require more conservative compression strategies.

Activation Perplexity Framework. Our adaptation reconceptualizes perplexity evaluation for activation compression contexts. Unlike parameter compression where perplexity directly reflects model output degradation, activation compression primarily influences gradient computation accuracy during backpropagation rather than affecting model predictions directly. The compressed activations $\tilde{\mathcal{A}}_i$ serve as intermediate representations that impact parameter gradient calculations $\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i}$ without immediate consequence for forward pass outputs.

To address the exponential complexity of exhaustively evaluating all possible rank combinations across tensor modes, we substitute compression ratios with a discretized set of explained variance thresholds $\mathcal{E} \in (0, 1]^E$. This formulation provides a tractable framework for systematic exploration, where E represents the number of threshold values under consideration. Each threshold generates a specific rank configuration across all layers, creating a manageable search space for optimization procedures.

The activation perplexity estimation methodology operates through a two-phase approach:

- For each explained variance threshold $\varepsilon_e \in \mathcal{E}$, we execute forward propagation using representative data samples. During this process, layer i generates both the original activation tensor $\mathcal{A}_i$ and its compressed counterpart $(\tilde{\mathcal{A}}_i)_e$. The compressed version results from applying HOSVD decomposition followed by truncation according to threshold ε_e, as detailed in Sec. 6.2.1. This procedure yields a corresponding set of truncation ranks denoted as $J_i^{(e)}$ for each layer-threshold combination.

- The subsequent backward propagation phase computes both original gradients $\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i}$ and their approximated versions $\left(\widetilde{\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i}}\right)_e$ derived from compressed activations. We define the activation perplexity $\mathcal{P}_{i,e}$ for layer i under threshold ε_e as the Frobenius norm measuring the discrepancy between these gradient computations:

$$\mathcal{P}_{i,e} = \left\| \frac{\partial \mathcal{L}}{\partial \mathcal{W}_i} - \left(\widetilde{\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i}} \right)_e \right\|_F . \tag{7.2}$$

Algorithm 4: ASI for layer i with preselected set of target ranks R_{opt_i}. Credits to Nguyen et al. [2025].

Require: Activation map $\mathcal{A}_i^{(t)} \in \mathbb{R}^{B\times C_i\times H_i\times W_i}$ at epoch t, set of target ranks R_{opt_i}.

Initialize $\mathcal{S}_i = \mathcal{A}_i^{(t)}$.;
for $k = 1$ ***to*** 4 **do**
  $A_{i,k} \leftarrow \text{Unfold}(\mathcal{A}_i^{(t)})$ along mode k.;
  if $t = 0$ **then**
    Initialize $V_{i,k} \in \mathbb{R}^{S_{i,k}\times J_{i,k}}$ from an i.i.d standard normal distribution.;
  else
    $V_{i,k} \leftarrow A_{i,k}^T U_{i,k}^{(t-1)}$.;
  $U_{i,k}^{(t)} \leftarrow \text{Orthogonalize}\,(A_{i,k}V_{i,k})$.;
  $\mathcal{S}_i \leftarrow \mathcal{S}_i \times_k U_{i,k}^{(t)}$.;
Return $\mathcal{S}_i, U_{i,k}^{(t)}$ with $k : 1 \to 4$.;

Systematic application across all network layers produces a comprehensive perplexity matrix $\mathcal{P} \in \mathbb{R}^{N\times E}$ alongside a corresponding rank tensor $\mathcal{R}^{N\times E\times 4}$. These data structures encode the complete relationship between layers, explained variance thresholds, and resulting rank selections across all four tensor modes.

Memory-Constrained Rank Optimization. The rank selection procedure operates under explicit memory budget constraints $B_{\text{mem}}^{\mathcal{A}}$ applied to the cumulative activation memory across fine-tuned layers $\mathcal{F}$. This formulation enables precise resource allocation while optimizing compression effectiveness within hardware limitations.

Building upon the memory analysis presented in Section 6.2.3, the activation memory requirements for HOSVD compression follow:

$$M_i(J_i) = \prod_{k=1}^{4} J_{i,k} + \sum_{k=1}^{4} S_{i,k}J_{i,k}, \tag{7.3}$$

where $S_{i,k}$ and $J_{i,k}$ respectively denote spatial dimensions and truncation ranks across the four tensor modes k of activation $\mathcal{A}_i$, with $J_i = \{J_{i,k}\},\ k \in [1, 4]$.

The optimal rank determination employs a recursive backtracking algorithm designed to identify the configuration that minimizes perplexity while satisfying memory constraints. Let $\mathcal{J} = [1, |\mathcal{F}|] \times [1, E]$ represent the complete set of layer-threshold pairs. The objective involves discovering optimal explained variance

threshold assignments $\mathcal{J}_{\text{opt}} \in [1, E]^{|\mathcal{F}|}$ such that:

$$R_{\text{opt}_i} = \mathcal{R}_{i,e} \mid e = \mathcal{J}_{\text{opt}}[i], \tag{7.4}$$

$$\mathcal{J}_{\text{opt}} = \arg\min_{\mathcal{J}} \left(\sum_{i,e \in \mathcal{J}} \mathcal{P}_{i,e} \right) \Bigg| \sum_{i=1}^{|\mathcal{F}|} M_i(J_i^{(e)}) \leq B_{\text{mem}}^{\mathcal{A}}. \tag{7.5}$$

This optimization framework ensures that the selected rank configuration minimizes gradient computation errors while strictly adhering to memory budget requirements. The backtracking approach systematically explores the solution space, pruning infeasible configurations early to maintain computational efficiency during the search process.

7.2.3 The ASI Algorithm

Unified Framework Implementation. Our ASI methodology integrates the rank selection strategy, subspace iteration technique, and warm-start initialization introduced in previous sections into a cohesive algorithmic framework designed for practical activation compression deployment. Algorithm 4 provides implementation details for processing layer i with subspace iteration, using the optimal rank configuration $R_{\text{opt}_i} = \{J_{i,k}\}_{k:1\to4}$ determined following the approach presented in Sec. 7.2.2.

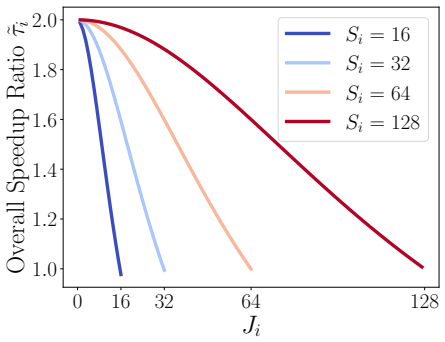
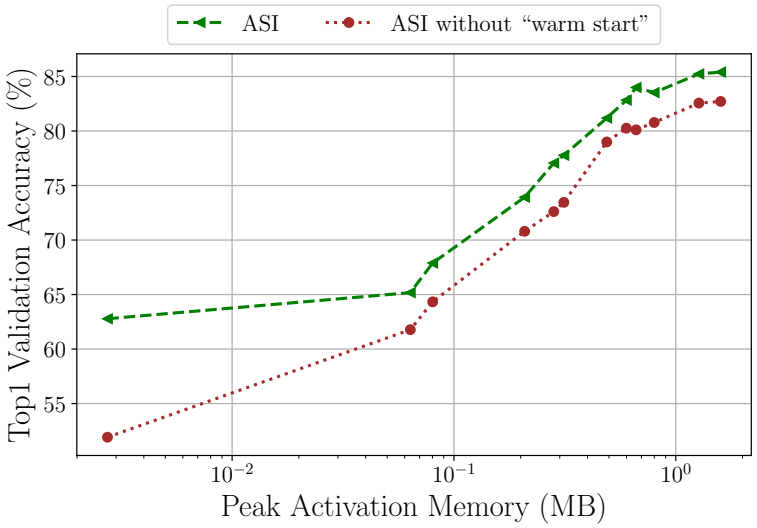


Figure 7.1: Expected variations in overall speedup ratio $\tilde{\tau}_i$ as a function of J_i, when comparing ASI with standard training for typical values of S_i.

The algorithmic workflow initiates by establishing the compressed tensor $\mathcal{S}_i$ as a copy of the original activation map, subsequently applying mode-wise subspace iteration across all four tensor dimensions. The initialization strategy varies depending on the training epoch: during the initial iteration ($t = 0$), subspace matrices receive random normal initialization to establish baseline subspace directions without prior information. For all subsequent epochs, the warm-start protocol activates, computing initial subspace approximations as $V_{i,k} = A_{i,k}^T U_{i,k}^{(t-1)}$ to leverage temporal continuity from previous iterations.

Orthogonalization is performed through the Gram-Schmidt process, ensuring that subspace matrices preserve orthonormal characteristics. The iterative refinement proceeds through mode-k tensor contractions $\mathcal{S}_i = \mathcal{S}_i \times_k U_{i,k}^{(t)}$, systematically reducing tensor dimensionality while preserving essential structural information across all spatial dimensions.



t

Figure 7.2: Performance comparison of ASI with and without warm-start for an MCUNet pre-trained on ImageNet and finetuned CIFAR-10. Credits to Nguyen et al. [2025].

Temporal Stability. The effectiveness of warm-start initialization relies fundamentally on the temporal stability characteristics exhibited by activation patterns during neural network training. This principle extends naturally from observations in distributed optimization contexts, where Vogels et al. [2019] demonstrate that gradient features undergo only marginal modifications between consecutive training steps. Consequently, properly configured projection subspaces remain approximately stable across sufficiently small temporal windows, enabling effective reuse of previously computed decomposition components.

The mathematical foundation for this stability stems from the linear properties of subspace projection operators, which ensure that sequences of optimization steps within consistent subspace configurations converge toward eigenvector representations of temporally averaged stochastic gradients. This convergence behavior produces lower approximation variance compared to approaches that independently recalculate projection subspaces at each iteration, thereby improving both computational efficiency and numerical stability.

Extending this framework to activation tensor scenarios reveals analogous stability properties governing the evolution of intermediate network representations. Layer i computes its activation map through the functional relationship:

$$\mathcal{A}_i = \sigma\left[\text{conv}\left(\mathcal{W}_{i-1}, \mathcal{A}_{i-1}\right)\right], \tag{7.6}$$

where $\sigma(\cdot)$ represents the nonlinear activation function applied element-wise to the convolution output. The incremental nature of gradient-based parameter updates,

as established in Sec. 5.2.2 through our analysis of input channel gradient stability, ensures that weight matrices $\mathcal{W}_{i-1}$ experience only modest modifications between consecutive training iterations. These bounded weight changes propagate through the network architecture, resulting in correspondingly limited variations in activation tensors $\mathcal{A}_i$ across temporal sequences.

The temporal stability assumption receives additional theoretical support from Lipschitz continuity analysis of common activation functions. Research by Virmaux and Scaman [2018] establishes that widely employed nonlinearities, including ReLU, Leaky ReLU, SoftPlus, Tanh, Sigmoid, ArcTan, and Softsign, along with max-pooling operations, exhibit Lipschitz constants equal to one. This property guarantees that bounded input perturbations produce proportionally bounded output variations, providing formal justification for the gradual evolution of activation patterns during training and supporting the mathematical validity of warm-start approximation strategies.

Computational Complexity Analysis. The ASI algorithm executes subspace iteration procedures independently across all four modes of activation tensor $\mathcal{A}_i$, with each mode contributing computational overhead according to the single-iteration cost established in Eq (7.1). The aggregate forward pass complexity accumulates contributions from all tensor modes, yielding:

$$(\Theta_{\text{FLOPs}})[\text{ASI}_i] = \sum_{k=1}^{4} J_{i,k} \left(J_{i,k}^2 + 2S_{i,k} \prod_{l=1, l\neq k}^{4} S_{i,l} \right). \tag{7.7}$$

Following the analytical simplifications introduced in Sec. 6.2.3, we adopt uniform representations S_i and J_i for spatial dimensions and truncation ranks respectively, enabling a more tractable complexity expression:

$$(\mathcal{O}_{\text{FLOPs}})[\text{ASI}_i] = 4(J_i^3 + 2J_i S_i^4). \tag{7.8}$$

This formulation reveals significant computational advantages compared to HOSVD decomposition, which exhibits complexity scaling exclusively with spatial dimensions raised to the seventh power. The ASI approach demonstrates more favorable scaling properties, with complexity depending on spatial dimensions to the fourth power combined with truncation rank terms that typically remain substantially smaller than spatial dimensions. This improved scaling characteristic translates directly into reduced latency overhead during practical deployment scenarios.

The low-rank tensor representations generated through ASI maintain functional equivalence to those produced by HOSVD during backpropagation, enabling direct application of the speedup analysis framework developed for tensor decomposition approaches. The overall training speedup ratio compares computational costs between standard and modified forward and backward passes, with the simplified

speedup metric $\tilde{\tau}_i$ expressed as:

$$\tilde{\tau}_i = \frac{2S_i^7}{S_i^7 + J_i^4 S_i + J_i^3 S_i^2 + J_i^2 S_i^5 + 2J_i S_i^4 + 4(J_i^3 + 2J_i S_i^4)} \tag{7.9}$$

Fig. 7.1 illustrates the relationship between speedup ratio $\tilde{\tau}_i$ and truncation rank J_i across representative spatial dimension values. The analysis reveals that aggressive compression configurations can achieve up to twofold training acceleration, while conservative compression settings ensure that ASI never introduces latency penalties compared to standard training.

Memory Storage Requirements. The auxiliary memory requirements for maintaining subspace matrices $U_{i,k}^{(t)}$ across training iterations remain computationally manageable, demanding $(\Theta_{\text{space}})[U_i^{(t)}] = \sum_k S_{i,k} J_{i,k}$ additional storage capacity per network layer. This supplementary overhead represents a minor fraction of total activation tensor storage demands, especially considering the substantial compression ratios attained through the tensor decomposition framework. The favorable memory scaling properties ensure that subspace matrix storage does not compromise the primary memory efficiency objectives that motivate activation compression strategies.

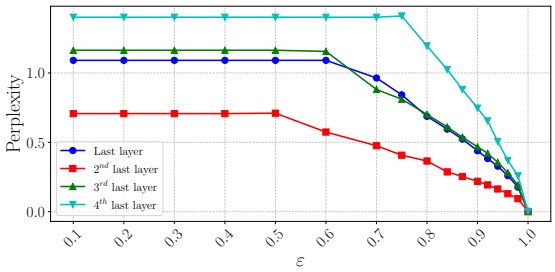


Figure 7.3: Perplexity as a function of the explained variance threshold ε for the last layers of MCUNet. Credits to Nguyen et al. [2025].

7.3 Experimental Results

The experimental evaluation of ASI follows the framework established in Chapter 6 while incorporating additional analysis specific to subspace iteration, including warm-start ablation study, perplexity variation analysis, and comprehensive latency measurements on a Raspberry Pi 5.

The primary experimental difference involves replacing explained variance thresholds with explicit memory budget constraints. To ensure fair comparison, we set ASI's memory budget to match HOSVD$_\varepsilon$'s peak memory consumption when fine-tuning equivalent layer depths. Results include FLOPs and memory usage for ASI and standard training, while HOSVD$_\varepsilon$ reports peak values due to its dynamic resource consumption.

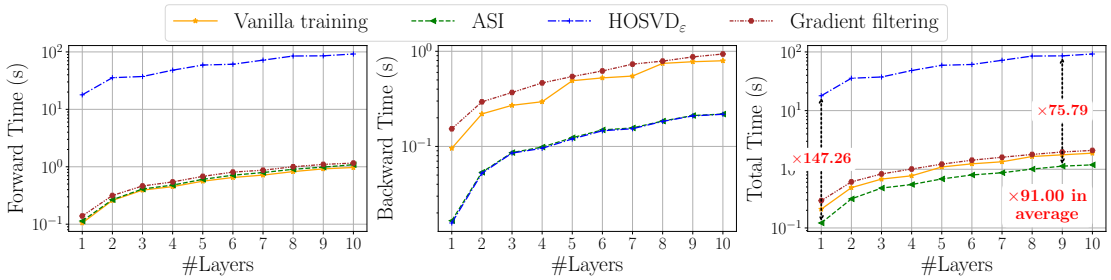


Figure 7.4: Training time of MCUNet over 5 iterations on CIFAR-10 with batch size 128 on a Raspberry Pi 5. Credits to Nguyen et al. [2025].

The perplexity-based rank selection employs explained variance thresholds $\mathcal{E} = \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ to measure layer perplexity $\mathcal{P}_{i,e}$ as defined in Equation (7.2), requiring initial calibration experiments to establish threshold-rank relationships for each architecture.

7.3.1 Exploratory Experiments

Warm-Start. We begin by investigation the warm-start effects. Fig. 7.2 represents the comparison of ASI performance with and without warm-start initialization when fine-tuning MCUNet on CIFAR-10, with the model pre-trained on ImageNet-1k. Our evaluation systematically increases the number of fine-tuned layers from the final layer upward, measuring accuracy-memory trade-offs across different fine-tuning depths. Results demonstrate that even without warm-start initialization, single subspace iteration steps achieve convergent training, validating the fundamental approximation approach. However, incorporating warm-start mechanisms improves accuracy by an average of 3.87% across tested configurations, confirming the benefit of temporal stability exploitation for enhanced approximation quality.

Perplexity Variations. The perplexity behavior across different explained variance thresholds ε for MCUNet's final four layers is depicted in Fig. 7.3. The analysis reveals the expected inverse relationship between explained variance thresholds and resulting perplexity values, where increased ε consistently produces reduced compression errors. Notably, the perplexity curves exhibit saturation behavior for thresholds below 0.5, with minimal variation observed in this regime.

This saturation phenomenon reflects the energy distribution characteristics within activation tensors, where dominant singular values capture larger amounts of total tensor energy. As evidenced in Fig. 6.3, the leading singular value alone accounts for over 50% of the activation map energy in typical configurations. Consequently, explained variance thresholds below this concentration level fail to yield meaningful rank reductions, as the first singular component already satisfies the variance preser-

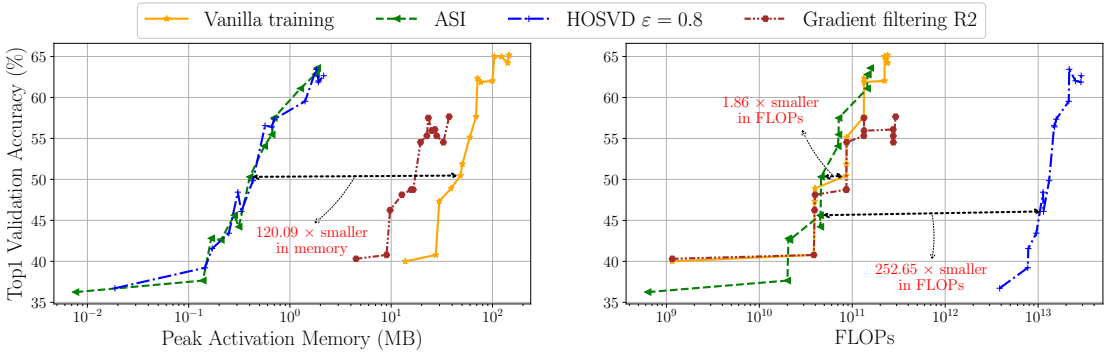


Figure 7.5: Performance of MCUNet pre-trained on ImageNet and finetuned on Pets with different strategies. Credits to Nguyen et al. [2025].

vation criterion. Based on this observation, we restrict perplexity evaluation to $\varepsilon > 0.4$ across all experimental configurations to optimize computational efficiency without compromising analysis completeness.

Latency Analysis. Fig. 7.4 provides critical validation of ASI's practical deployment advantages with respect to latency. Experiments conducted on Raspberry Pi 5 hardware measure training time across the first five iterations of fine-tuning MCUNet on CIFAR-10 with batch size 128. Forward pass measurements reveal that HOSVD incurs 106.13× longer execution time compared to both ASI and vanilla training, confirming the prohibitive computational overhead of full tensor decomposition. Backward pass analysis demonstrates that low-rank computation enables both ASI and HOSVD to achieve 3.95× speedup compared to vanilla training. The combination of minimal forward pass overhead and substantial backward pass acceleration results in ASI achieving 91.0× speedup compared to HOSVD, 1.86× speedup compared to gradient filtering approaches, and 1.56× speedup compared to vanilla training.

7.3.2 Main Results

We conduct a similar experiment as the one presented in Fig. 6.5 but this time fine-tuning MCUNet on Pets and represent the results in Fig. 7.5. We compare performance between ASI, HOSVD with $\varepsilon = 0.8$, and vanilla training approaches. ASI achieves compression ratios comparable to HOSVD while consuming fewer FLOPs than vanilla training for equivalent fine-tuning depths. At similar accuracy levels, ASI reduces memory usage by up to 120.09× compared to vanilla training while requiring 252.65× fewer FLOPs than HOSVD. The overall computational efficiency demonstrates up to 1.86× speedup compared to vanilla training as predicted in Fig. 7.1, establishing ASI's practical advantages for resource-constrained

Table 7.1: Performance evaluation on ImageNet fine-tuning with "#Layers" denoting the number of fine-tuned convolutional layers, counted from the model's end. Activation memory is presented in MegaBytes (MB), and computational complexity is expressed in terms of Gigaflops (GFLOPs). Gradient filtering is applied with patch size R2. Credits to Nguyen et al. [2025].

Method	**MobileNetV2**				**Method**	**ResNet18**			
	#Layers	Acc ↑	Mem (MB) ↓	GFLOPs ↓		#Layers	Acc ↑	Mem (MB) ↓	GFLOPs ↓
Vanilla training	All	74.0	1651.84	830.82	Vanilla training	All	72.8	532.88	291.79
	2	62.6	15.31	4.50		2	69.9	12.25	29.60
	4	65.8	28.71	57.48		4	71.5	30.63	46.45
Gradient filtering R2	2	62.6	5.00	4.50	Gradient filtering R2	2	68.7	4.00	29.60
	4	65.2	9.38	57.50		4	69.3	7.00	47.70
HOSVD ($\varepsilon = 0.8$)	2	61.1	0.15	3049.71	HOSVD ($\varepsilon = 0.8$)	2	69.2	0.97	1581.42
	4	63.9	0.73	5895.31		4	70.5	2.89	4048.56
ASI	2	60.3	0.13	2.81	ASI	2	68.9	0.93	19.04
	4	64.0	0.71	31.54		4	70.6	2.63	33.14
Method	**MCUNet**				**Method**	**ResNet34**			
	#Layers	Acc ↑	Mem (MB) ↓	GFLOPs ↓		#Layers	Acc ↑	Mem (MB) ↓	GFLOPs ↓
Vanilla training	All	67.4	632.98	248.84	Vanilla training	All	75.6	839.04	528.55
	2	62.1	13.78	19.31		2	69.6	12.25	29.60
	4	64.7	19.52	19.88		4	72.2	24.50	59.19
Gradient filtering R2	2	61.8	4.50	19.31	Gradient filtering R2	2	68.8	4.00	29.60
	4	64.4	6.38	19.89		4	70.9	8.00	59.21
HOSVD ($\varepsilon = 0.8$)	2	61.7	0.48	1988.05	HOSVD ($\varepsilon = 0.8$)	2	68.7	0.30	1579.64
	4	63.9	0.88	2457.59		4	71.1	1.11	3160.46
ASI	2	61.7	0.38	11.01	ASI	2	68.3	0.25	16.44
	4	63.5	0.83	11.93		4	71.1	1.09	35.31

environments. A more complete
ImageNet-1k classification experiments across ResNet-18, ResNet-34, and MobileNetV2 architectures confirm ASI's generalization capabilities beyond the MCUNet evaluation focus. Results presented in Tab. 7.1 consistently demonstrate that ASI achieves the lowest resource consumption while maintaining accuracy comparable to HOSVD across different model depths and architectural variations.
The comparison with gradient filtering techniques provides additional context for ASI's performance characteristics. While gradient filtering approaches focus on backward pass acceleration through selective gradient computation, ASI addresses both memory efficiency and computational overhead through activation compression. The combined benefits result in superior overall performance for scenarios where both memory constraints and computational efficiency represent critical requirements.
A more complete set of results is presented in Sec. A.3.5 with results on other downstream tasks, segmentation and even Large Language Models (LLM).

7.4 Limitations, Conclusions and Future Works

This chapter introduced Activation Subspace Iteration (ASI) as an effective solution to the computational limitations of HOSVD-based activation compression. By leveraging temporal stability properties of activation maps during training, ASI achieves comparable memory compression ratios while dramatically reducing computational overhead, making activation compression practical for real-world on-device learning scenarios.

The key innovations of ASI include the adaptation of perplexity-based rank selection for activation compression, providing explicit memory budget control compared to our previous variance threshold approach. The subspace iteration technique with warm-start initialization successfully approximates tensor decomposition with minimal computational cost, while maintaining the theoretical advantages of low-rank approximation methods. Experimental validation demonstrates substantial practical improvements, including up to 120.09× memory reduction, 1.86× training speedup, and 91.0× latency improvement compared to HOSVD on embedded hardware.

Future research directions for ASI include extension to larger language models where activation memory represents an even more critical bottleneck. The adaptation of subspace iteration techniques to transformer architectures with attention mechanisms presents both challenges and opportunities for further computational optimization.

Additional optimization opportunities include exploration of mixed-precision subspace iteration, leveraging lower precision arithmetic for further computational acceleration while preserving numerical stability. The integration of ASI with model pruning and quantization techniques represents another promising direction for comprehensive model compression strategies.

Chapter 8

Conclusions and Future Perspectives

"*It's done*"

Frodo Baggins

Contents

Chapter 8 Abstract

This thesis has addressed fundamental challenges in memory-efficient on-device learning by developing two complementary approaches: strategic subnetwork selection for efficient fine-tuning and activation tensor decomposition for memory-efficient training. These contributions establish a foundation for practical deployment of adaptive neural networks on resource-constrained edge devices.

8.1 Summary of Contributions

Our research has developed theoretical frameworks and practical algorithms that advance the state-of-the-art in memory-constrained neural network training across the following two complementary domains.

Efficient Subnetwork Selection. We established a comprehensive theoretical foundation for dynamic subnetwork selection during fine-tuning. Our analysis of fine-tuning dynamics through heavy-tailed gradient theory revealed predictable patterns that enable principled subnetwork selection decisions. This groundwork culminated in the development of the MeDyate algorithm, which dynamically and adaptively selects optimal subnetworks while respecting strict memory constraints. MeDyate incorporates several key innovations that distinguish it from existing approaches: the LaRa metric for improved layer importance characterization, budget-adaptive layer selection mechanisms, exploitation of channel importance temporal stability for probabilistic sampling, and the integration of stochastic selection strategies to replace deterministic approaches for enhanced robustness. Experimental validation demonstrates that MeDyate consistently outperforms existing approaches while maintaining memory budgets as low as a few hundred kilobytes.

Activation Maps Compression. In parallel, we addressed the critical issue of activation storage during training and developed novel compression techniques using tensor decomposition. Our HOSVD-based approach demonstrated controlled compression capabilities with theoretical convergence guarantees. To overcome computational limitations inherent in direct HOSVD application, we developed ASI (Activation Subspace Iteration), which leverages the stability of activation maps to achieve up to 120× memory reduction and 91× computational speedup compared to HOSVD while maintaining comparable performance to vanilla training.

Our contributions extend beyond algorithmic development to provide rigorous theoretical analysis, including formalization of fine-tuning dynamics through heavy-tailed gradient theory, convergence guarantees for compressed activation training, and comprehensive computational complexity analysis. Extensive experimental val-

idation across diverse architectures ranging from efficient networks like MobileNet to transformer models, multiple datasets including ImageNet, CIFAR, and Oxford Pets, and real-world deployment scenarios including Raspberry Pi implementations demonstrates the practical effectiveness of our approaches under extreme memory constraints.

8.2 Future Research Directions

The foundations established in this thesis open several promising avenues for advancing memory-efficient on-device learning capabilities.
Limitations. The primary improvement opportunity for our subnetwork selection framework lies in translating algorithmic advances into optimized on-device implementations. A key limitation of our dynamic channel selection frameworks is the absence of hardware implementation, which prevents evaluation of our proposed strategies on actual devices. Such implementation would prove essential for motivating widespread adoption of our methods in practical applications.
Another significant opportunity highlighted throughout this manuscript is the complementary nature of subnetwork selection and activation compression, suggesting substantial potential for unified frameworks that simultaneously optimize both approaches. Joint optimization strategies could achieve greater memory efficiency through coordinated parameter selection and activation compression decisions, potentially surpassing the individual benefits of each approach.
Further Research Opportunities Here we present a series of research directions naturally extending from our works.

- *Activation Derivatives Compression.* As established in the related works, weight compression represents a well-explored research area. In our work, we investigated simultaneous activation and weight derivative sparsity through subnetwork selection and activation compression through tensor decomposition. However, one critical element of neural network learning that, to our knowledge, remains largely unexplored in the literature is efficient loss backpropagation. We believe that either of the methods developed in this work could be adapted to reduce both the memory footprint and computational cost of activation derivative computation and propagation. Future work should explore activation gradient sparsity exploitation for compressed backpropagation Dai et al. [2020]; Sarma et al. [2021], potentially extending the computational and memory gains beyond what we achieved in our research.

- *Continual Learning.* We also emphasize that one primary motivation for

developing on-device learning solutions is enabling continual learning to compensate for data drift phenomena Cano and Krawczyk [2022]; Casado et al. [2022]. Although our MeDyate algorithm was developed with single and stable downstream tasks in mind, its adaptive mechanism that continually renews its sampling probability distribution based on observed gradient norms could provide significant advantages when flexibility is required to adapt to distribution changes in input data. Despite the speculative nature of this hypothesis, we believe it warrants further investigation.

- *Training from Scratch.* Both of the memory-efficient lines of work developed in this manuscript rely on the hypothesis of fine-tuning pre-trained networks, thus leaving the dynamics of training from scratch unexplored. A natural prolongation of our work would be to adapt the methods developed to full training, allowing for memory and computational savings in all phases of a network's lifetime Lee and Yoo [2021]; Mehlin et al. [2023].

- *Green AI.* Finally, although this work was developed with enabling on-device learning as the primary objective, we wish to emphasize the broader applications of our research. The immense energy footprint of training and deploying extremely large models stands in contradiction to the necessary energy reduction of all human activities in the current environmental crisis context. In this regard, enabling efficient low-energy deep neural network learning bridges the gap toward implementing green AI solutions that align with sustainability objectives Schwartz et al. [2020]; Verdecchia et al. [2023].

The convergence of these research directions promises to establish a comprehensive framework for practical on-device learning that operates effectively under severe edge device constraints while maintaining the adaptability and performance essential for real-world applications. As demand for privacy-preserving, low-latency AI systems continue to grow, this thesis provides a foundation for bringing advanced machine learning capabilities to resource-constrained environments.

Chapter 9

Bibliography

Supriya Agrahari and Anil Kumar Singh. 2022. Concept drift detection in data stream mining: A literature review. *Journal of King Saud University-Computer and Information Sciences*, 34(10):9523–9540. 4

Axel Andersson, Nadezhda Koriakina, Nataša Sladoje and Joakim Lindblad. 2022. End-to-end multiple instance learning with gradient accumulation. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 2742–2746. IEEE. 11

Lukas Bossard, Matthieu Guillaumin and Luc Van Gool. 2014. Food-101–mining discriminative components with random forests. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI 13*. 27

Andrea Bragagnolo, Enzo Tartaglione and Marco Grangetto. 2022. To update or not to update? neurons at equilibrium in deep models. *Advances in Neural Information Processing Systems*, 35. xi, 21, 22

Han Cai, Chuang Gan, Ligeng Zhu and Song Han. 2020. Tinytl: Reduce memory, not parameters for efficient on-device learning. *Advances in Neural Information Processing Systems*, 33:11285–11297. 4, 84

Han Cai, Ligeng Zhu and Song Han. 2019. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*. 26

Alberto Cano and Bartosz Krawczyk. 2022. Rose: robust online self-adjusting ensemble for continual learning on imbalanced drifting data streams. *Machine Learning*, 111(7):2561–2599. 120

Fernando E Casado, Dylan Lema, Marcos F Criado, Roberto Iglesias, Carlos V Regueiro and Senén Barro. 2022. Concept drift detection and adaptation for

federated and continual learning. *Multimedia Tools and Applications*, 81(3):3397–3419. 120

Liang-Chieh Chen, George Papandreou, Florian Schroff and Hartwig Adam. 2017. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*. 94, 98,

Tianqi Chen, Bing Xu, Chiyuan Zhang and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*. 10

Yu Cheng, Duo Wang, Pan Zhou and Tao Zhang. 2018. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35(1):126–136. 8

Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard and Rocky Rhodes. 2019. Visual wake words dataset. *arXiv preprint arXiv:1906.05721*. 27

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*. 142

MMSegmentation Contributors. 2020. Mmsegmentation: Openmmlab semantic segmentation toolbox and benchmark. 94

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth and Bernt Schiele. 2016. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223. 94

Pengcheng Dai, Jianlei Yang, Xucheng Ye, Xingzhou Cheng, Junyu Luo, Linghao Song, Yiran Chen and Weisheng Zhao. 2020. Sparsetrain: Exploiting dataflow sparsity for efficient convolutional neural networks training. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE. 119

Pierre Vilar Dantas, Waldir Sabino da Silva Jr, Lucas Carvalho Cordeiro and Celso Barbosa Carvalho. 2024. A comprehensive review of model compression techniques in machine learning. *Applied Intelligence*, 54(22):11804–11844. 3

Dorottya Demszky, Kelvin Guu and Percy Liang. 2018. Transforming question answering datasets into natural language inference datasets. *arXiv preprint arXiv:1809.02922*. 52

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. 26

Lei Deng, Guoqi Li, Song Han, Luping Shi and Yuan Xie. 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532. 8

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. *Advances in neural information processing systems*, 27. 9

Radosvet Desislavov, Fernando Martínez-Plumed and José Hernández-Orallo. 2021. Compute and energy consumption trends in deep learning inference. *arXiv preprint arXiv:2109.05472*. 3

Sauptik Dhar, Junyao Guo, Jiayi Liu, Samarth Tripathi, Unmesh Kurup and Mohak Shah. 2021. A survey of on-device machine learning: An algorithms and learning theory perspective. *ACM Transactions on Internet of Things*, 2(3):1–49. 2

Paula Fraga-Lamas, Sérgio Ivan Lopes and Tiago M Fernández-Caramés. 2021. Green iot and edge ai as key technological enablers for a sustainable digital transition towards a smart circular economy: An industry 5.0 use case. *Sensors*, 21(17):5745. 2

Robert M French. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135. 10

Joshua I Glaser, Ari S Benjamin, Raeed H Chowdhury, Matthew G Perich, Lee E Miller and Konrad P Kording. 2020. Machine learning for neural decoding. *eneuro*, 7(4). 2

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia and Kaiming He. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*. 27

Rémi Gribonval, Volkan Cevher and Mike E. Davies. 2012. Compressible distributions for high-dimensional statistics. *IEEE Transactions on Information Theory*, 58(8):5016–5034. 38

Marawan Gamal Abdel Hameed, Marzieh S Tahaei, Ali Mosleh and Vahid Partovi Nia. 2022. Convolutional neural network compression through generalized

kronecker product decomposition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 771–779. 12, 88

D Corydon Hammond. 2007. What is neurofeedback? *Journal of neurotherapy*, 10(4):25–36. 2

Song Han, Jeff Pool, John Tran and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28. 8

Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 97

Geoffrey Hinton. 2022. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*. 10

Geoffrey Hinton, Oriol Vinyals and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*. 9

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR. 10

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen and others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3. 10

Zilong Hu, Jinshan Tang, Ziming Wang, Kai Zhang, Ling Zhang and Qingling Sun. 2018. Deep learning for image-based cancer detection and diagnosis- a survey. *Pattern Recognition*, 83:134–149. 2

Ozlem Durmaz Incel and Sevda Ozge Bursa. 2023. On-device deep learning for mobile and wearable sensing applications: A review. *IEEE Sensors Journal*. 3

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713. 8

Chuan-Jia Jhang, Cheng-Xin Xue, Je-Min Hung, Fu-Chun Chang and Meng-Fan Chang. 2021. Challenges and trends of sram-based computing-in-memory for ai

edge devices. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(5):1773–1786. 3

Gal Kaplun, Andrey Gurevich, Tal Swisa, Mazor David, Shai Shalev-Shwartz and Eran Malach. 2023. Less is more: Selective layer finetuning with subtuning. 42

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. 52

Serhat Kılıçarslan, Kemal Adem and Mete Çelik. 2021. An overview of the activation functions used in deep learning algorithms. *Journal of New Results in Science*, 10(3):75–88. 4

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang and Dongjun Shin. 2015. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*. 9, 88

Timo Kirschstein and Rüdiger Köhling. 2009. What is the source of the eeg? *Clinical EEG and neuroscience*, 40(3):146–149. 2

Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*. 8

Alex Krizhevsky, Geoffrey Hinton and others. 2009. Learning multiple layers of features from tiny images. 27

Young D. Kwon, Rui Li, Stylianos I. Venieris, Jagmohan Chauhan, Nicholas D. Lane and Cecilia Mascolo. 2024. Tinytrain: Resource-aware task-adaptive sparse training of dnns at the data-scarce edge. xi, 11, 20, 21, 26

Yann LeCun, John Denker and Sara Solla. 1989. Optimal brain damage. *Advances in neural information processing systems*, 2. 8

Jinsu Lee and Hoi-Jun Yoo. 2021. An overview of energy-efficient hardware accelerators for on-device deep-neural-network training. *IEEE Open Journal of the Solid-State Circuits Society*, 1:115–128. 120

Namhoon Lee, Thalaiyasingam Ajanthan and Philip HS Torr. 2018. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*. 8

Yoonho Lee, Annie S Chen, Fahim Tajwar, Ananya Kumar, Huaxiu Yao, Percy Liang and Chelsea Finn. 2022. Surgical fine-tuning improves adaptation to distribution shifts. *arXiv preprint arXiv:2210.11466*. 10, 19, 42

En Li, Liekang Zeng, Zhi Zhou and Xu Chen. 2019. Edge ai: On-demand accelerating deep neural network inference via edge computing. *IEEE transactions on wireless communications*, 19(1):447–457. 3

Jing Li, Ji-hang Cheng, Jing-yuan Shi and Fei Huang. 2012. Brief introduction of back propagation (bp) neural network algorithm and its improvement. In *Advances in computer science and information engineering: volume 2*, pages 553–558. Springer. 4

Tailin Liang, John Glossner, Lei Wang, Shaobo Shi and Xiaotong Zhang. 2021. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403. 3

Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han and others. 2020. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems*, 33. 8, 26, 94

Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan and Song Han. 2022. On-device training under 256kb memory. *Advances in Neural Information Processing Systems*, 35. xi, 11, 17, 18, 19, 20, 26, 27, 38, 67, 72, 74

Seppo Linnainmaa. 1970. *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. Ph.D. thesis, Master's Thesis (in Finnish), Univ. Helsinki. 3

Yinhan Liu. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 364. 52

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022. 51

Jonathan Long, Evan Shelhamer and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440. 98,

Ivan Markovsky. 2008. Structured low-rank approximation and its applications. *Automatica*, 44(4):891–909. 12

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR. 94

Vanessa Mehlin, Sigurd Schacht and Carsten Lanquillon. 2023. Towards energy-efficient deep learning: An overview of energy-efficient approaches along the deep learning lifecycle. *arXiv preprint arXiv:2303.01980*. 120

Tobias Meuser, Lauri Lovén, Monowar Bhuyan, Shishir G Patil, Schahram Dustdar, Atakan Aral, Suzan Bayhan, Christian Becker, Eyal De Lara and Aaron Yi Ding, et al. 2024. Revisiting edge ai: Opportunities and challenges. *IEEE Internet Computing*, 28(4):49–59. 3

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev and Ganesh Venkatesh, et al. 2017. Mixed precision training. *arXiv preprint arXiv:1710.03740*. 11

Mohammad Mohammadi, Adel Mohammadpour and Hiroaki Ogata. 2015. On estimating the tail index and the spectral measure of multivariate α-stable distributions. *Metrika*, 78(5):549–561. 45

Le-Trung Nguyen, Aël Quélennec, Van-Tam Nguyen and Enzo Tartaglione. 2025. Beyond low-rank decomposition: A shortcut approach for efficient on-device learning. *arXiv preprint arXiv:2505.05086*. xiv, xv, xvi, 6, 106, 108, 110, 112, 113, 114, 115,

Le-Trung Nguyen, Aël Quélennec, Enzo Tartaglione, Samuel Tardieu and Van-Tam Nguyen. 2024. Activation map compression through tensor decomposition for deep learning. *Advances in Neural Information Processing Systems*, 37:130384–130407. xiii, xiv, xv, xvi, 5, 85, 93, 94, 96, 97, 98, 99, 138, 139,

Luis Fernando Nicolas-Alonso and Jaime Gomez-Gil. 2012. Brain computer interfaces, a review. *sensors*, 12(2):1211–1279. 2

Maria-Elena Nilsback and Andrew Zisserman. 2008. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE. 27

Jinhui Ouyang, Mingzhu Wu, Xinglin Li, Hanhui Deng and Di Wu. 2024. Briedge: Eeg-adaptive edge ai for multi-brain to multi-robot interaction. *arXiv preprint arXiv:2403.15432*. 2

Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman and CV Jawahar. 2012. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3498–3505. IEEE. 27

Adam Poliak. 2020. A survey on recognizing textual entailment as an nlp evaluation. *arXiv preprint arXiv:2010.03061*. 52

Edson David Pontes, Mauro Pinto, Fábio Lopes and César Teixeira. 2024. Concept-drifts adaptation for machine learning eeg epilepsy seizure prediction. *Scientific Reports*, 14(1):8204. 4

Mangesh Pujari, Anil Kumar Pakina and Anshul Goel. 2023. Balancing innovation and privacy: A red teaming approach to evaluating phone-based large language models under ai privacy regulations. *International Journal Science and Technology*, 2(3). 2

Aël Quélennec, Enzo Tartaglione, Pavlo Mozharovskyi and Van-Tam Nguyen. 2024. Towards on-device learning on the edge: Ways to select neurons to update under a budget constraint. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 685–694. xi, 5, 24, 26

Aël Quélennec, Nour Hezbri, Pavlo Mozharovskyi, Van-Tam Nguyen and Enzo Tartaglione. 2025a. Study of training dynamics for memory-constrained fine-tuning. xi, xii, xv, 5, 35, 43, 45, 46, 47, 48, 50, 51, 52, 53, 54,

Aël Quélennec, Pavlo Mozharovskyi, Van-Tam Nguyen and Enzo Tartaglione. 2025b. Memory constrained dynamic subnetwork update for transfer learning. xii, xiii, xiv, xv, xvi, 5, 62, 65, 67, 68, 69, 70, 71, 72, 73, 74, 76, 78, 136,

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever and others. 2018. Improving language understanding by generative pre-training. ... 2

Berkman Sahiner, Weijie Chen, Ravi K Samala and Nicholas Petrick. 2023. Data drift in medical machine learning: implications and potential remedies. *The British Journal of Radiology*, 96(1150):20220878. 4

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520. 8, 26

Anup Sarma, Sonali Singh, Huaipan Jiang, Ashutosh Pattnaik, Asit K Mishra, Vijaykrishnan Narayanan, Mahmut T Kandemir and Chita R Das. 2021. Exploiting activation based gradient output sparsity to accelerate backpropagation in cnns. *arXiv preprint arXiv:2109.07710*. 119

Roy Schwartz, Jesse Dodge, Noah A Smith and Oren Etzioni. 2020. Green ai. *Communications of the ACM*, 63(12):54–63. 120

Jaime Sevilla, Lennart Heim, Anson Ho, Tamay Besiroglu, Marius Hobbhahn and Pablo Villalobos. 2022. Compute trends across three eras of machine learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE. 3

Umut Simsekli, Mert Gürbüzbalaban, Thanh Huy Nguyen, Gaël Richard and Levent Sagun. 2019. On the heavy-tailed theory of stochastic gradient descent for deep neural networks. *arXiv preprint arXiv:1912.00018*, 222. 36, 45

Jake Snell, Kevin Swersky and Richard S. Zemel. 2017. Prototypical networks for few-shot learning. 20

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics. 52

GW Stewart and JH Miller. 1975. Methods of simultaneous iteration for calculating eigenvectors of matrices. *Topics in Numerical Analysis II*, 2. 103, 105

Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*. 8

Paul Audu Tonga, Zubaida Said Ameen, Auwalu Saleh Mubarak and Fadi Al-Turjman. 2022. A review on on device privacy and machine learning training. In *2022 International Conference on Artificial Intelligence in Everything (AIE)*, pages 679–684. IEEE. 2

M Alex O Vasilescu and Demetri Terzopoulos. 2003. Multilinear subspace analysis of image ensembles. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 2, pages II–93. IEEE. 87

Roberto Verdecchia, June Sallou and Luís Cruz. 2023. A systematic review of green ai. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(4):e1507. 120

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra and others. 2016. Matching networks for one shot learning. *Advances in neural information processing systems*, 29. 20

Aladin Virmaux and Kevin Scaman. 2018. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 31. 111

Thijs Vogels, Sai Praneeth Karimireddy and Martin Jaggi. 2019. Powersgd: Practical low-rank gradient compression for distributed optimization. *Advances in Neural Information Processing Systems*, 32. xvii, 105, 110

Yijun Wan, Melih Barsbey, Abdellatif Zaidi and Umut Simsekli. 2023. Implicit compressibility of overparametrized neural networks trained with heavy-tailed sgd. *arXiv preprint arXiv:2306.08125*. 37, 38, 39, 66

Guagnyu Wang, Wenchao Liu, Yuhong He, Cong Xu, Lin Ma and Haifeng Li. 2024. Eegpt: Pretrained transformer for universal and reliable representation of eeg signals. In *Advances in Neural Information Processing Systems*, volume 37, pages 39249–39280. Curran Associates, Inc. 2

Han Wang, Balaji Muthurathinam Panneer Chelvan, Muhammed Golec, Sukhpal Singh Gill and Steve Uhlig. 2025. Healthedgeai: Gai and xai based healthcare system for sustainable edge ai and cloud computing environments. *Concurrency and Computation: Practice and Experience*, 37(9-11):e70057. 2

Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie and Pietro Perona. 2010. Caltech-ucsd birds 200. Technical Report CNS-TR-201, Caltech. 27

Qingsong Wen, Jing Liang, Carles Sierra, Rose Luckin, Richard Tong, Zitao Liu, Peng Cui and Jiliang Tang. 2024. Ai for education (ai4edu): Advancing personalized education with llm and adaptive learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6743–6744. 2

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen and Hai Li. 2016. Learning structured sparsity in deep neural networks. 8

Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang and Jian Sun. 2018. Unified perceptual parsing for scene understanding. In *Proceedings of the European conference on computer vision (ECCV)*, pages 418–434. 98,

Ou Xinwei, Chen Zhangxin, Zhu Ce and Liu Yipeng. 2023. Low rank optimization for efficient deep learning: Making a balance between compact architecture and fast training. *Journal of Systems Engineering and Electronics*. 12, 87

Dianlei Xu, Tong Li, Yong Li, Xiang Su, Sasu Tarkoma, Tao Jiang, Jon Crowcroft and Pan Hui. 2021. Edge intelligence: Empowering intelligence to the edge of network. *Proceedings of the IEEE*, 109(11):1778–1837. 3

Jian Xue, Jinyu Li and Yifan Gong. 2013. Restructuring of deep neural network acoustic models with singular value decomposition. In *Interspeech*, pages 2365–2369. 12

Yuedong Yang, Hung-Yueh Chiang, Guihong Li, Diana Marculescu and Radu Marculescu. 2023a. Efficient low-rank backpropagation for vision transformer adaptation. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 14725–14736. 12

Yuedong Yang, Guihong Li and Radu Marculescu. 2023b. Efficient on-device training via gradient filtering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3811–3820. 10, 94, 95, 97, 136, 137

Jason Yosinski, Jeff Clune, Yoshua Bengio and Hod Lipson. 2014. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27. 9

Zhihang Yuan, Yuzhang Shang, Yue Song, Qiang Wu, Yan Yan and Guangyu Sun. 2023. Asvd: Activation-aware singular value decomposition for compressing large language models. *arXiv preprint arXiv:2312.05821*. 106

Chiyuan Zhang, Samy Bengio and Yoram Singer. 2019. Are all layers created equal? *ArXiv*. 51

Jianyu Zhang and Léon Bottou. 2024. Fine-tuning with very large dropout. 42

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang and Wei Lu. 2024. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*. 142

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen and Tuo Zhao. 2023. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*. 104

Xiangyu Zhang, Jianhua Zou, Kaiming He and Jian Sun. 2015. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955. 87

Hengling Zhao, Yipeng Liu, Xiaolin Huang and Ce Zhu. 2021. Semi-tensor product-based tensordecomposition for neural network compression. *arXiv preprint arXiv:2109.15200*. 12

Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang and Jiaya Jia. 2017. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890. 98,

Sha Zhu, Kaoru Ota and Mianxiong Dong. 2021. Green ai for iiot: Energy efficient intelligent edge computing for industrial internet of things. *IEEE Transactions on Green Communications and Networking*, 6(1):79–88. 4

Sha Zhu, Kaoru Ota and Mianxiong Dong. 2022. Energy-efficient artificial intelligence of things with intelligent edge. *IEEE Internet of Things Journal*, 9(10):7525–7532. 4

Chapter A

Appendix

Contents

A.1 HOSVD Detailed Weight Derivatives Computation

Since the convolution operation involves other variables such as stride, dilation, and groups, we provide a more precise description by rewriting in Eq. (3.2), $\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i}$ as $\Delta \mathcal{W} \in \mathbb{R}^{N_{out} \times C \times D \times D}$, $\mathcal{A}_i$ as $\mathcal{I} \in \mathbb{R}^{B \times C \times H \times W}$ and $\frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}}$ as $\Delta \mathcal{Y} \in \mathbb{R}^{B \times C' \times H' \times W'}$ which gives us:

$$\Delta \mathcal{W}_{c'_g,c,k,l} = \sum_{b=1}^{B} \sum_{h'=1}^{H'} \sum_{w'=1}^{W'} \underline{\mathcal{I}}_{b,c_g,h,w} \Delta \mathcal{Y}_{b,c'_g,h',w'},$$

where:

$$\begin{aligned}
&h = h' \times \text{Stride} + k \times \text{Dilation},\\
&w = w' \times \text{Stride} + l \times \text{Dilation},\\
&N_{out} = \left\lfloor \frac{\text{Groups}}{C'} \right\rfloor,\\
&N_{in} = \left\lfloor \frac{\text{Groups}}{C} \right\rfloor,\\
&c'_g = g \times N_{out} + c',\\
&c_g = g \times N_{in} + c,\\
&c' \in [1, N_{out}],\\
&c \in [1, N_{in}],\\
&g \in [1, \text{Groups}],\\
&k \text{ and } l \in [1, D],\\
&\underline{\mathcal{I}} \text{ is the padded input.}
\end{aligned} \tag{A.1}$$

In our case, before being saved in memory, $\mathcal{I}$ is decomposed and compressed through truncated HOSVD as presented in (6.6):

$$\tilde{\mathcal{I}}_{b,c_g,h,w} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} \sum_{k_3=1}^{K_3} \sum_{k_4=1}^{K_4} \hat{\mathcal{S}}_{k_1,k_2,k_3,k_4} \times U^{(1)}_{(K_1)_{b,k_1}} \times U^{(2)}_{(K_2)_{c_g,k_2}} \times U^{(3)}_{(K_3)_{h,k_3}} \times U^{(4)}_{(K_4)_{w,k_4}}. \tag{A.2}$$

Therefore, in the backward pass, its padded version can be restored using the

following formula:

$$\underline{\tilde{\mathcal{I}}}_{b,c_g,h,w} = \sum_{k_1=1}^{K_1}\sum_{k_2=1}^{K_2}\sum_{k_3=1}^{K_3}\sum_{k_4=1}^{K_4} \hat{\mathcal{S}}_{k_1,k_2,k_3,k_4} \times U^{(1)}_{(K_1)_{b,k_1}} \times U^{(2)}_{(K_2)_{c_g,k_2}} \times \underline{U}^{(3)}_{(K_3)_{h,k_3}} \times \underline{U}^{(4)}_{(K_4)_{w,k_4}}, \tag{A.3}$$

where, $\underline{U}^{(3)}_{(K_3)}$ and $\underline{U}^{(4)}_{(K_4)}$ are $U^{(3)}_{(K_3)}$ and $U^{(4)}_{(K_4)}$ with vertical padding (top and bottom edges only), respectively. Then, we substitute (A.3) into (4.3) which gives us, after reordering and grouping:

$$\mathcal{Z}^{(1)}_{k_1,c'_g,h',w'} = \sum_{b=1}^{B} \Delta\mathcal{Y}_{b,c'_g,h',w'} U^{(1)}_{(K_1)_{b,k_1}}, \tag{A.4}$$

$$\mathcal{Z}^{(2)}_{k_1,k_2,h,k_4} = \sum_{k_3=1}^{K_3} \hat{\mathcal{S}}_{k_1,k_2,k_3,k_4} \underline{U}^{(3)}_{(K_3)_{h,k_3}}, \tag{A.5}$$

$$\mathcal{Z}^{(3)}_{k_1,k_2,h,w} = \sum_{k_4=1}^{K_4} Z^{(2)}_{k_1,k_2,h,k_4} \underline{U}^{(4)}_{(K_4)_{w,k_4}}, \tag{A.6}$$

$$\mathcal{Z}^{(4)}_{c'_g,k_2,k,l} = \sum_{h'=1}^{H'}\sum_{w'=1}^{W'}\sum_{k_1=1}^{K_1} Z^{(3)}_{k_1,k_2,h,w} Z^{(1)}_{k_1,c'_g,h',w'}, \tag{A.7}$$

$$\Delta\mathcal{W}_{c'_g,c,k,l} = \sum_{k_2=1}^{K_2} Z^{(4)}_{c'_g,k_2,k,l} U^{(2)}_{(K_2)_{c_g,k_2}}. \tag{A.8}$$

Computing (A.4), (A.5), (A.6) and (A.8) corresponds to performing 1×1 convolutions while (A.7) is a $H' \times W'$ convolution, resulting in (6.7).

A.2 Additional Experimental Results

A.2.1 MeDyate Alternative Probability Initialization

Both TraDy and MeDyate rely on uniform probability distributions for channel sampling, either maintained throughout the fine-tuning process or established at initialization. This design choice treats all channels within the selected layer subset as equally important. Recognizing that layer importance rankings are available through the LaRa metric, we explored an alternative sampling strategy where channel selection probabilities are weighted proportionally to their respective layer's LaRa value. This *boosting* mechanism aims to guide channel selection

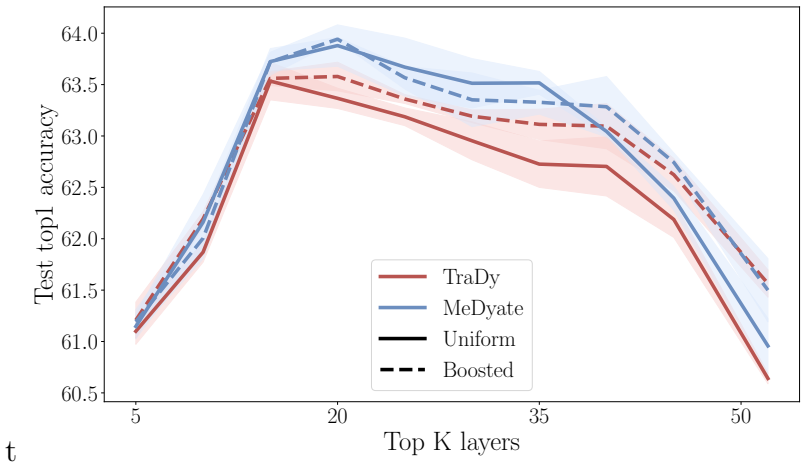


Figure A.1: Effects of probability distribution in probabilistic sampling approaches with respect to the number of layers selected when fine-tuning MobileNetV2 on Food dataset under memory constraints. Results show test top-1 accuracy across different numbers of selected layers K. Credits to Quélennec et al. [2025b].

more effectively in TraDy and facilitate faster convergence toward optimal channel distributions in MeDyate.
Fig. A.1 presents the final test top-1 accuracy as a function of the number of selected LaRa-ranked layers K, comparing uniform and boosted variants for both methods. The boosting strategy demonstrates its primary impact at higher K values, where it reduces performance degradation. Nevertheless, within the optimal K range that yields peak accuracy, boosting provides only marginal gains for TraDy and shows no statistically significant improvement for MeDyate.

A.2.2 HOSVD Supplementary Experimental Details and Results

Detailed Training Configurations. Our experimental protocols maintain consistency with established benchmarking standards while incorporating necessary modifications for compression evaluation:

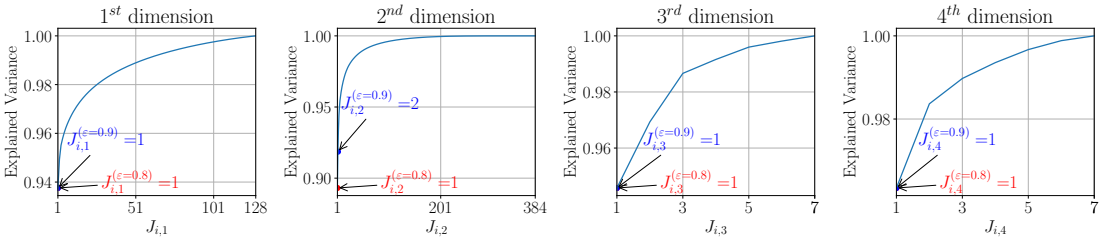
- For *ImageNet classification*, we adopt training policies similar to Yang et al. [2023b] with specific adaptations. Models undergo fine-tuning for 90 epochs incorporating L2 gradient clipping with threshold equal to 2.0. Optimization employs SGD with weight decay 1×10^{-4} and momentum of 0.9. Data preprocessing includes random resizing, horizontal flipping, and normalization, with batch sizes of 64 elements. Cross-entropy loss provides the optimization objective. A key modification involves the initial learning rate configuration:

we implement linear warm-up over 4 epochs reaching 0.005 (compared to 0.05 in Yang et al. [2023b]), followed by cosine annealing decay for subsequent epochs.

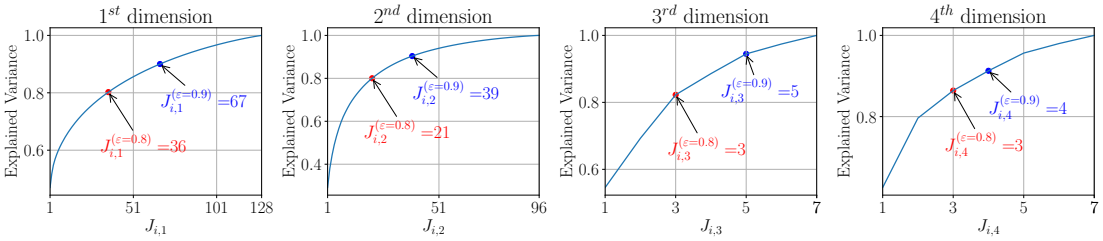
- For *other dataset fine-tuning*, we maintain hyperparameter consistency with Yang et al. [2023b] across both experimental setups. Training employs cross-entropy loss with SGD optimization, initial learning rate 0.05 with cosine annealing decay, zero momentum, and weight decay 1×10^{-4}. L2 gradient clipping maintains threshold value to 2.0. Setup B incorporates batch normalization fusion with convolutional layers—a standard inference acceleration technique. Batch size increases to 128 with standard data normalization preprocessing.

- For *semantic segmentation*, we utilize pre-trained and calibrated checkpoints from Yang et al. [2023b] to ensure fair comparison. Models trained on Cityscapes via MMSegmentation undergo fine-tuning on augmented Pascal-VOC12. Optimization begins with learning rate 0.01 following cosine annealing schedule, weight decay 5×10^{-4}, and momentum 0.9. Training proceeds for $20,000$ steps with batch size 8. Data augmentation encompasses random cropping, horizontal flipping, photometric distortions, and normalization, with cross-entropy serving as the loss function.

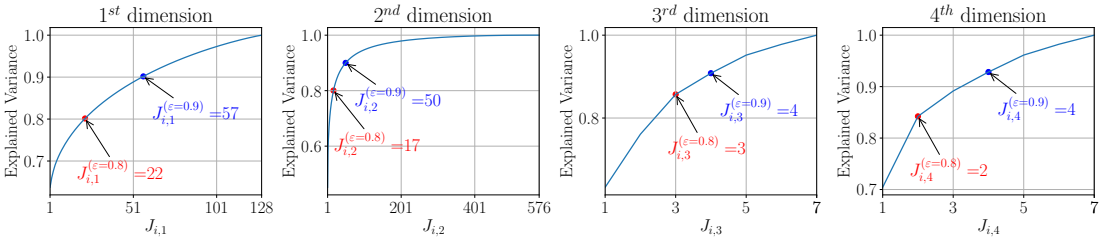
Additional Explained Variance Visualisations. Fig. A.2 presents the variation of explained variance as a function of $J_{i,k}$ when fine-tuning the last four layers of a MCUNet on CIFAR-10 and decomposing the activation maps with HOSVD. In general, we observe that for each layer i and dimension k, 80% of the explained variance is contained in less than 20% of the principal components.
Maintaining the same experimental framework, Fig. A.3 shows that the typical evolution $J_{i,k}$ tends to gradually increase in the initial training epoch and is then followed by a decline phase. Intuitively, we can understand this trend under as the initial fine-tuning epochs corresponding to large modifications of the architecture to adapt to the downstream task, thus resulting in more components being necessary to account for the same level of explained variance. Over time, the model learns more efficient representations of the downstream task features leading to intrinsically more compressible activation maps and a gradual decrease in $J_{i,k}$.



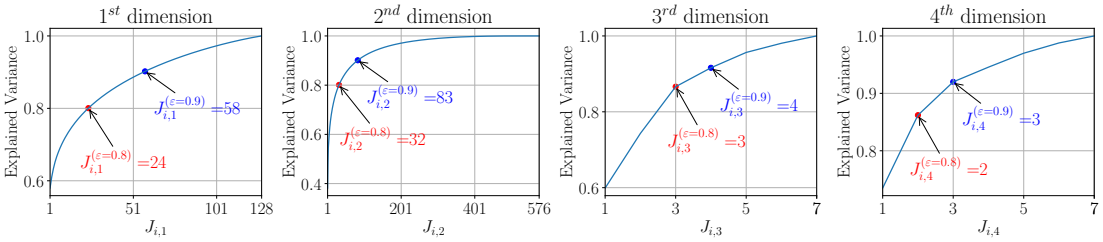
(a) Last Layer ($i = 42$).



(b) 2^{nd} Last Layer ($i = 41$).

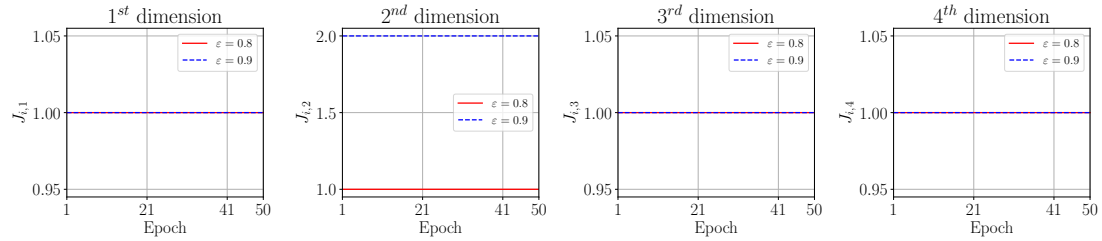


(c) 3^{rd} Last Layer ($i = 40$).

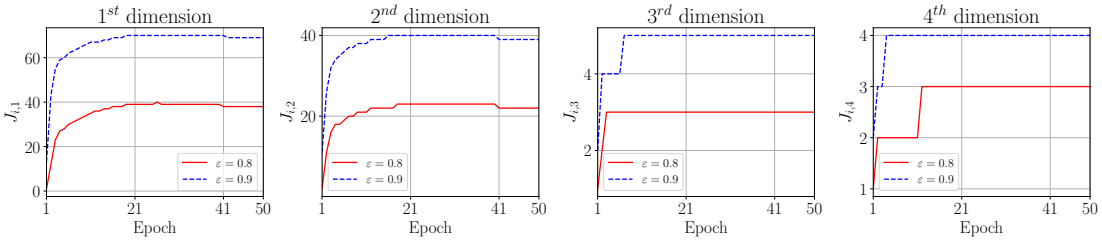


(d) 4^{th} Last Layer ($i = 39$).

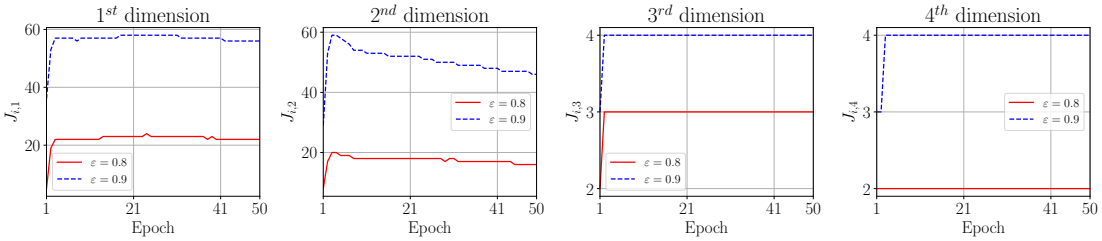
Figure A.2: $J_{i,k}$ and variance for each of the last four layers of an MCUNet when fine-tuning them using HOSVD on CIFAR-10 following setup A. Credits to Nguyen et al. [2024].



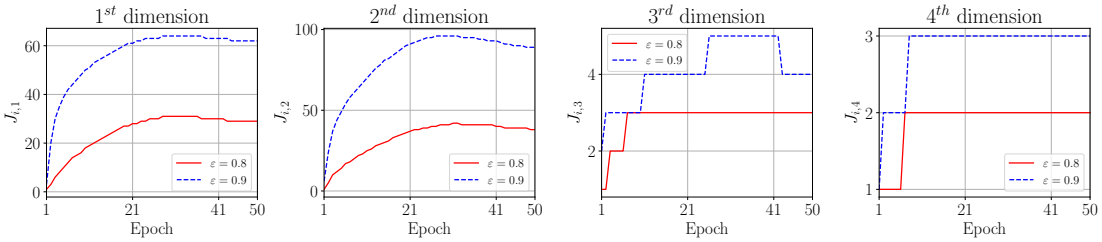
(a) Last Layer ($i = 42$).



(b) 2^{nd} Last Layer ($i = 41$).



(c) 3^{rd} Last Layer ($i = 40$).



(d) 4^{th} Last Layer ($i = 39$).

Figure A.3: Evolution of $J_{i,k}$ during training for each of the last four layers of an MCUNet when fine-tuning them using HOSVD on CIFAR-10 dataset following setup A with two different values of ε. Credits to Nguyen et al. [2024].

A.3 Full Result Tables

A.3.1 Velocity Results Table

Tab. A.1 presents the complete set of results described in Sec. 3.3.2. It appears clearly that while Velocity consistently outperforms SU, it remains below the naive random neuron selection baseline.

A.3.2 TraDy Results Tables

To facilitate visualization, we plot in separate tables results on static and dynamic strategies.

- **Vision Results.** In Tab. A.2 we represent a report of final test top-1 accuracies across our full experimental matrix spanning multiple architectures (CNN and SwinT), datasets, and memory budgets when comparing SU and the dynamic selection strategies.
  The corresponding table with the static strategies is provided in Tab. A.3.

- **NLP Results.** In Tab. A.4 we represent a report of final test top-1 accuracies across our full NLP experimental matrix as presented in Sec. 4.3.3 when comparing the dynamic subnetwork selection strategies.
  The corresponding table with the static strategies is provided in Tab. A.5.

A.3.3 MeDyate Results Tables

Tab. A.6 displays results for CNN architectures, while Tab. A.7 and Tab. A.8 respectively presents results with SwinT and NLP models.

A.3.4 HOSVD Results Table

In addition to the main experiments, we also performed many classification experiments with setup A on various datasets, the results are shown in Tab. A.9. Especially, when fine-tuning SwinT on CUB200, Pets, and CIFAR-100, applying HOSVD to the last four layers (maintaining the rest of the architecture frozen) consumes less memory than fully fine-tuning the last two layers. This happens because the variance of the activation maps is concentrated in few components: for the same ε, $J_{i,k}$ will be smaller.

Table A.1: Comparison of final top1 test accuracies between SU, Velocity and random neuron selection over various pretrained models, datasets, and budgets.

Model	B^{w}_{mem}	Method	CIFAR-10	CIFAR-100	CUB	Flowers	Food	Pets	VWW	Average
MbV2-w0.35	22 074	SU	89.10±0.26	67.34±0.18	56.85±0.22	80.33±0.56	61.62±0.13	76.53±0.26	87.73±0.06	74.21±0.74
		Velocity	89.84±0.19	68.14±0.17	57.51±0.30	79.46±0.32	61.79±0.12	76.24±0.23	88.29±0.18	74.47±0.60
		Random	90.51±0.09	69.34±0.45	58.69±0.46	79.89±0.65	63.86±0.22	76.56±0.43	88.84±0.11	75.38±1.04
	55 920	SU	90.42±0.12	68.73±0.29	57.97±0.25	81.15±0.51	64.56±0.17	77.04±0.28	87.76±0.16	75.38±0.74
		Velocity	90.94±0.31	69.74±0.45	58.45±0.59	80.13±0.62	65.43±0.23	77.14±0.37	88.31±0.22	75.73±1.13
		Random	91.42±0.17	70.65±0.21	59.56±0.54	80.52±0.75	66.81±0.17	76.73±0.46	88.24±0.41	76.28±1.16
	81 792	SU	90.69±0.17	69.17±0.09	57.92±0.35	81.09±0.39	65.33±0.23	77.12±0.16	87.3±0.32	75.52±0.70
		Velocity	91.37±0.19	70.62±0.13	59.03±0.29	80.57±0.26	66.69±0.3	76.67±0.33	88.11±0.3	76.15±0.70
		Random	91.80±0.10	71.12±0.24	59.93±0.34	80.88±0.81	67.79±0.20	76.99±0.46	87.87±0.21	76.63±1.07
	288 864	Baseline	92.72±0.03	72.69±0.16	60.03±0.18	81.88±0.34	70.79±0.20	76.68±0.33	88.58±0.19	77.62±0.60
MCUNet-in1	12 096	SU	89.51±0.23	68.41±0.27	60.68±0.27	82.92±0.43	65.57±0.06	81.15±0.29	89.14±0.1	76.77±0.69
		Velocity	89.93±0.11	69.20±0.15	60.69±0.50	82.05±0.34	64.42±0.21	81.21±0.25	89.53±0.08	76.72±0.72
		Random	90.81±0.10	70.03±0.40	61.70±0.15	82.38±0.52	67.46±0.18	81.30±0.36	89.95±0.09	77.66±0.80
	52 480	SU	91.65±0.26	70.96±0.23	62.03±0.32	83.79±0.53	69.77±0.03	81.52±0.11	88.67±0.14	78.34±0.73
		Velocity	92.24±0.03	72.31±0.24	62.56±0.42	82.80±0.51	70.42±0.17	81.40±0.59	89.37±0.23	78.73±0.96
		Random	92.43±0.21	73.02±0.44	62.93±0.61	82.64±0.47	71.48±0.20	81.27±0.44	89.42±0.31	79.03±1.08
	98 560	SU	92.07±0.13	71.58±0.15	61.44±0.41	83.74±0.47	71.02±0.15	81.07±0.24	88.77±0.31	78.53±0.78
		Velocity	92.90±0.11	73.66±0.29	62.53±0.51	82.99±0.54	72.32±0.14	80.87±0.45	89.36±0.04	79.23±0.93
		Random	93.02±0.03	73.92±0.16	62.55±0.25	82.9±0.48	73.07±0.09	81.11±0.34	89.65±0.09	79.46±0.67
	463 216	Baseline	93.87±0.10	76.03±0.18	61.62±0.62	83.45±0.42	75.74±0.14	79.49±0.60	90.06±0.16	80.04±1.00
Proxyless-w0.3	20 736	SU	91.00±0.25	68.94±0.16	57.04±0.36	82.36±0.25	63.30±0.11	78.96±0.43	88.26±0.26	75.69±0.74
		Velocity	90.69±0.10	69.12±0.06	55.98±0.12	81.85±0.34	61.46±0.13	78.58±0.50	88.67±0.20	75.19±0.67
		Random	91.71±0.20	70.69±0.17	58.04±0.17	82.08±0.13	65.08±0.14	78.8±0.42	88.98±0.17	76.48±0.58
	62 976	SU	91.88±0.27	70.34±0.19	58.33±0.36	83.15±0.28	66.49±0.29	78.99±0.74	87.82±0.12	76.71±0.98
		Velocity	92.37±0.03	71.84±0.08	58.72±0.72	82.35±0.20	66.63±0.13	78.9±0.44	88.53±0.18	77.05±0.90
		Random	92.77±0.02	72.86±0.29	59.77±0.36	82.77±0.22	68.59±0.17	78.99±0.51	88.52±0.40	77.75±0.84
	89 088	SU	92.42±0.16	71.32±0.12	58.52±0.25	83.24±0.25	67.18±0.09	79.03±0.24	87.92±0.17	77.09±0.51
		Velocity	92.82±0.23	72.51±0.22	59.71±0.46	82.61±0.46	68.18±0.06	79.01±0.22	88.40±0.15	77.61±0.77
		Random	93.10±0.18	73.29±0.30	59.86±0.27	83.03±0.44	69.72±0.16	78.9±0.61	88.46±0.09	78.05±0.89
	357 680	Baseline	93.71±0.12	74.81±0.13	61.75±0.12	84.44±0.50	72.98±0.09	78.53±0.10	88.95±0.04	79.31±0.56

A.3.5 ASI Results Tables

Additional Classification Results. We compare the performance of various techniques across different models, all of which are pretrained on ImageNet-1K. The results, presented in Tab. A.10, demonstrate that ASI consistently achieves a comparable compression rate to HOSVD_ε while maintaining similar accuracy. Furthermore, ASI yields a substantial reduction in FLOPs, even outperforming vanilla training in terms of computational efficiency.

Semantic Segmentation. Tab. A.11 reports the performance of the evaluated techniques on the segmentation task. As previously observed, ASI continues to demonstrate superior performance in terms of both compression ratio and computational cost, confirming its effectiveness beyond classification.

ASI on Large Language Models. In this experiment, we fine-tune TinyLlama 1B (Zhang et al. [2024]) on the BoolQ (Clark et al. [2019]) dataset to compare ASI with vanilla training. The dataset is divided into batches of size 8, with each sample having a maximum length of 512 tokens. Due its computational cost, applying HOSVD_ε to this model is infeasible. Therefore, instead of budget-based compression, we directly set the compression rank to 20. Tab. A.12 presents the results when fine-tuning from 1 and 5 layers. Notably, fine-tuning 5 layers leads to up to a 2500× reduction in activation memory, and computational cost decreases by approximately 1.9×, while retaining decent accuracy. Remarkably, as the number of fine-tuned layers increases, both memory compression and FLOP savings improve significantly under ASI.

Table A.2: Comparison of final top1 test accuracies between SU and dynamic channel selection strategies over various pretrained architectures, datasets, and budgets. Credits to Quélennec et al. [2025a].

Model	B_{mem}	Method	CIFAR-10	CIFAR-100	CUB	Flowers	Food	Pets	VWW	Average
MbV2-w0.35	27 946	SU	89.10±0.26	67.34±0.18	56.85±0.22	80.33±0.56	61.62±0.13	76.53±0.26	87.73±0.06	74.22±0.74
		D-Full Random	89.32±0.15	67.85±0.30	57.42±0.12	79.19±0.26	60.69±0.16	76.63±0.19	88.56±0.12	74.24±0.52
		D-Det RGN	89.29±0.08	67.48±0.05	57.70±0.36	79.94±0.54	61.88±0.17	76.80±0.04	88.36±0.21	74.49±0.71
		TRaDy	89.88±0.19	68.68±0.17	57.90±0.08	79.57±0.52	62.61±0.15	76.99±0.17	88.76±0.15	74.91±0.64
	66 592	SU	90.42±0.12	68.73±0.29	57.97±0.25	81.15±0.51	64.56±0.17	77.04±0.28	87.76±0.16	75.37±0.74
		D-Full Random	90.06±0.08	68.93±0.28	58.44±0.15	79.59±0.45	62.96±0.23	76.88±0.13	88.76±0.34	75.09±0.70
		D-Det RGN	90.26±0.05	68.82±0.13	58.73±0.10	80.58±0.40	64.22±0.20	76.65±0.52	88.25±0.15	75.36±0.72
		TRaDy	90.79±0.21	69.57±0.27	59.09±0.15	80.09±0.51	64.96±0.22	76.64±0.11	88.22±0.32	75.62±0.75
	93 696	SU	90.69±0.17	69.17±0.09	57.92±0.35	81.09±0.39	65.33±0.23	77.12±0.16	87.30±0.32	75.52±0.70
		D-Full Random	90.69±0.16	69.41±0.22	58.74±0.08	79.99±0.51	63.90±0.22	76.51±0.40	88.85±0.22	75.44±0.77
		D-Det RGN	90.70±0.13	69.41±0.28	58.86±0.20	80.93±0.43	65.48±0.07	76.96±0.23	87.84±0.06	75.74±0.62
		TRaDy	90.95±0.33	70.04±0.03	58.91±0.15	80.76±0.37	65.89±0.04	77.21±0.32	88.01±0.35	75.97±0.70
	1 252 320	Baseline	92.72±0.03	72.69±0.16	60.03±0.18	81.88±0.34	70.79±0.20	76.68±0.33	88.58±0.19	77.62±0.60
MCUNet-in1	15 936	SU	89.51±0.23	68.41±0.27	60.68±0.27	82.92±0.43	65.57±0.06	81.15±0.29	89.14±0.10	76.77±0.69
		D-Full Random	90.22±0.06	69.08±0.24	61.21±0.22	82.37±0.26	65.71±0.16	81.20±0.16	89.96±0.05	77.11±0.48
		D-Det RGN	90.29±0.2	69.06±0.28	61.03±0.40	82.34±0.37	65.95±0.07	81.07±0.13	89.90±0.19	77.09±0.69
		TRaDy	90.38±0.18	69.72±0.14	61.30±0.20	82.54±0.59	66.78±0.17	81.10±0.11	89.79±0.27	77.37±0.74
	64 832	SU	91.65±0.26	70.96±0.23	62.03±0.32	83.79±0.53	69.77±0.03	81.52±0.11	88.67±0.14	78.34±0.73
		D-Full Random	91.70±0.13	71.58±0.18	62.43±0.10	82.33±0.31	69.07±0.28	81.26±0.09	89.75±0.16	78.30±0.52
		D-Det RGN	91.60±0.19	71.11±0.15	61.86±0.36	82.99±0.56	69.53±0.19	80.97±0.92	89.32±0.04	78.20±1.18
		TRaDy	92.16±0.25	72.11±0.40	62.20±0.10	83.02±0.52	70.57±0.17	81.11±0.28	89.30±0.27	78.64±0.83
	112 640	SU	92.07±0.13	71.58±0.15	61.44±0.41	83.74±0.47	71.02±0.15	81.07±0.24	88.77±0.31	78.53±0.78
		D-Full Random	92.20±0.18	72.71±0.16	62.85±0.11	82.84±0.03	70.70±0.05	81.30±0.07	89.54±0.17	78.88±0.33
		D-Det RGN	92.01±0.03	72.30±0.13	62.36±0.56	83.02±0.37	71.16±0.32	80.76±0.27	89.15±0.17	78.68±0.82
		TRaDy	92.53±0.21	72.95±0.27	62.12±0.14	83.25±0.36	71.88±0.12	81.29±0.25	89.39±0.31	79.06±0.66
	1 309 808	Baseline	93.87±0.10	76.03±0.18	61.62±0.62	83.45±0.42	75.74±0.14	79.49±0.60	90.06±0.16	80.04±1.00
Proxyless-w0.3	25 984	SU	91.00±0.25	68.94±0.16	57.04±0.36	82.36±0.25	63.30±0.11	78.96±0.43	88.26±0.26	75.69±0.74
		D-Full Random	90.76±0.23	69.20±0.24	56.55±0.13	81.54±0.64	62.69±0.12	78.64±0.29	88.90±0.11	75.47±0.80
		D-Det RGN	91.06±0.04	69.20±0.16	57.70±0.34	81.80±0.64	64.22±0.16	78.72±0.45	88.71±0.12	75.92±0.89
		TRaDy	91.34±0.14	69.83±0.46	57.62±0.26	82.13±0.34	64.30±0.21	78.73±0.48	88.86±0.21	76.12±0.86
	72 960	SU	91.88±0.27	70.34±0.19	58.33±0.36	83.15±0.28	66.49±0.29	78.99±0.74	87.82±0.12	76.71±0.98
		D-Full Random	91.97±0.36	71.04±0.11	58.22±0.43	81.63±0.78	65.62±0.36	79.00±0.28	88.86±0.25	76.62±1.10
		D-Det RGN	91.92±0.26	70.67±0.16	58.72±0.23	82.51±0.55	66.83±0.03	79.20±0.6	88.08±0.18	76.85±0.92
		TRaDy	92.27±0.36	71.39±0.27	58.80±0.47	82.39±0.18	67.17±0.10	79.10±0.14	88.31±0.23	77.06±0.73
	101 376	SU	92.42±0.16	71.32±0.12	58.52±0.25	83.24±0.25	67.18±0.09	79.03±0.24	87.92±0.17	77.09±0.51
		D-Full Random	92.21±0.01	71.54±0.21	58.86±0.42	82.41±0.16	66.69±0.02	78.80±0.39	89.04±0.39	77.08±0.74
		D-Det RGN	92.37±0.09	71.06±0.02	59.27±0.61	82.73±0.51	67.97±0.19	79.06±0.63	88.1±0.03	77.22±1.04
		TRaDy	92.50±0.24	72.18±0.33	59.34±0.25	82.80±0.45	68.05±0.21	79.29±0.28	88.06±0.24	77.46±0.78
	1 162 032	Baseline	93.71±0.12	74.81±0.13	61.75±0.12	84.44±0.50	72.98±0.09	78.53±0.10	88.95±0.04	79.31±0.56
SwinT	27 946	D-Full Random	96.34±0.09	82.77±0.10	71.83±4.14	88.64±0.36	80.66±0.09	90.76±0.33	93.79±0.07	86.40±4.17
		D-Det RGN	96.60±0.04	83.18±0.04	74.56±0.31	88.52±0.28	81.25±0.10	90.91±0.29	93.25±0.17	86.90±0.55
		TRaDy	96.30±0.06	82.85±0.16	74.40±0.13	88.61±0.51	80.75±0.05	91.15±0.20	93.73±0.09	86.83±0.60
	112 640	D-Full Random	96.59±0.20	83.44±0.09	72.76±0.32	82.26±6.56	80.88±0.45	90.61±0.8	93.92±0.10	85.78±6.64
		D-Det RGN	96.82±0.07	83.77±0.05	74.67±0.40	89.51±0.04	82.43±0.09	90.78±0.11	92.96±0.14	87.28±0.46
		TRaDy	96.74±0.07	83.55±0.13	74.30±0.14	88.60±0.44	81.56±0.10	91.11±0.24	93.83±0.02	87.10±0.55
	633 859	D-Full Random	97.06±0.12	84.65±0.24	75.11±0.39	89.10±0.17	83.59±0.12	90.95±0.22	93.25±0.04	87.67±0.56
		D-Det RGN	97.37±0.08	85.11±0.09	75.20±0.08	90.45±0.51	83.94±0.05	91.39±0.07	93.32±0.24	88.11±0.59
		TRaDy	97.25±0.03	84.65±0.24	75.11±0.39	89.10±0.17	83.59±0.12	90.95±0.22	93.25±0.04	87.70±0.55
	2 767 686	D-Full Random	97.40±0.07	85.77±0.09	75.89±0.29	90.00±0.49	84.76±0.13	91.55±0.38	93.74±0.17	88.44±0.73
		D-Det RGN	97.64±0.06	85.88±0.12	76.26±0.42	91.46±0.52	84.95±0.05	91.20±0.20	93.88±0.17	88.75±0.73
		TRaDy	97.62±0.09	85.77±0.09	75.89±0.29	90.00±0.49	84.76±0.13	91.55±0.38	93.74±0.17	88.48±0.73
	31 889 952	Baseline	97.78±0.16	86.30±0.05	74.89±0.20	90.57±0.43	86.07±0.23	90.18±0.60	93.72±0.10	88.50±0.31

Table A.3: Comparison of final top1 test accuracies between static channel selection strategies over various pretrained vision models, datasets, and budgets. Credits to Quélennec et al. [2025a].

Model	B_{mem}	Method	CIFAR-10	CIFAR-100	CUB	Flowers	Food	Pets	VWW	Average
MbV2-w0.35	27 946	S-Full Random	87.79±0.21	66.52±0.07	55.6±0.55	79.15±0.43	58.21±0.21	76.46±0.04	88.17±0.03	73.13±0.76
		S-Det RGN	88.78±0.07	67.25±0.12	57.11±0.49	79.59±0.41	60.05±0.11	76.70±0.26	88.44±0.15	73.99±0.73
		S-Top$K$ Random	88.52±0.02	66.93±0.14	56.52±0.55	79.68±0.62	59.52±0.39	76.71±0.55	88.12±0.58	73.71±1.22
	66 592	S-Full Random	88.62±0.12	66.81±0.12	56.75±0.09	79.28±0.63	59.75±0.78	76.50±0.51	87.87±0.09	73.65±1.14
		S-Det RGN	89.88±0.06	68.14±0.20	57.84±0.17	80.31±0.25	62.70±0.13	76.61±0.54	88.11±0.23	74.80±0.70
		S-Top$K$ Random	89.76±0.27	68.10±0.18	57.49±0.36	80.39±0.46	62.08±0.29	76.64±0.24	87.75±0.13	74.60±0.78
	93 696	S-Full Random	89.01±0.22	67.21±0.15	56.87±0.51	79.78±0.79	60.40±0.56	76.58±0.35	88.22±0.48	74.01±1.27
		S-Det RGN	90.45±0.10	68.84±0.04	57.80±0.09	80.59±0.11	64.11±0.19	76.75±0.26	87.54±0.06	75.15±0.37
		S-Top$K$ Random	90.25±0.05	68.41±0.36	57.94±0.18	80.42±0.33	63.14±0.18	76.67±0.22	87.61±0.15	74.92±0.61
MCUNet-in1	15 936	S-Full Random	88.78±0.17	67.78±0.35	60.14±0.18	82.20±0.45	62.68±0.38	81.09±0.19	89.50±0.23	76.02±0.79
		S-Det RGN	89.12±0.14	67.97±0.10	60.06±0.22	82.14±0.37	63.77±0.15	80.79±0.34	89.55±0.01	76.20±0.59
		S-Top$K$ Random	89.08±0.11	67.86±0.30	60.26±0.17	82.34±0.82	63.60±0.20	81.09±0.30	89.56±0.11	76.26±0.97
	64 832	S-Full Random	90.02±0.52	69.70±0.02	60.98±0.17	82.67±0.20	64.87±1.29	80.95±0.53	89.44±0.06	76.95±1.51
		S-Det RGN	89.97±0.18	67.78±0.19	61.80±0.24	80.86±0.96	65.16±0.24	81.76±0.56	89.27±0.12	76.66±1.20
		S-Top$K$ Random	90.82±0.27	70.34±0.15	60.90±0.14	82.92±0.54	67.29±0.28	81.34±0.29	89.11±0.06	77.53±0.76
	112 640	S-Full Random	90.82±0.13	70.75±0.41	61.22±0.11	82.77±0.35	66.92±0.77	80.80±0.15	89.01±0.12	77.47±0.97
		S-Det RGN	91.28±0.13	71.53±0.20	61.02±0.16	82.69±0.58	69.17±0.23	80.64±0.16	88.89±0.17	77.89±0.73
		S-Top$K$ Random	91.58±0.13	71.55±0.34	60.95±0.73	82.80±0.26	69.28±0.17	80.42±0.29	88.73±0.33	77.90±0.98
Proxyless-w0.3	25 984	S-Full Random	89.15±0.33	67.90±0.21	55.22±0.23	81.64±0.54	58.72±0.26	78.32±0.14	88.39±0.08	74.19±0.77
		S-Det RGN	90.19±0.27	68.50±0.20	57.13±0.25	81.89±0.37	61.69±0.19	78.90±0.14	88.51±0.10	75.26±0.61
		S-Top$K$ Random	89.98±0.18	68.33±0.22	56.17±0.11	81.89±0.50	60.60±0.12	78.17±0.25	88.39±0.19	74.79±0.68
	72 960	S-Full Random	90.34±0.08	68.78±0.14	56.35±0.38	81.95±0.36	61.32±0.87	78.79±0.48	88.40±0.22	75.13±1.16
		S-Det RGN	91.30±0.12	70.38±0.15	58.26±0.46	82.62±0.46	65.09±0.01	78.67±0.29	87.94±0.54	76.32±0.91
		S-Top$K$ Random	91.09±0.14	70.10±0.32	57.32±0.45	82.15±0.10	63.75±0.16	78.4±0.32	87.86±0.40	75.81±0.79
	101 376	S-Full Random	90.64±0.15	69.37±0.14	57.18±0.63	82.15±0.33	62.60±0.63	78.51±0.20	88.06±0.32	75.50±1.04
		S-Det RGN	91.76±0.15	71.28±0.35	58.6±0.18	82.82±0.42	66.45±0.25	78.70±0.49	87.84±0.27	76.78±0.85
		S-Top$K$ Random	91.61±0.33	70.73±0.46	57.88±0.35	82.51±0.25	64.92±0.17	78.46±0.06	87.58±0.27	76.24±0.78
SwinT	27 946	S-Full Random	96.31±0.15	82.94±0.07	73.80±0.11	88.34±0.19	80.55±0.21	91.08±0.15	93.63±0.08	86.66±0.39
		S-Det RGN	96.36±0.09	83.05±0.12	74.38±0.12	88.76±0.29	80.62±0.11	91.03±0.11	93.60±0.15	86.83±0.41
		S-Top$K$ Random	96.28±0.07	82.97±0.09	74.03±0.07	88.58±0.11	80.58±0.25	90.87±0.21	93.70±0.04	86.72±0.37
	112 640	S-Full Random	96.54±0.10	83.25±0.36	74.00±0.23	88.74±0.09	81.09±0.07	91.21±0.11	93.64±0.11	86.92±0.48
		S-Det RGN	96.70±0.06	83.59±0.16	74.68±0.20	89.36±0.32	81.97±0.09	90.97±0.24	92.99±0.06	87.18±0.49
		S-Top$K$ Random	96.60±0.11	83.29±0.20	74.19±0.34	88.83±0.16	81.19±0.16	90.99±0.03	93.50±0.17	86.94±0.50
	633 859	S-Full Random	96.99±0.12	84.24±0.20	74.55±0.44	89.33±0.05	82.99±0.12	91.26±0.07	93.1±0.09	87.49±0.53
		S-Det RGN	97.26±0.05	84.78±0.13	75.69±0.23	90.29±0.18	83.72±0.18	91.28±0.28	79.72±23.27	86.11±23.27
		S-Top$K$ Random	97.06±0.06	84.24±0.20	74.55±0.44	89.33±0.05	82.99±0.12	91.26±0.07	93.10±0.09	87.50±0.52
	2 767 686	S-Full Random	97.50±0.06	85.53±0.08	75.75±0.14	89.84±0.08	84.46±0.12	91.30±0.23	93.73±0.17	88.30±0.36
		S-Det RGN	97.50±0.06	85.53±0.08	75.75±0.14	89.84±0.08	84.46±0.12	91.30±0.23	93.73±0.17	88.30±0.36
		S-Top$K$ Random	97.51±0.11	85.53±0.08	75.75±0.14	89.84±0.08	84.46±0.12	91.30±0.23	93.73±0.17	88.30±0.38

Table A.4: Comparison of final top1 test accuracies between dynamic channel selection strategies with pretrained BERT and RoBERTa models, fine-tuned on various datasets and budgets. Credits to Quélennec et al. [2025a].

Model	B_{mem}	**Method**	**QNLI**	**RTE**	**SST2**	**Average**
BERT	27 946	D-Full Random	84.50±0.23	56.32±0.36	89.41±0.46	76.74±0.63
		D-Det RGN	87.78±0.45	57.28±1.78	91.17±0.40	78.74±1.88
		TRaDy	84.38±0.21	57.76±0.96	89.53±0.13	77.22±0.57
	112 640	D-Full Random	84.50±0.04	58.24±2.32	89.41±0.53	77.38±2.38
		D-Det RGN	89.00±0.22	60.05±2.66	91.25±0.57	80.10±2.73
		TRaDy	84.56±0.28	57.88±0.91	89.60±0.26	77.35±0.57
	1 912 629	D-Full Random	85.85±0.47	54.99±0.91	89.76±0.48	76.87±1.13
		D-Det RGN	89.83±0.09	60.53±0.55	91.48±0.24	80.61±0.61
		TRaDy	85.84±0.30	56.68±0.72	90.10±0.35	77.54±0.49
	8 351 308	D-Full Random	88.68±0.14	58.24±1.46	89.60±0.46	78.84±1.54
		D-Det RGN	90.47±0.16	60.17±3.07	91.67±0.52	80.77±3.12
		TRaDy	88.97±0.20	57.16±1.50	90.86±0.18	79.00±0.88
	96 225 792	Baseline	90.81±0.27	62.45±1.81	91.74±0.5	81.67±1.90
RoBERTa	27 946	D-Full Random	89.69±0.04	57.40±0.72	93.23±0.34	80.11±0.80
		D-Det RGN	90.97±0.22	76.29±0.55	92.51±0.40	86.59±0.71
		TRaDy	89.71±0.13	57.16±0.75	93.31±0.07	80.06±0.76
	112 640	D-Full Random	89.99±0.26	58.12±1.25	93.12±0.20	80.41±1.29
		D-Det RGN	90.78±0.23	77.02±0.21	93.00±0.11	86.93±0.33
		TRaDy	90.05±0.12	59.57±2.87	93.31±0.13	80.98±1.66
	1 912 629	D-Full Random	91.23±0.18	65.10±0.83	93.85±0.65	83.39±1.07
		D-Det RGN	91.23±0.26	75.09±2.25	93.04±0.26	86.45±2.28
		TRaDy	91.23±0.26	68.83±1.78	93.43±1.16	84.5±1.24
	8 351 308	D-Full Random	91.54±0.11	73.77±2.92	93.27±0.13	86.19±2.92
		D-Det RGN	91.90±0.13	70.28±15.25	93.58±0.11	85.25±15.25
		TRaDy	91.36±0.23	73.53±1.78	93.31±0.35	86.07±1.83
	96 225 792	Baseline	92.31±0.14	76.41±0.55	93.16±0.92	87.29±1.08

Table A.5: Comparison of final top1 test accuracies between static channel selection strategies with pretrained BERT and RoBERTa models, fine-tuned on various datasets and budgets. Credits to Quélennec et al. [2025a].

Model	B_{mem}	Method	QNLI	RTE	SST2	Average
BERT	27 946	S-Full Random	84.48±0.22	57.28±0.83	89.68±0.11	77.15±0.87
		S-Det RGN	86.42±0.13	58.72±2.18	90.86±0.35	78.67±2.21
		S-Top$K$ Random	84.53±0.34	58.24±2.32	89.37±0.13	77.38±2.35
	112 640	S-Full Random	84.51±0.08	58.72±2.40	89.72±0.48	77.65±2.45
		S-Det RGN	88.30±0.43	59.09±1.46	91.21±0.46	79.53±1.59
		S-Top$K$ Random	84.69±0.19	58.00±2.21	89.53±0.26	77.41±2.23
	1 912 629	S-Full Random	86.08±0.49	57.04±1.44	89.91±0.34	77.68±1.56
		S-Det RGN	89.22±0.16	59.81±2.05	91.55±0.29	80.19±2.08
		S-Top$K$ Random	86.80±0.43	58.60±2.05	90.29±0.07	78.56±2.10
	8 351 308	S-Full Random	88.55±0.17	56.80±0.21	90.29±0.18	78.55±0.32
		S-Det RGN	89.87±0.22	61.01±2.53	91.55±0.29	80.81±2.57
		S-Top$K$ Random	88.77±0.69	57.76±0.72	91.28±0.57	79.27±1.15
RoBERTa	27 946	S-Full Random	89.68±0.17	56.56±0.75	93.31±0.18	79.85±0.79
		S-Det RGN	90.81±0.12	76.90±0.72	93.43±0.35	87.04±0.81
		S-Top$K$ Random	89.66±0.07	56.68±0.72	93.31±0.07	79.88±0.73
	112 640	S-Full Random	89.69±0.10	59.09±3.62	93.39±0.29	80.72±3.63
		S-Det RGN	90.91±0.61	76.77±2.61	92.85±0.35	86.85±2.70
		S-Top$K$ Random	89.60±0.02	56.80±0.55	93.27±0.18	79.89±0.58
	1 912 629	S-Full Random	90.95±0.26	62.33±9.49	93.58±0.34	82.29±9.50
		S-Det RGN	91.10±0.24	77.26±1.65	92.89±0.34	87.08±1.70
		S-Top$K$ Random	91.14±0.22	62.33±6.14	93.23±0.46	82.23±6.16
	8 351 308	S-Full Random	91.28±0.26	72.80±0.75	92.78±0.00	85.62±0.79
		S-Det RGN	91.20±0.31	75.09±0.00	93.20±0.66	86.49±0.73
		S-Top$K$ Random	91.09±0.17	71.84±3.21	93.08±0.26	85.34±3.22

Table A.6: Comparison of final top1 test accuracies between channel selection strategies over various pretrained convolutional architectures, datasets, and budgets. Credits to Quélennec et al. [2025b].

Model	B_{mem}	Method	CIFAR-10	CIFAR-100	CUB	Flowers	Food	Pets	VWW	Average
MbV2-w0.35	27 946	Prev MeDyate	90.18±0.13	68.84±0.20	58.19±0.64	80.07±0.42	63.32±0.04	77.02±0.26	88.66±0.11	75.18±0.85
		LaRa TraDy	89.97±0.06	68.67±0.23	58.67±0.26	80.34±0.60	63.60±0.12	77.72±0.24	88.48±0.25	75.35±0.79
		LaRa Det Raw	89.89±0.03	68.17±0.20	58.0±0.19	80.65±0.35	62.61±0.01	77.60±0.45	88.10±0.29	75.00±0.70
		LaRa Prob Raw	90.13±0.17	68.75±0.23	58.20±0.36	80.25±0.42	63.93±0.07	77.28±0.20	88.58±0.31	75.30±0.73
		LaRa MeDyate	90.24±0.11	68.83±0.13	58.73±0.12	80.31±0.15	63.7±0.16	77.45±0.25	88.53±0.11	75.40±0.41
	66 592	Prev MeDyate	90.70±0.05	69.66±0.17	58.96±0.33	80.80±0.34	65.78±0.11	77.16±0.70	87.65±0.14	75.82±0.88
		LaRa TraDy	90.82±0.18	69.65±0.16	58.50±0.23	80.27±0.29	65.08±0.22	76.90±0.20	88.06±0.22	75.61±0.58
		LaRa Det Raw	90.57±0.12	69.22±0.16	58.62±0.08	81.10±0.62	64.58±0.26	77.24±0.38	87.98±0.05	75.62±0.80
		LaRa Prob Raw	91.02±0.09	69.59±0.28	58.87±0.24	80.68±0.51	65.38±0.25	77.00±0.59	87.72±0.24	75.75±0.93
		LaRa MeDyate	91.02±0.18	69.70±0.31	58.90±0.10	80.71±0.59	65.37±0.08	76.92±0.10	88.01±0.29	75.80±0.77
	93 696	Prev MeDyate	91.27±0.11	70.23±0.31	58.86±0.54	81.27±0.43	66.72±0.19	77.11±0.33	88.02±0.18	76.21±0.87
		LaRa TraDy	91.37±0.06	70.09±0.21	59.03±0.32	80.66±0.32	66.39±0.06	76.86±0.24	87.96±0.21	76.05±0.60
		LaRa Det Raw	90.91±0.04	69.73±0.08	58.88±0.24	81.48±0.35	65.53±0.12	77.24±0.46	87.93±0.26	75.96±0.69
		LaRa Prob Raw	91.44±0.10	70.23±0.15	59.45±0.03	81.05±0.39	66.83±0.04	77.13±0.14	87.87±0.19	76.29±0.49
		LaRa MeDyate	91.35±0.15	70.21±0.05	59.08±0.36	80.94±0.40	66.99±0.14	77.04±0.08	87.9±0.36	76.22±0.69
	1 252 320	Baseline	92.72±0.03	72.69±0.16	60.03±0.18	81.88±0.34	70.79±0.20	76.68±0.33	88.58±0.19	77.62±0.60
MCUNet-in1	15 936	Prev MeDyate	90.63±0.10	69.94±0.12	61.70±0.05	82.51±0.22	66.94±0.23	81.21±0.04	89.69±0.33	77.52±0.49
		LaRa TraDy	90.89±0.06	69.88±0.13	62.23±0.07	83.00±0.41	67.34±0.15	81.62±0.24	89.72±0.24	77.81±0.58
		LaRa Det Raw	90.28±0.14	69.66±0.28	61.83±0.63	82.65±0.40	66.29±0.19	80.85±0.34	89.67±0.15	77.32±0.91
		LaRa Prob Raw	90.91±0.18	69.74±0.20	61.75±0.39	82.61±0.19	67.33±0.26	80.98±0.26	89.55±0.21	77.55±0.66
		LaRa MeDyate	90.89±0.10	69.89±0.05	62.39±0.41	82.76±0.43	67.39±0.12	81.37±0.24	89.94±0.04	77.80±0.66
	64 832	Prev MeDyate	92.33±0.12	72.41±0.25	62.40±0.50	83.29±0.63	71.20±0.28	81.44±0.37	89.22±0.31	78.90±1.02
		LaRa TraDy	92.09±0.02	72.24±0.38	62.60±0.34	83.07±0.53	70.44±0.03	81.46±0.44	89.09±0.24	78.71±0.89
		LaRa Det Raw	91.94±0.30	71.57±0.11	62.29±0.50	82.71±0.40	69.98±0.16	80.96±0.09	89.19±0.15	78.38±0.75
		LaRa Prob Raw	92.14±0.10	72.26±0.18	62.51±0.29	82.93±0.42	70.79±0.10	81.59±0.45	89.32±0.29	78.79±0.77
		LaRa MeDyate	92.23±0.18	72.00±0.13	62.44±0.09	83.24±0.29	71.00±0.19	81.20±0.03	89.18±0.23	78.76±0.48
	112 640	Prev MeDyate	92.78±0.09	72.88±0.36	61.83±0.49	83.48±0.44	72.42±0.29	81.16±0.48	89.14±0.49	79.10±1.06
		LaRa TraDy	92.78±0.10	73.71±0.03	61.85±0.54	83.24±0.26	72.20±0.14	81.17±0.55	89.14±0.09	79.16±0.84
		LaRa Det Raw	92.51±0.19	72.88±0.20	62.14±0.12	83.16±0.34	71.68±0.11	80.78±0.28	89.59±0.29	78.96±0.62
		LaRa Prob Raw	92.78±0.23	73.46±0.21	62.06±0.30	82.96±0.23	72.55±0.11	81.15±0.19	89.50±0.06	79.21±0.54
		LaRa MeDyate	92.72±0.20	73.55±0.28	62.36±0.34	83.46±0.42	72.73±0.02	80.83±0.50	89.51±0.13	79.31±0.82
	1 309 808	Baseline	93.87±0.10	76.03±0.18	61.62±0.62	83.45±0.42	75.74±0.14	79.49±0.60	90.06±0.16	80.04±1.00
Proxyless-w0.3	25 984	Prev MeDyate	91.43±0.20	70.30±0.08	57.74±0.38	82.15±0.22	64.80±0.24	78.77±0.05	88.57±0.13	76.25±0.56
		LaRa TraDy	91.46±0.17	69.38±0.15	58.18±0.30	81.76±0.25	64.58±0.05	79.10±0.22	88.57±0.05	76.15±0.51
		LaRa Det Raw	91.26±0.11	68.59±0.11	57.87±0.10	82.11±0.31	63.40±0.07	78.91±0.16	88.65±0.18	75.83±0.44
		LaRa Prob Raw	91.53±0.24	69.39±0.12	58.14±0.28	81.89±0.40	64.66±0.24	79.27±0.52	88.74±0.17	76.23±0.82
		LaRa MeDyate	91.38±0.21	69.71±0.10	58.27±0.06	81.96±0.46	64.75±0.27	78.92±0.24	88.77±0.20	76.25±0.66
	72 960	Prev MeDyate	92.27±0.13	71.76±0.13	59.06±0.29	82.59±0.20	67.81±0.08	79.28±0.35	88.46±0.02	77.32±0.54
		LaRa TraDy	92.29±0.24	71.26±0.17	58.73±0.24	82.50±0.33	67.14±0.13	79.27±0.22	88.44±0.26	77.09±0.62
		LaRa Det Raw	92.32±0.07	71.32±0.03	58.87±0.20	83.09±0.28	67.20±0.35	78.49±0.20	88.02±0.24	77.04±0.59
		LaRa Prob Raw	92.48±0.04	71.83±0.33	59.75±0.67	82.89±0.46	67.85±0.19	79.31±0.47	88.02±0.18	77.45±1.03
		LaRa MeDyate	92.50±0.15	71.54±0.17	59.25±0.79	82.90±0.12	67.66±0.17	79.05±0.32	88.05±0.14	77.28±0.92
	101 376	Prev MeDyate	92.65±0.25	72.28±0.26	59.58±0.20	83.04±0.30	68.77±0.21	79.23±0.22	88.35±0.17	77.70±0.62
		LaRa TraDy	92.56±0.09	71.76±0.29	59.58±0.48	82.6±0.20	67.79±0.16	79.05±0.18	88.3±0.20	77.38±0.68
		LaRa Det Raw	92.37±0.19	71.36±0.31	59.34±0.61	83.25±0.46	67.35±0.07	79.00±0.45	88.30±0.27	77.28±1.00
		LaRa Prob Raw	92.61±0.15	71.62±0.12	59.91±0.26	82.99±0.28	68.32±0.14	79.31±0.36	88.47±0.45	77.60±0.73
		LaRa MeDyate	92.42±0.02	71.86±0.09	59.73±0.28	82.9±0.23	68.52±0.22	79.45±0.50	88.23±0.14	77.59±0.68
	1 162 032	Baseline	93.71±0.12	74.81±0.13	61.75±0.12	84.44±0.50	72.98±0.09	78.53±0.10	88.95±0.04	79.31±0.56

Table A.7: Comparison of final top1 test accuracies between channel selection strategies over various datasets, and budgets when considering a SwinT architecture. Credits to Quélennec et al. [2025b].

Model	B_{mem}	Method	CIFAR-10	CIFAR-100	CUB	Flowers	Food	Pets	VWW	Average
SwinT	27 946	Prev MeDyate	96.35±0.11	82.91±0.10	73.98±0.06	88.44±0.34	80.74±0.04	90.97±0.20	93.75±0.15	86.73±0.45
		LaRa TraDy	96.30±0.07	83.16±0.17	74.31±0.23	88.59±0.24	80.78±0.05	90.91±0.12	92.77±0.14	86.69±0.43
		LaRa Det Raw	96.34±0.06	82.96±0.10	74.42±0.10	88.21±0.14	80.76±0.11	90.98±0.11	92.93±0.14	86.66±0.29
		LaRa Prob Raw	96.33±0.04	83.14±0.18	74.35±0.02	88.56±0.34	80.90±0.17	90.87±0.29	92.81±0.06	86.71±0.52
		LaRa MeDyate	96.29±0.09	83.10±0.06	74.34±0.01	88.63±0.51	80.97±0.05	90.99±0.19	92.76±0.07	86.73±0.56
	112 640	Prev MeDyate	96.76±0.05	83.58±0.16	74.43±0.15	88.85±0.23	81.58±0.04	90.99±0.19	93.55±0.38	87.11±0.53
		LaRa TraDy	96.94±0.14	83.88±0.17	73.24±0.20	85.60±0.64	81.85±0.07	90.61±0.26	92.75±0.11	86.41±0.76
		LaRa Det Raw	96.69±0.12	83.49±0.12	73.69±0.32	86.46±0.15	81.30±0.10	91.12±0.27	93.19±0.14	86.56±0.51
		LaRa Prob Raw	96.79±0.15	83.71±0.13	73.89±0.06	86.15±0.28	81.59±0.16	91.15±0.19	93.04±0.15	86.62±0.45
		LaRa MeDyate	96.87±0.03	83.82±0.15	73.85±0.17	86.52±0.33	81.79±0.08	91.01±0.12	93.01±0.09	86.70±0.44
	633 859	Prev MeDyate	97.32±0.03	84.72±0.18	75.14±0.57	89.79±0.36	83.51±0.11	91.02±0.13	93.33±0.27	87.83±0.77
		LaRa TraDy	97.31±0.04	84.90±0.08	74.36±0.27	86.56±0.89	83.63±0.12	91.08±0.17	92.72±0.13	87.22±0.97
		LaRa Det Raw	97.19±0.03	84.55±0.13	74.25±0.15	87.67±0.54	83.20±0.03	91.13±0.27	92.62±0.28	87.23±0.70
		LaRa Prob Raw	97.32±0.03	84.71±0.09	74.30±0.08	87.53±0.19	83.69±0.05	91.18±0.15	92.75±0.34	87.35±0.44
		LaRa MeDyate	97.25±0.12	85.01±0.10	74.67±0.55	87.56±0.31	83.66±0.13	91.21±0.06	92.72±0.08	87.44±0.67
	2 767 686	Prev MeDyate	97.70±0.08	85.91±0.13	76.13±0.29	90.73±0.34	84.87±0.08	91.38±0.33	93.77±0.12	88.64±0.59
		LaRa TraDy	97.52±0.07	85.81±0.12	75.22±0.06	88.21±0.46	84.70±0.02	91.23±0.18	93.33±0.07	88.00±0.52
		LaRa Det Raw	97.54±0.02	85.69±0.21	75.42±0.57	90.26±0.22	84.83±0.06	91.04±0.37	93.52±0.22	88.33±0.78
		LaRa Prob Raw	97.71±0.05	85.79±0.30	75.78±0.58	89.48±0.41	84.88±0.10	91.12±0.65	93.57±0.05	88.33±1.02
		LaRa MeDyate	97.67±0.13	85.82±0.18	75.71±0.10	89.24±0.46	84.84±0.12	91.18±0.22	79.9±23.43	86.34±23.44
	31 889 952	Baseline	97.78±0.16	86.30±0.05	74.89±0.20	90.57±0.43	86.07±0.23	90.18±0.60	93.72±0.10	88.50±0.31

Table A.8: Comparison of final top1 test accuracies between channel selection strategies with pretrained BERT and RoBERTa models, fine-tuned on various datasets and budgets. Credits to Quélennec et al. [2025b].

Model	B_{mem}	Method	QNLI	RTE	SST2	Average
BERT	27 946	Prev MeDyate	84.38±0.06	58.24±1.82	89.37±0.13	77.33±1.83
		LaRa TraDy	84.92±0.05	57.76±2.60	89.53±0.13	77.40±2.60
		LaRa Det Raw	84.75±0.13	56.92±0.75	88.91±0.35	76.86±0.84
		LaRa Prob Raw	84.62±0.30	57.04±1.30	88.91±0.18	76.86±1.35
		LaRa MeDyate	84.58±0.07	58.48±1.65	89.18±0.24	77.41±1.67
	112 640	Prev MeDyate	84.51±0.25	58.48±0.63	89.53±0.33	77.51±0.75
		LaRa TraDy	85.16±0.19	59.09±2.18	89.68±0.11	77.98±2.19
		LaRa Det Raw	85.69±0.04	58.24±1.16	89.11±0.46	77.68±1.25
		LaRa Prob Raw	85.28±0.11	57.52±0.55	89.56±0.00	77.45±0.56
		LaRa MeDyate	85.19±0.21	56.56±0.21	89.41±0.37	77.05±0.47
	1 912 629	Prev MeDyate	86.78±0.43	55.72±1.63	90.37±0.34	77.62±1.72
		LaRa TraDy	87.43±0.30	56.68±0.36	90.56±0.48	78.22±0.67
		LaRa Det Raw	88.36±0.16	58.12±1.65	90.83±0.23	79.10±1.67
		LaRa Prob Raw	88.31±0.10	58.12±2.25	90.63±0.66	79.02±2.35
		LaRa MeDyate	87.79±0.10	55.84±1.04	90.44±0.76	78.02±1.29
	8 351 308	Prev MeDyate	89.05±0.14	61.13±1.99	91.21±0.66	80.46±2.10
		LaRa TraDy	89.13±0.32	57.76±1.88	90.75±0.18	79.21±1.92
		LaRa Det Raw	89.96±0.12	63.06±0.83	91.74±0.72	81.59±1.11
		LaRa Prob Raw	89.69±0.14	61.01±0.96	91.48±0.13	80.73±0.98
		LaRa MeDyate	89.95±0.18	61.49±1.71	90.90±0.53	80.78±1.80
	96 225 792	Baseline	90.81±0.27	62.45±1.81	91.74±0.50	81.67±1.90
RoBERTa	27 946	Prev MeDyate	89.69±0.06	57.04±0.72	93.31±0.07	80.01±0.73
		LaRa TraDy	89.57±0.29	59.33±1.82	93.35±0.11	80.75±1.85
		LaRa Det Raw	89.02±1.15	68.23±0.36	93.00±0.30	83.42±1.24
		LaRa Prob Raw	89.42±0.22	66.91±3.47	93.08±0.54	83.14±3.52
		LaRa MeDyate	89.81±0.27	63.18±4.72	93.31±0.18	82.10±4.73
	112 640	Prev MeDyate	90.05±0.09	60.41±2.29	93.39±0.18	81.28±2.30
		LaRa TraDy	89.65±0.53	62.33±2.05	93.46±0.40	81.81±2.15
		LaRa Det Raw	89.62±0.10	66.55±2.21	92.85±0.07	83.01±2.21
		LaRa Prob Raw	89.66±0.15	66.55±2.35	92.89±0.34	83.03±2.38
		LaRa MeDyate	89.77±0.33	67.75±3.28	93.27±0.13	83.60±3.30
	1 912 629	Prev MeDyate	91.40±0.06	75.45±0.72	93.92±0.53	86.92±0.90
		LaRa TraDy	90.82±0.25	69.68±0.96	93.16±0.33	84.55±1.05
		LaRa Det Raw	90.85±0.11	73.04±5.01	92.09±0.11	85.33±5.01
		LaRa Prob Raw	90.87±0.19	75.21±1.46	92.39±0.46	86.16±1.54
		LaRa MeDyate	90.60±0.47	74.61±1.85	92.97±0.07	86.06±1.91
	8 351 308	Prev MeDyate	91.52±0.41	75.21±1.63	93.16±0.46	86.63±1.74
		LaRa TraDy	90.87±0.21	74.13±2.92	93.23±0.40	86.08±2.95
		LaRa Det Raw	91.21±0.25	74.73±1.08	92.66±0.61	86.20±1.27
		LaRa Prob Raw	90.67±0.24	74.37±0.72	92.93±0.07	85.99±0.76
		LaRa MeDyate	90.49±0.78	70.52±3.51	92.97±0.40	84.66±3.62
	96 225 792	Baseline	92.31±0.14	76.41±0.55	93.16±0.92	87.29±1.08

Table A.9: Comparison of accuracy and compression performance between tensor decomposition strategies over various pretrained models, datasets, and number of fine-tuned layers. Credits to Nguyen et al. [2024].

Model	Method	#Layers	CUB200			Flowers102			Pets			CIFAR-10			CIFAR-100		
			Acc ↑	Peak Mem (MB)↓	Mean Mem (MB)↓	Acc ↑	Peak Mem (MB)↓	Mean Mem (MB)↓	Acc ↑	Peak Mem (MB)↓	Mean Mem (MB)↓	Acc ↑	Peak Mem (MB)↓	Mean Mem (MB)↓	Acc ↑	Peak Mem (MB)↓	Mean Mem (MB)↓
MobileNetV2	Vanilla training	All	52.9	3303.67	3303.67 ± 0.00	80.7	3303.67	3303.67 ± 0.00	89.8	3303.67	3303.67 ± 0.00	95.3	3303.67	3303.67 ± 0.00	77.9	3303.67	3303.67 ± 0.00
		2	48.4	30.63	30.63 ± 0.00	80.5	30.63	30.63 ± 0.00	88.4	30.63	30.63 ± 0.00	88.1	30.63	30.63 ± 0.00	64.7	30.63	30.63 ± 0.00
		4	53.1	57.42	57.42 ± 0.00	83.0	57.42	57.42 ± 0.00	89.5	57.42	57.42 ± 0.00	89.5	57.42	57.42 ± 0.00	67.6	57.42	57.42 ± 0.00
	Gradient Filter R2	2	46.7	10.00	10.00 ± 0.00	80.9	10.00	10.00 ± 0.00	88.3	10.00	10.00 ± 0.00	87.9	10.00	10.00 ± 0.00	64.7	10.00	10.00 ± 0.00
		4	50.2	18.75	18.75 ± 0.00	82.7	18.75	18.75 ± 0.00	89.2	18.75	18.75 ± 0.00	89.3	18.75	18.75 ± 0.00	67.2	18.75	18.75 ± 0.00
	Gradient Filter R7	2	46.6	0.63	0.63 ± 0.00	81.0	0.63	0.63 ± 0.00	88.7	0.63	0.63 ± 0.00	86.1	0.63	0.63 ± 0.00	63.6	0.63	0.63 ± 0.00
		4	46.4	1.17	1.17 ± 0.00	81.4	1.17	1.17 ± 0.00	88.9	1.17	1.17 ± 0.00	86.3	1.17	1.17 ± 0.00	64.6	1.17	1.17 ± 0.00
	HOSVD ($\varepsilon = 0.8$)	2	44.2	0.14	0.10 ± 0.01	79.1	0.25	0.07 ± 0.03	88.3	0.16	0.15 ± 0.01	85.6	0.30	0.22 ± 0.03	61.3	0.16	0.15 ± 0.01
		4	48.9	0.55	0.54 ± 0.00	81.5	0.66	0.49 ± 0.03	89.2	0.73	0.71 ± 0.01	88.5	0.68	0.67 ± 0.01	66.1	0.72	0.68 ± 0.03
	SVD ($\varepsilon = 0.8$)	2	16.2	4.15	2.47 ± 0.57	23.2	3.85	2.73 ± 0.64	88.1	3.38	3.17 ± 0.15	86.4	3.96	3.08 ± 0.42	62.7	3.30	2.84 ± 0.30
		4	16.7	11.34	9.70 ± 0.53	24.8	11.63	10.46 ± 0.63	89.1	12.75	12.50 ± 0.13	89.6	13.65	12.94 ± 0.76	67.0	14.10	13.46 ± 0.89
MCUNet	Vanilla training	All	30.8	1265.96	1265.96 ± 0.00	45.6	1265.96	1265.96 ± 0.00	75.2	1265.96	1265.96 ± 0.00	91.2	1265.96	1265.96 ± 0.00	65.6	1265.96	1265.96 ± 0.00
		2	9.6	27.56	27.56 ± 0.00	39.3	27.56	27.56 ± 0.00	40.6	27.56	27.56 ± 0.00	72.3	27.56	27.56 ± 0.00	45.4	27.56	27.56 ± 0.00
		4	12.4	39.05	39.05 ± 0.00	44.1	39.05	39.05 ± 0.00	49.2	39.05	39.05 ± 0.00	83.6	39.05	39.05 ± 0.00	55.3	39.05	39.05 ± 0.00
	Gradient Filter R2	2	10.0	9.00	9.00 ± 0.00	39.1	9.00	9.00 ± 0.00	40.5	9.00	9.00 ± 0.00	71.2	9.00	9.00 ± 0.00	45.1	9.00	9.00 ± 0.00
		4	12.3	12.75	12.75 ± 0.00	42.6	12.75	12.75 ± 0.00	49.1	12.75	12.75 ± 0.00	82.5	12.75	12.75 ± 0.00	54.2	12.75	12.75 ± 0.00
	Gradient Filter R7	2	9.6	0.56	0.56 ± 0.00	38.3	0.56	0.56 ± 0.00	40.5	0.56	0.56 ± 0.00	70.9	0.56	0.56 ± 0.00	45.0	0.56	0.56 ± 0.00
		4	10.3	0.80	0.80 ± 0.00	41.6	0.80	0.80 ± 0.00	44.2	0.80	0.80 ± 0.00	78.4	0.80	0.80 ± 0.00	51.0	0.80	0.80 ± 0.00
	HOSVD ($\varepsilon = 0.8$)	2	9.1	0.13	0.12 ± 0.01	35.6	0.12	0.12 ± 0.00	18.6	0.13	0.13 ± 0.00	65.5	0.07	0.05 ± 0.01	40.8	0.05	2.19 ± 0.00
		4	8.2	0.23	0.18 ± 0.06	31.5	0.12	0.10 ± 0.01	18.8	0.14	0.09 ± 0.01	77.3	0.25	0.21 ± 0.05	49.4	0.19	7.69 ± 0.00
	SVD ($\varepsilon = 0.8$)	2	9.1	5.41	5.09 ± 0.33	36.0	4.77	4.64 ± 0.11	39.5	5.77	5.49 ± 0.28	66.9	2.11	1.65 ± 0.51	41.8	1.51	1.30 ± 0.28
		4	9.6	7.32	6.06 ± 1.63	33.9	4.53	4.13 ± 0.21	46.6	7.56	6.77 ± 0.98	80.8	7.24	6.19 ± 1.21	53.7	6.36	5.55 ± 1.08
ResNet18	Vanilla training	All	56.5	1065.75	1065.75 ± 0.00	82.4	1065.75	1065.75 ± 0.00	88.4	1065.75	1065.75 ± 0.00	95.4	1065.75	1065.75 ± 0.00	78.6	1065.75	1065.75 ± 0.00
		2	55.0	24.50	24.50 ± 0.00	83.8	24.50	24.50 ± 0.00	88.9	24.50	24.50 ± 0.00	91.1	24.50	24.50 ± 0.00	70.5	24.50	24.50 ± 0.00
		4	54.0	61.25	61.25 ± 0.00	84.5	61.25	61.25 ± 0.00	88.9	61.25	61.25 ± 0.00	92.5	61.25	61.25 ± 0.00	73.3	61.25	61.25 ± 0.00
	Gradient Filter R2	2	55.1	8.00	8.00 ± 0.00	83.7	8.00	8.00 ± 0.00	88.7	8.00	8.00 ± 0.00	90.4	8.00	8.00 ± 0.00	69.8	8.00	8.00 ± 0.00
		4	50.2	14.00	14.00 ± 0.00	83.5	14.00	14.00 ± 0.00	88.6	14.00	14.00 ± 0.00	91.5	14.00	14.00 ± 0.00	71.6	14.00	14.00 ± 0.00
	Gradient Filter R7	2	52.5	0.50	0.50 ± 0.00	82.1	0.50	0.50 ± 0.00	88.4	0.50	0.50 ± 0.00	88.9	0.50	0.50 ± 0.00	68.7	0.50	0.50 ± 0.00
		4	44.8	0.88	0.88 ± 0.00	82.3	0.88	0.88 ± 0.00	86.1	0.88	0.88 ± 0.00	90.1	0.88	0.88 ± 0.00	70.1	0.88	0.88 ± 0.00
	HOSVD ($\varepsilon = 0.8$)	2	54.2	1.64	1.12 ± 0.08	82.4	1.44	0.84 ± 0.13	89.1	1.93	1.61 ± 0.07	90.8	1.50	1.43 ± 0.06	70.2	1.16	1.12 ± 0.02
		4	53.2	3.21	2.38 ± 0.16	83.8	3.43	2.52 ± 0.22	88.4	3.84	3.43 ± 0.13	92.2	1.93	1.83 ± 0.07	71.8	1.64	1.47 ± 0.13
	SVD ($\varepsilon = 0.8$)	2	54.2	12.95	10.00 ± 0.62	82.4	12.69	9.45 ± 1.03	88.7	13.58	12.42 ± 0.33	91.0	12.80	12.69 ± 0.13	70.5	11.55	11.36 ± 0.17
		4	53.7	30.91	26.82 ± 0.91	83.9	32.19	27.15 ± 1.52	89.1	32.83	31.08 ± 0.82	92.4	25.47	25.16 ± 0.16	72.6	24.63	24.16 ± 0.38
ResNet34	Vanilla training	All	60.6	1678.25	1678.25 ± 0.00	76.9	1678.25	1678.25 ± 0.00	90.3	1678.25	1678.25 ± 0.00	96.6	1678.25	1678.25 ± 0.00	82.1	1678.25	1678.25 ± 0.00
		2	57.8	24.50	24.50 ± 0.00	83.5	24.50	24.50 ± 0.00	90.8	24.50	24.50 ± 0.00	91.0	24.50	24.50 ± 0.00	70.4	24.50	24.50 ± 0.00
		4	60.7	49.00	49.00 ± 0.00	83.1	49.00	49.00 ± 0.00	90.5	49.00	49.00 ± 0.00	92.3	49.00	49.00 ± 0.00	72.8	49.00	49.00 ± 0.00
	Gradient Filter R2	2	57.5	8.00	8.00 ± 0.00	81.1	8.00	8.00 ± 0.00	90.9	8.00	8.00 ± 0.00	90.5	8.00	8.00 ± 0.00	70.0	8.00	8.00 ± 0.00
		4	58.1	16.00	16.00 ± 0.00	82.1	16.00	16.00 ± 0.00	90.2	16.00	16.00 ± 0.00	91.6	16.00	16.00 ± 0.00	70.6	16.00	16.00 ± 0.00
	Gradient Filter R7	2	55.6	0.50	0.50 ± 0.00	81.9	0.50	0.50 ± 0.00	90.9	0.50	0.50 ± 0.00	89.8	0.50	0.50 ± 0.00	69.6	0.50	0.50 ± 0.00
		4	53.3	1.00	1.00 ± 0.00	81.1	1.00	1.00 ± 0.00	90.2	1.00	1.00 ± 0.00	90.7	1.00	1.00 ± 0.00	69.2	1.00	1.00 ± 0.00
	HOSVD ($\varepsilon = 0.8$)	2	56.1	0.60	0.38 ± 0.03	80.0	0.62	0.25 ± 0.08	90.6	0.70	0.56 ± 0.04	90.7	0.57	0.54 ± 0.02	69.9	0.44	0.41 ± 0.02
		4	58.4	1.40	0.80 ± 0.12	81.0	1.27	0.61 ± 0.18	90.5	1.78	1.34 ± 0.17	91.6	1.26	1.16 ± 0.07	70.6	1.04	0.93 ± 0.09
	SVD ($\varepsilon = 0.8$)	2	56.7	9.99	6.90 ± 0.60	80.7	9.75	5.99 ± 1.04	91.1	10.79	9.50 ± 0.40	91.0	10.46	10.28 ± 0.11	70.2	9.41	9.12 ± 0.24
		4	58.9	19.24	13.99 ± 1.41	82.7	20.37	12.82 ± 2.36	90.5	21.63	18.79 ± 1.33	92.1	20.84	20.06 ± 0.40	71.4	19.78	18.71 ± 0.78
SwinT	Vanilla training	All	79.0	3748.88	3748.88 ± 0.00	89.6	3748.88	3748.88 ± 0.00	94.2	3748.88	3748.88 ± 0.00	98.0	3748.88	3748.88 ± 0.00	87.2	3748.88	3748.88 ± 0.00
		2	61.0	73.88	73.88 ± 0.00	81.7	73.88	73.88 ± 0.00	93.1	73.88	73.88 ± 0.00	92.1	73.88	73.88 ± 0.00	74.9	73.88	73.88 ± 0.00
		4	72.1	92.25	92.25 ± 0.00	86.9	92.25	92.25 ± 0.00	93.8	92.25	92.25 ± 0.00	94.2	92.25	92.25 ± 0.00	78.2	92.25	92.25 ± 0.00
	HOSVD ($\varepsilon = 0.8$)	2	55.1	1.91	1.85 ± 0.02	74.9	2.42	2.36 ± 0.02	93.1	4.01	3.93 ± 0.04	91.6	5.52	5.41 ± 0.11	73.5	6.39	6.19 ± 0.23
		4	67.4	1.55	0.65 ± 0.17	82.5	2.58	1.21 ± 0.48	93.4	3.01	2.48 ± 0.11	93.8	6.00	5.65 ± 0.41	77.2	5.84	5.35 ± 0.73
	SVD ($\varepsilon = 0.8$)	2	55.3	35.85	35.65 ± 0.11	75.2	36.78	36.57 ± 0.09	93.1	40.75	40.36 ± 0.16	91.6	46.08	45.90 ± 0.19	73.5	47.37	47.05 ± 0.38
		4	68.5	40.23	27.01 ± 2.85	82.6	43.78	35.71 ± 3.34	93.4	45.58	44.02 ± 0.39	93.9	56.62	56.09 ± 0.70	77.3	57.38	56.31 ± 1.84

Table A.10: Additional performance evaluations on a wide variety of architectures, datasets and activation compression strategies. Credits to Nguyen et al. [2025].

Model	Method	#Layers	CUB200			Flowers102			Pets			CIFAR-10			CIFAR-100		
			Acc ↑	Mem (MB)↓	TFLOPs↓	Acc ↑	Mem (MB)↓	TFLOPs↓	Acc ↑	Mem (MB)↓	TFLOPs↓	Acc ↑	Mem (MB)↓	TFLOPs↓	Acc ↑	Mem (MB)↓	TFLOPs↓
MobileNet V2	Vanilla training	All	54.25	3303.67	1.66	80.58	3303.67	1.66	89.84	3303.67	1.66	95.08	3303.67	1.66	78.03	3303.67	1.66
		2	49.57	30.63	0.01	81.25	30.63	0.01	88.28	30.63	0.01	88.03	30.63	0.01	64.78	30.63	0.01
		4	53.73	57.42	0.11	83.15	57.42	0.11	89.53	57.42	0.11	89.45	57.42	0.11	67.57	57.42	0.11
	Gradient filtering R2	2	46.68	10.00	0.01	80.92	10.00	0.01	88.28	10.00	0.01	87.90	10.00	0.01	64.72	10.00	0.01
		4	50.20	18.75	0.11	82.70	18.75	0.11	89.22	18.75	0.11	89.25	18.75	0.11	67.22	18.75	0.11
	HOSVD ($\varepsilon = 0.8$)	2	46.27	0.27	11.88	79.02	0.30	11.88	88.28	0.21	11.88	85.65	0.26	11.88	61.17	0.16	11.88
		4	50.61	0.67	22.91	81.58	0.76	22.92	89.69	0.77	22.92	88.48	0.68	22.92	66.09	0.71	22.92
	ASI	2	46.44	0.26	0.01	80.13	0.26	0.01	88.13	0.15	0.01	85.09	0.26	0.01	61.03	0.15	0.01
		4	50.69	0.63	0.06	81.58	0.63	0.06	88.91	0.76	0.06	88.03	0.63	0.06	66.18	0.63	0.06
MCUNet	Vanilla training	All	31.94	1265.96	0.50	44.53	1265.96	0.50	74.37	1265.96	0.50	91.25	1265.96	0.50	65.11	1265.96	0.50
		2	11.46	27.56	0.04	39.29	27.56	0.04	40.78	27.56	0.04	72.20	27.56	0.04	45.49	27.56	0.04
		4	12.59	39.05	0.04	43.42	39.05	0.04	48.91	39.05	0.04	83.53	39.05	0.04	54.93	39.05	0.04
	Gradient filtering R2	2	9.96	9.00	0.04	39.06	9.00	0.04	40.47	9.00	0.04	71.18	9.00	0.04	45.07	9.00	0.04
		4	12.30	12.75	0.04	42.63	12.75	0.04	49.06	12.75	0.04	82.51	12.75	0.04	54.25	12.75	0.04
	HOSVD ($\varepsilon = 0.8$)	2	9.20	0.13	7.73	34.82	0.14	7.73	39.22	0.14	7.73	65.48	0.07	7.73	41.18	0.05	7.73
		4	8.07	0.26	9.56	31.03	0.13	9.56	43.44	0.25	9.56	77.29	0.25	9.56	49.71	0.19	9.56
	ASI	2	9.20	0.06	0.02	35.49	0.13	0.02	37.66	0.14	0.02	65.17	0.06	0.02	40.70	0.04	0.02
		4	8.16	0.21	0.02	33.04	0.11	0.02	42.81	0.17	0.02	73.93	0.21	0.02	42.34	0.19	0.02
ResNet18	Vanilla training	All	55.12	1065.75	0.58	81.81	1065.75	0.58	89.53	1065.75	0.58	95.32	1065.75	0.58	78.91	1065.75	0.58
		2	57.20	24.50	0.06	84.15	24.50	0.06	89.22	24.50	0.06	91.13	24.50	0.06	70.51	24.50	0.06
		4	55.30	61.25	0.09	84.60	61.25	0.09	88.75	61.25	0.09	92.43	61.25	0.09	73.33	61.25	0.09
	Gradient filtering R2	2	55.08	8.00	0.06	83.71	8.00	0.06	88.75	8.00	0.06	90.43	8.00	0.06	69.80	8.00	0.06
		4	50.20	14.00	0.10	83.48	14.00	0.10	88.59	14.00	0.10	91.55	14.00	0.10	71.62	14.00	0.10
	HOSVD ($\varepsilon = 0.8$)	2	56.86	1.75	6.13	82.14	1.59	6.13	88.75	2.15	6.13	90.80	1.51	6.13	70.09	1.15	6.13
		4	55.38	3.51	15.57	83.71	3.82	15.57	88.75	4.43	15.57	91.87	1.92	15.57	71.71	1.62	15.57
	ASI	2	56.77	1.51	0.04	82.59	1.51	0.04	88.75	2.01	0.04	90.77	1.24	0.04	69.70	0.90	0.04
		4	55.73	3.50	0.07	84.82	3.75	0.07	88.44	4.34	0.07	91.90	1.89	0.06	71.93	1.61	0.06
ResNet34	Vanilla training	All	60.42	1678.25	1.06	77.90	1678.25	1.06	90.47	1678.25	1.06	96.71	1678.25	1.06	81.82	1678.25	1.06
		2	59.46	24.50	0.06	83.15	24.50	0.06	90.94	24.50	0.06	91.15	24.50	0.06	70.46	24.50	0.06
		4	60.59	49.00	0.12	83.15	49.00	0.12	90.31	49.00	0.12	92.43	49.00	0.12	72.83	49.00	0.12
	Gradient filtering R2	2	57.52	8.00	0.06	81.14	8.00	0.06	90.94	8.00	0.06	90.52	8.00	0.06	70.03	8.00	0.06
		4	58.11	16.00	0.12	82.14	16.00	0.12	90.16	16.00	0.12	91.59	16.00	0.12	70.57	16.00	0.12
	HOSVD ($\varepsilon = 0.8$)	2	58.77	0.80	6.13	80.02	0.69	6.13	90.63	0.92	6.13	90.61	0.56	6.13	69.86	0.44	6.13
		4	58.77	1.70	12.25	81.70	1.48	12.25	90.16	2.16	12.26	91.53	1.25	12.25	70.68	1.02	12.25
	ASI	2	58.77	0.60	0.03	80.36	0.60	0.03	91.09	0.81	0.04	90.09	0.49	0.03	69.66	0.44	0.03
		4	58.85	1.58	0.07	81.70	1.38	0.07	91.41	2.05	0.07	91.54	1.24	0.07	70.60	1.00	0.07
SwinT	Vanilla training	All	75.87	3491.25	1.40	93.08	3491.25	1.40	94.06	3491.25	1.40	98.00	3491.25	1.40	87.48	3491.25	1.40
		2	72.31	91.88	0.11	86.27	91.88	0.11	93.75	91.88	0.11	94.23	91.88	0.11	77.93	91.88	0.11
		4	76.82	183.75	0.23	89.29	183.75	0.23	93.91	183.75	0.23	95.22	183.75	0.23	80.85	183.75	0.23
	HOSVD ($\varepsilon = 0.8$)	2	70.57	2.76	116.01	85.38	3.83	116.01	93.44	4.77	116.01	93.57	6.36	116.01	77.13	5.07	116.01
		4	76.65	4.22	232.01	89.51	6.23	232.02	94.38	7.99	232.01	94.99	11.68	232.02	80.19	9.97	232.02
	ASI	2	70.66	2.45	0.08	85.71	3.54	0.09	93.44	3.54	0.09	93.83	5.96	0.09	77.11	4.88	0.09
		4	77.78	3.45	0.15	88.95	5.89	0.17	93.91	7.98	0.17	95.15	11.21	0.18	80.87	9.87	0.18

Table A.11: Experimental results for semantic segmentation. mIoU is the mean Intersection over Union, and mAcc is the micro averaged accuracy. Credits to Nguyen et al. [2025].

Method	PSPNet Zhao et al. [2017]					Method	PSPNet-M Zhao et al. [2017]				
	#Layers	mIoU ↑	mAcc ↑	Mem (MB) ↓	TFLOPs ↓		#Layers	mIoU ↑	mAcc ↑	Mem (MB) ↓	TFLOPs ↓
Vanilla training	All	54.97	68.46	920.78	0.88	Vanilla training	All	48.92	62.11	2622.49	2.96
	5	39.36	51.79	128.00	0.08		5	36.22	46.31	104.00	0.06
	10	53.17	67.18	352.00	0.62		10	45.62	58.35	604.00	1.19
Gradient Filter	5	39.34	51.59	8.00	0.08	Gradient Filter	5	35.73	45.78	6.50	0.06
	10	51.20	65.01	22.00	0.62		10	44.89	57.38	37.75	1.19
HOSVD ($\varepsilon = 0.8$)	5	38.11	49.29	0.47	177.06	HOSVD ($\varepsilon = 0.8$)	5	33.40	42.50	0.03	117.06
	10	49.23	62.39	1.40	322.25		10	40.06	51.79	1.47	744.71
ASI	5	37.90	49.11	0.32	0.05	ASI	5	32.73	41.89	0.02	0.03
	10	47.72	60.34	1.40	0.36		10	40.51	52.12	1.36	0.63
Method	**DLV3 Chen et al. [2017]**					**Method**	**DLV3-M Chen et al. [2017]**				
	#Layers	mIoU ↑	mAcc ↑	Mem (MB) ↓	TFLOPs ↓		#Layers	mIoU ↑	mAcc ↑	Mem (MB) ↓	TFLOPs ↓
Vanilla training	All	58.44	72.03	1128.02	0.97	Vanilla training	All	55.87	69.47	2758.01	3.02
	5	40.75	52.95	336.00	0.17		5	38.38	49.61	240.00	0.12
	10	55.04	69.07	560.00	0.71		10	47.91	61.67	620.00	0.71
Gradient Filter	5	32.18	42.93	27.47	0.17	Gradient Filter	5	35.70	46.71	20.71	0.12
	10	47.44	60.08	83.47	0.71		10	45.40	58.97	65.62	0.71
HOSVD ($\varepsilon = 0.8$)	5	38.52	50.14	2.66	247.63	HOSVD ($\varepsilon = 0.8$)	5	35.55	45.64	0.63	139.57
	10	50.11	63.14	1.93	392.78		10	42.68	54.17	1.04	611.29
ASI	5	38.43	50.16	2.37	0.12	ASI	5	35.76	45.93	0.62	0.07
	10	45.72	58.45	1.89	0.41		10	42.30	54.13	1.03	0.38
Method	**FCN Long et al. [2015]**					**Method**	**UPerNet Xiao et al. [2018]**				
	#Layers	mIoU ↑	mAcc ↑	Mem (MB) ↓	TFLOPs ↓		#Layers	mIoU ↑	mAcc ↑	Mem (MB) ↓	TFLOPs ↓
Vanilla training	All	45.36	59.53	952.00	0.90	Vanilla training	All	64.71	77.32	2168.78	3.55
	5	27.31	38.21	288.00	0.41		5	48.05	61.66	1380.00	3.33
	10	43.54	57.96	480.00	0.72		10	48.90	63.10	1436.00	3.35
Gradient Filter	5	27.24	38.10	18.00	0.41	Gradient Filter	5	46.79	60.50	33.00	3.33
	10	36.91	50.14	120.00	0.72		10	47.89	62.44	36.50	3.35
HOSVD ($\varepsilon = 0.8$)	5	26.34	36.14	1.43	206.12	HOSVD ($\varepsilon = 0.8$)	5	45.28	57.80	1.35	10866.86
	10	30.91	41.19	3.77	295.93		10	46.44	58.93	1.68	10881.59
ASI	5	26.51	36.26	0.99	0.23	ASI	5	42.73	54.99	1.25	1.77
	10	36.44	48.05	3.65	0.43		10	45.18	58.00	1.57	1.78

Table A.12: Performance comparison between vanilla training and ASI when fine-tuning TinyLlama 1B with BoolQ dataset. Credits to Nguyen et al. [2025].

#Layers	Vanilla training			ASI (rank=20)		
	Acc ↑	Mem (MB) ↓	TFLOPs ↓	Acc ↑	Mem (MB) ↓	TFLOPs ↓
1	65.94	1408	3.02	64.69	0.51	1.68
2	66.37	1920	6.04	64.81	0.74	3.33
3	66.91	2432	9.07	65.00	0.98	4.98
4	67.44	3840	12.09	66.31	1.49	6.66
5	67.78	4352	15.11	66.34	1.72	8.31

ECOLE DOCTORALE

Titre: Intelligence Artificielle Économe en Énergie et en Mémoire pour l'Apprentissage Embarqué

Mots clés: IA Frugale, Apprentissage Économe en Mémoire, Apprentissage Embarqué, Sélection de Sous-Réseaux, Compression d'Activations

Résumé: L'apprentissage embarqué permet aux réseaux de neurones de s'adapter continuellement sur des dispositifs en edge, offrant une meilleure confidentialité, une latence réduite et une efficacité énergétique améliorée. Cependant, les ressources mémoire et computationnelles limitées posent des défis importants, en particulier durant la backpropagation. Cette thèse aborde ces verrous à travers deux approches complémentaires : la sélection stratégique de sous-réseaux pour un fine-tuning efficace et la compression de cartes d'activation pour un apprentissage économe en mémoire.

La première ligne de travail introduit des méthodes dynamiques qui identifient de manière adaptative les composants importants du réseau à mettre à jour. Nous développons **Tra**ining **Dy**namics (TraDy), un cadre intégrant la théorie des gradients heavy-tailed pour la sélection dynamique de sous-réseaux sous contraintes mémoire strictes, et **Me**mory-constrained **Dy**namic Upd**ate** (MeDyate), une stratégie de sélection adaptative de canaux avec stochasticité et mécanismes d'échantillonnage. La validation expérimentale démontre des performances à l'état de l'art pour le fine-tuning sous contraintes mémoire.

La seconde ligne de travail traite du goulot d'étranglement mémoire des activations dans la backpropagation via la compression par décomposition tensorielle. Nous proposons une compression utilisant la High-Order Singular Value Decomposition (HOSVD) avec perte d'information contrôlée et garanties de convergence. Pour surmonter le coût computationnel de HOSVD, nous développons ASI (Activation Subspace Iteration), qui exploite la stabilité des cartes d'activation. En effectuant la sélection de rang une fois avant l'entraînement et en utilisant des itérations simples de sous-espaces avec départs à chaud, ASI atteint une réduction mémoire significative (jusqu'à 120$\times$ de compression) et une accélération (jusqu'à 91$\times$ plus rapide) tout en maintenant des performances comparables.

Les contributions théoriques incluent l'analyse formelle des dynamiques de fine-tuning, des garanties de convergence pour l'entraînement avec activations compressées, et l'analyse de complexité des méthodes de décomposition tensorielle. Une validation extensive sur diverses architectures, jeux de données, et scénarios réels incluant des implémentations sur Raspberry Pi démontre l'efficacité pratique.

Title: Energy and Memory-Efficient Artificial Intelligence for On-Device Learning

Keywords: Frugal AI, Memory-Efficient Learning, On-Device Learning, Subnetwork Selection, Activation Compression

Abstract: On-device learning enables neural networks to continuously adapt on edge devices, offering enhanced privacy, reduced latency, and improved energy efficiency. However, limited memory and computational resources pose significant challenges, particularly during backpropagation. This thesis addresses these bottlenecks through two complementary approaches: strategic subnetwork selection for efficient fine-tuning and activation map compression for memory-efficient training.

The first line of work introduces dynamic methods that adaptively identify important network components for updating. We develop **Tra**ining **Dy**namics (TraDy), a framework incorporating heavy-tailed gradient theory for dynamic subnetwork selection under strict memory constraints, and **Me**mory-constrained **Dy**namic Upd**ate** (MeDyate), an adaptive channel selection strategy with stochasticity and sampling mechanisms. Experimental validation demonstrates state-of-the-art performance in memory-constrained fine-tuning.

The second line of work addresses the activation memory bottleneck in backpropagation through tensor decomposition-based compression. We propose compression using High-Order Singular Value Decomposition (HOSVD) with controlled information loss and convergence guarantees. To overcome HOSVD's computational overhead, we develop ASI (Activation Subspace Iteration), which leverages activation map stability. By performing rank selection once before training and utilizing single subspace iterations with warm starts, ASI achieves significant memory reduction (up to 120$\times$ compression) and speedup (up to 91$\times$ faster) while maintaining comparable performance.

Theoretical contributions include formal analysis of fine-tuning dynamics, convergence guarantees for compressed activation training, and complexity analysis of tensor decomposition methods. Extensive validation across diverse architectures, datasets, and real-world scenarios including Raspberry Pi implementations demonstrates practical effectiveness.

Institut Polytechnique de Paris
91120 Palaiseau, France