EFFICIENT RESOURCE-CONSTRAINED BACKPROPAGATION THROUGH QUASI-STABLE LAYER FREEZING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent studies have shown that different layers in a neural network converge at different stages of training. As a result, updating all layers throughout the entire process is often unnecessary and leads to significant computational overhead. Motivated by the observation that each layer progressively stabilizes its representation transformation, we introduce Quasi-stable Layer Freezing (QL), which dynamically estimates layer-wise transformation stability and skips updates to quasi-stable layers. This strategy eliminates a large portion of redundant backpropagation and significantly reduces training cost. QL is designed in the context of on-device learning, where models are trained under severely constrained computational resources. Experiments on standard benchmarks demonstrate that QL reduces the computational cost of backward pass by up to approximately 96% in terms of FLOPs, and achieves speedups of $16.7\times$ on a Raspberry Pi 5. We will release the code upon paper acceptance.

1 INTRODUCTION

On-device learning has recently emerged as an important research direction in the rapid development of AI. It enables fine-tuning a pre-trained model directly on edge devices, which typically operate under strictly limited computational and memory constraints and are deployed without access to the validation set (De Lange et al., 2021). By performing learning locally, on-device learning removes major concerns related to data privacy and energy consumption, while giving users full control over their models (Dhar et al., 2021). This paradigm is especially valuable in scenarios with limited or no internet connectivity and high security requirements, such as autonomous vehicles and smartphones (Dhar et al., 2021; Hard et al., 2018).

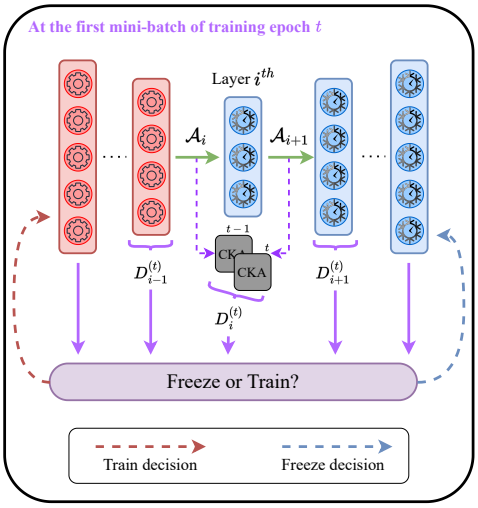


Figure 1: QL dynamically freezes layers during training to skip redundant backpropagation. Unlike prior methods, QL is task-agnostic and introduces negligible persistent memory overhead.

Despite these advantages, deploying on-device learning in practice remains challenging. The main difficulty lies in the resource-intensive nature of backpropagation (Lin et al., 2022). While backpropagation has proven highly effective in terms of optimization performance, outperforming many other methods such as Forward-Forward (Hinton, 2022) and PEPITA (Pau & Aymone, 2023), it is not yet a "resource-friendly" algorithm. Specifically, for each trainable layer, backpropagation requires storing in memory the intermediate tensors from the forward pass (Gomez et al., 2017), which are later reused to compute weight gradients during the backward pass to update the model. As model size increases, the associated memory footprint and computational cost scale accordingly, making training increasingly prohibitive on resource-constrained devices.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Although prior work has made significant progress in mitigating memory-related constraints (Lin et al., 2022; Nguyen et al., 2024; 2025; 2026), the computational cost, measured in floating-point operations (FLOPs), remains a major bottleneck. Indeed, when training a neural network, the backward pass and weight updates account for the majority of FLOPs, far exceeding the cost of forward propagation alone (Plaut et al., 1986; Baydin et al., 2018).

To address this challenge, reducing the number of trainable parameters is a natural and effective solution (Frankle & Carbin, 2019). LoRA (Hu et al., 2022) and its variants (Xu et al., 2023; Zhang et al., 2023; Hayou et al., 2024; Liu et al., 2024) achieve this by training additional low-rank adapters while freezing the original model architecture, but at the cost of additional memory to store these adapters. Lin et al. (2022) take a different direction by proposing the Sparse Update (SU), which partially freezes the model to reduce the number of trainable parameters and enables training on a device with 256 KB SRAM. However, this approach requires an expensive evolutionary search before each deployment. Other related approaches share similar goals (Brock et al., 2017; Goutam et al., 2020; He et al., 2021; Liu et al., 2021a; Safayenikoo & Akturk, 2021; Bragagnolo et al., 2022; Yuan et al., 2022; Wang et al., 2023; Li et al., 2024; Quélennec et al., 2024), but they either rely on validation set, lack explicit resource constraints, or require substantial extra memory, making them infeasible for practical on-device learning (De Lange et al., 2021).

Motivated by the existence of stable convergence subspaces during training (Radiya-Dixit & Wang, 2020; Li & Zhang, 2021; Nguyen et al., 2026), we propose Quasi-stable Layer Freezing (QL) to reduce redundant backpropagation computation. Our key insight is that the temporal stability of a layer's input-output transformation can serve as an online proxy for the utility of further updates. QL detects this temporal stability online from streaming training data and freezes stable layers to avoid redundant computation. Unlike prior approaches, QL requires no validation set, negligible memory overhead, and adapts its decisions to the actual downstream tasks, making it suitable for unpredictable on-device input distributions.

Our main contributions are as follows:

- We formulate the redundant computational bottleneck of backpropagation and analyze its impact on training efficiency (Sec. 3.1).
- Building on prior findings on representation subspaces, we propose layer-wise transformation stability as an online proxy for the utility of further updates, and introduce QL to dynamically skip updates to quasi-stable layers. (Sec. 3.2).
- We demonstrate the effectiveness and generality of QL through extensive experiments on standard on-device learning benchmarks, across multiple model architectures (Sec. 4).

2 Related Works

In this section, we review existing approaches aimed at reducing the number of trainable parameters in order to save computational cost. They can be broadly categorized into two groups: *low-rank adapters* and *model freezing*. While several other research directions, such as low-rank decomposition, compact model design, quantization, sparsification, and knowledge distillation have also been explored, they lie beyond the scope of this work (Cheng et al., 2017; Deng et al., 2020). Therefore, they are not discussed here.

2.1 Low-rank Adapters

LoRA (Hu et al., 2022) is the most prominent candidate in this category. It introduces additional low-rank adapters while freezing the original model parameters, reducing the number of trainable parameters by up to four orders of magnitude.
Subsequent works (Xu et al., 2023; Zhang et al., 2023; Hayou et al., 2024; Liu et al., 2024) further explore this direction, but they all share a key limitation: during training, both the frozen weights and the adapters must co-exist in memory. This increases total memory usage and poses a serious challenge for on-device training scenarios, where the memory budget is strictly constrained.

2.2 Model Freezing

WUS (Safayenikoo & Akturk, 2021) freezes the entire model and updates only biases when validation accuracy plateaus, but this approach relies on validation set and lacks explicit resource constraints. FreezeOut (Brock et al., 2017), SpFDE (Yuan et al., 2022), and LayerOut (Goutam et al., 2020) progressively or randomly freeze layers without a principled metric or constraint, making them impractical for deployment. AutoFreeze (Liu et al., 2021a), PipeTransformer (He et al., 2021), Egeria (Wang et al., 2023), and Smartfrz (Li et al., 2024) determine freezing decisions by accumulating gradients or maintaining reference models, which introduces substantial memory overhead and is unsuitable for resource-constrained environments.

Another line of research focuses on freezing subsets of parameters within individual layers. Bragagnolo et al. (2022) and Quélennec et al. (2024) evaluate neuron velocity at the end of each training epoch to selectively update or freeze neurons accordingly. However, these approaches heavily depend on validation set and require significant additional computation and memory to repeatedly estimate neuron velocity.

The most relevant method in this line of work is SU (Lin et al., 2022), which statically freezes a subset of convolutional model parameters throughout training. By sparsely updating selected layers, SU reduces the FLOPs required for gradient computation during updates and enables on-device training with only 256 KB of SRAM. However, this approach requires an expensive evolutionary search to identify which parts of the model should be updated before deployment. As a result, the update policy is fixed in advance and cannot adapt to the actual downstream training dynamics observed on device.

Our proposed method addresses the above limitations. By leveraging the existence of stable subspaces during training (Nguyen et al., 2026; Radiya-Dixit & Wang, 2020; Li & Zhang, 2021), we introduce a principled "plug-and-play" layer-freezing strategy for on-device learning scenario that requires negligible additional memory, relies solely on information from the current training process, and operates under explicit resource constraints.

3 Method

We begin by identifying the redundant computations of backpropagation in Sec. 3.1. We then derive our key insights and present our method in Sec. 3.2. All derivations in this section are presented for convolutional layers, similar derivations for linear layers in transformer-based model and further theoretical analysis are provided in Appendix A.

3.1 Redundant Computation in Backpropagation

Computational Bottleneck. We consider a deep convolutional model and focus on a layer indexed by $i \in [0, L-1]$. The layer is parameterized by a kernel $\mathcal{W}_i \in \mathbb{R}^{C_i' \times C_i \times D_i \times D_i}$, which maps an input tensor $\mathcal{A}_i \in \mathbb{R}^{B \times C_i \times H_i \times W_i}$ to an output tensor $\mathcal{A}_{i+1} \in \mathbb{R}^{B \times C_i' \times H_i' \times W_i'}$. Here, B denotes the batch size, C_i and C_i' are the numbers of input and output channels, D_i is the spatial kernel size, and (H_i, W_i) and (H_i', W_i') denote the spatial resolutions of the input and output feature maps, respectively.

During the backward pass, the layer is updated as follows:

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

$$\mathcal{W}_i = \mathcal{W}_i + \eta \cdot \frac{\partial \mathcal{L}}{\partial \mathcal{W}_i}. \tag{2}$$

where $\mathcal{L}$ is the training loss, η is the learning rate, and $\textsc{conv}(\cdot, \cdot)$ denotes the convolution operation between the layer inputs and the upstream gradients. Eq. 1 and Eq.2 constitute a major computational cost of backpropagation (Plaut et al., 1986; Baydin et al., 2018; Bragagnolo et al., 2022).

Redundant Computation in Converged Layers. SVCCA (Raghu et al., 2017) was originally introduced to analyze the convergence behavior of deep neural networks by comparing layer representations across different layers or across different training epochs. Building on this line of work, PWCCA (Morcos et al., 2018) and later CKA (Kornblith et al., 2019) were proposed to address several limitations of earlier methods.

Formally, let $X \in \mathbb{R}^{n \times p_1}$ and $Y \in \mathbb{R}^{n \times p_2}$ denote the representations of n samples obtained either from two different layers or from the same layer at two different training states. The kernel CKA is defined as

$$\mathrm{CKA}(X, Y) = \frac{\mathrm{HSIC}(X^\dagger, Y^\dagger)}{\sqrt{\mathrm{HSIC}(X^\dagger, X^\dagger)\,\mathrm{HSIC}(Y^\dagger, Y^\dagger)}}. \tag{3}$$

Here, the empirical estimator of Hilbert-Schmidt Independence Criterion (HSIC) is given by

$$\mathrm{HSIC}(X^\dagger, Y^\dagger) = \frac{1}{(n-1)^2}\,\mathrm{tr}\big(X^\dagger H Y^\dagger H\big)\,, \tag{4}$$

where H is the centering matrix $H_n = I_n - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$. $X^\dagger$ and $Y^\dagger$ are computed using RBF kernel[1]

$$X^\dagger_{i,j} = \exp\left(-\frac{\|X_i - X_j\|^2}{2\sigma^2}\right). \tag{5}$$

with the bandwidth σ selected by the median heuristic. This normalization yields a similarity index that is invariant to orthogonal transformations and isotropic scaling, making CKA a reliable tool for measuring subspace similarity between layer representations in deep networks.

Using this framework during training, Kornblith et al. (2019) compare the representation of each layer at intermediate epochs with its final representation, and show that convergence in deep models is inherently heterogeneous: different layers reach their final representational subspaces at different stages of training. Despite this, standard training procedures still update all layers at every training step by computing Eq. 1 and Eq. 2. As a result, a substantial portion of backpropagation is spent on layers that have already reached their final representation.

3.2 Quasi-stable Layer Freezing

Layer-wise Subspace Variation. Prior work has shown that over-parameterized neural networks in fact reside in a low-dimensional intrinsic subspace (Li et al., 2018; Aghajanyan et al., 2020). Moreover, Radiya-Dixit & Wang (2020) and Li & Zhang (2021) observed that fine-tuned models remain close to their pretrained counterparts in parameter space. This observation was further supported by Nguyen et al. (2026), who demonstrated that such intrinsic subspaces can be considered relatively stable during training. This motivates us to characterize layer-wise training dynamics through temporal changes in the representation transformation induced by each layer.

Since learned representations tend to converge toward stable subspaces (Morcos et al., 2018), their temporal similarity provides a natural signal for estimating transformation stability. We therefore measure this similarity using CKA as

$$\mathcal{S}_i^{(t)} = \mathrm{CKA}\big(A_i^{(t)}, A_{i+1}^{(t)}\big), \tag{6}$$

where $A_i^{(t)} \in \mathbb{R}^{B \times C_i H_i W_i}$ denotes the unfolded input $\mathcal{A}_i^{(t)}$ of layer i at epoch t.

The subspace variation of this layer can be defined by

$$\mathcal{D}_i^{(t)} = \left|\mathcal{S}_i^{(t)} - \mathcal{S}_i^{(t-1)}\right|. \tag{7}$$

Our key insight is that, as a layer stabilizes, its input-output transformation varies less across consecutive training states. Therefore, a smaller $\mathcal{D}_i^{(t)}$ indicates greater temporal stability of layer i. In Sec. 4.2, we empirically show that layers with smaller $\mathcal{D}_i$ benefit less from further updates and can therefore be temporarily frozen with limited impact on optimization. Based on this criterion, we propose two layer-freezing strategies:

QL$_\tau$. A straightforward way is to freeze a layer once its subspace variation becomes sufficiently small. However, permanently freezing a layer can be unreliable, since it may be temporarily trapped in shallow local minima or saddle points (Dauphin et al., 2014).

To address this issue, we introduce a threshold-based rule. Given a tolerance τ, layer i is frozen at epoch t only if $\mathcal{D}_i^{(t)} \leq \tau$, and remains trainable otherwise. This allows layers to be unfrozen if subspace instability re-emerges during training.

[1] Kornblith et al. (2019) shows that linear kernel exhibits similar performance, comparison is in Appendix B.2.

QL$_\rho$. Selecting an appropriate threshold τ may require careful hyperparameter tuning. Moreover, threshold-based freezing does not provide explicit control over how many layers remain trainable at each epoch, which can be impractical in real-world, resource-constrained settings. To provide a more predictable, user-friendly, and resource-efficient alternative, we propose a partition-based freezing strategy.

After every Δt epochs, we compute the subspace variations for all layers and greedily freeze the most stable layers subject to a prescribed frozen-parameter budget ρ, defined as $i \in \arg\min_\rho \left\{\mathcal{D}_i^{(t)}\right\}_{i=1}^{L}$, where $\arg\min_\rho$ returns the indices corresponding to the smallest ρ fraction of values.

This approach enables explicit control over the number of trainable parameters, which is particularly important for deployment in real-world scenarios.

The detailed algorithm and tractable error bound can be found in Appendix A.2.

4 EXPERIMENTS

In this section, we present a series of experiments demonstrating the effectiveness of the proposed method. We first describe the experimental setup in Sec. 4.1. Sec. 4.2 validates the stability signal and the design choices underlying QL. Sec. 4.3 evaluates accuracy-efficiency trade-offs across different settings. Sec. 4.4 measures the resulting wall-clock speedup on real hardware.

All simulation experiments are conducted using PyTorch 1.13.1 on an NVIDIA A100 GPU with 40 GB of VRAM, while on-device experiments are performed on a Raspberry Pi 5 equipped with a Cortex-A76 CPU and 8 GB of RAM.

4.1 EXPERIMENTAL SETUP

Overview. Our goal is to enable on-device training, where networks pretrained on large-scale datasets are locally fine-tuned using task-specific data (Murshed et al., 2021). Since practical feasibility is a key concern, our main experiments are performed on MCUNet (Lin et al., 2022), a model that has been deployed on real edge devices with only 256 kB of SRAM. We additionally evaluate ViT-B/32 (Dosovitskiy, 2020) to assess the generality of QL across different model families.

We follow the benchmark setting of prior on-device learning studies (Lin et al., 2022; Nguyen et al., 2026), fine-tuning ImageNet-1K (Deng et al., 2009) pretrained models on CIFAR-10, CIFAR-100 (Krizhevsky, 2009), CUB (Wah et al., 2011), Flowers (Nilsback & Zisserman, 2008), and Pets (Zhang et al., 2022) using the same training configuration. Full implementation details are provided in Appendix B.1. To further assess the generality of QL beyond fine-tuning, we also evaluate training from scratch on ImageNet-1K.

Baselines. We compare QL against directly comparable baselines under matched trainable-parameter budgets. For convolutional models, we use SU (Lin et al., 2022), while for transformer models, we compare against LoRA (Hu et al., 2022), as discussed in Sec. 2. Vanilla backpropagation is included as the full-training reference in all settings. Beyond these direct comparisons, we additionally evaluate QL against other layer-freezing methods, including PipeTransformer (He et al., 2021), AutoFreeze (Liu et al., 2021a), and FreezeOut (Brock et al., 2017). Unlike QL, these methods start from full-model training and progressively freeze layers, and therefore do not enforce a prescribed resource budget from the outset.

Metrics and Logging. All methods are evaluated under comparable trainable budgets, reported as the percentage of trainable parameters (%TRAIN). We report the percentage of FLOPs (δ_{FLOPS}) relative to vanilla backpropagation, accounting for the overhead introduced by our method, together with the corresponding top-1 validation accuracy (ACC. in %). For LoRA, we additionally report the percentage of extra memory overhead (δ_{EXTRA}) introduced by its adapters relative to the model size.

Additional results on other compact architectures, including MobileNetV2 (Sandler et al., 2018), ResNet (He et al., 2016), Swin Transformer Tiny (Liu et al., 2021b), and TinyLlama (Zhang et al., 2024) are provided in Appendix B.2.

4.2 Preliminary Results

In this section, we empirically examine the key assumptions underlying QL and validate the design choices introduced in Sec. 3.

Subspace Variation during Training. We fine-tune ViT-B/32 and MCUNet on Pets using vanilla backpropagation with a cosine annealing learning-rate scheduler. At each epoch, we compute the total subspace variation across all layers using the first mini-batch, $\bar{\mathcal{D}} = \sum_{i=0}^{L-1} \mathcal{D}_i$. The results are shown in Fig. 2.
As training progresses, $\bar{\mathcal{D}}$ exhibits a clear overall decreasing trend, indicating that layer transformations become increasingly stable. The trajectory is nevertheless non-monotonic and contains local fluctuations, which may arise from stochastic mini-batch measurements as well as the non-convex optimization dynamics of neural network training.

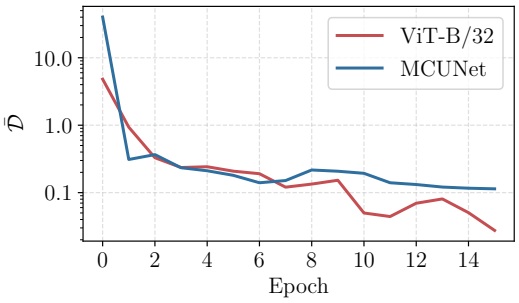


Figure 2: Total subspace variation $\bar{\mathcal{D}}$ across all layers of ViT-B/32 and MCUNet during fine-tuning on Pets.

Safe Freezing Ablation. We next directly examine whether $\mathcal{D}_i$ reflects how much a layer can still benefit from further optimization. We use Pets with a batch size of 256, which constructs a train set of 12 mini-batches. After the first two optimization steps, we compute $\mathcal{D}_i$ for every layer and rank them accordingly. Each model is then partitioned into six ordered chunks, $C_0, \ldots, C_5$, where C_0 contains the layers with the smallest $\mathcal{D}_i$ and C_5 those with the largest. We use six chunks because both ViT-B/32 and MCUNet can be evenly partitioned into six groups, enabling a consistent evaluation across architectures.
We evaluate two complementary settings over the remaining 10 mini-batches. In the *Train* setting, only one chunk C_j remains trainable while all other chunks are frozen. If a small $\mathcal{D}_i$ indeed indicates that a layer has largely stabilized, further updating such a chunk should provide limited additional optimization benefit. We therefore expect the final loss to decrease as the chunk-averaged $\mathcal{D}_i$ increases. In the *Freeze* setting, we instead freeze C_j and train the remaining chunks. The expected trend is reversed, since freezing a stable chunk with small $\mathcal{D}_i$ should have little effect, while freezing a less stable chunk should lead to a higher final loss.
As shown in Fig. 3, both settings exhibit the expected overall trends. Despite small fluctuations that may arise from stochastic mini-batch optimization, chunks with smaller $\mathcal{D}_i$ consistently behave as more stable components of the model. These results provide direct empirical evidence that $\mathcal{D}_i$ is a meaningful proxy for identifying layers that can be temporarily frozen with limited impact on optimization.

Effect of Reversible Freezing. The preceding results suggest that small $\mathcal{D}_i$ indicates layer stability at a given stage of training. However, such stability need not be permanent. To evaluate this effect, we introduce $\text{QL}_{\text{greedy}}$, which follows the same procedure as QL_τ but permanently freezes a layer once it is identified as quasi-stable.
We compare QL_τ and $\text{QL}_{\text{greedy}}$ when fine-tuning MCUNet with $\tau \in \{0.005, 0.01\}$. As shown in Fig. 4a, QL_τ consistently achieves higher accuracy across all evaluated datasets. This indicates that layers identified as stable at one stage of training may later benefit from additional updates. Allowing previously frozen layers to become trainable again is therefore important for preserving optimization flexibility and directly motivates the reversible freezing mechanism of QL.

Effect of Measurement Frequency. The main computational overhead of QL arises from periodically evaluating CKA. Although this overhead is generally small relative to backpropagation, it can become non-negligible for small layers or low-data regimes (see Appendix A.3). We therefore study the effect of the measurement interval Δt by fine-tuning ViT-B/32 with QL_ρ. Here, Δt denotes the number of epochs between two consecutive measurements of layer transformation stability.
Fig. 4b reports the average Top-1 accuracy and FLOPs across the evaluated datasets. Increasing Δt reduces the frequency of CKA evaluation and therefore lowers the associated computational overhead.

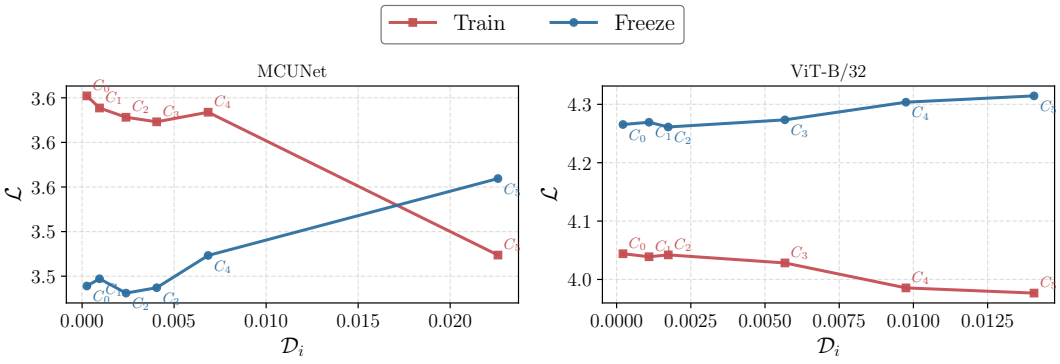
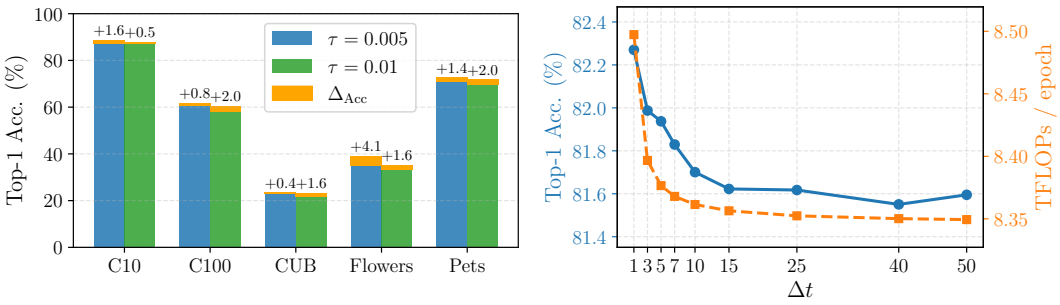


Figure 3: Layers are ranked by $\mathcal{D}_i$ and divided into six ordered chunks, from C_0 with the smallest $\mathcal{D}_i$ to C_5 with the largest. The horizontal axis reports the chunk-averaged $\mathcal{D}_i$, and the vertical axis the loss after the remaining optimization steps. Absolute losses are not directly comparable across the two settings because they follow different optimization trajectories.



(a) (b)

Figure 4: **(a)** Effect of reversible freezing. QL_τ consistently outperforms the irreversible $\text{QL}_{\text{greedy}}$ baseline on MCUNet, with Δ_{Acc} denoting the accuracy improvement over $\text{QL}_{\text{greedy}}$. **(b)** Accuracy–FLOPs trade-off of $\text{QL}_{\rho=0.9}$ under different measurement intervals Δt when fine-tuning ViT-B/32. The left axis reports the average Top-1 accuracy across datasets, while the right axis reports the average FLOPs per epoch, including both backward computation and CKA overhead.

However, less frequent measurements also make freezing decisions less responsive to changes in layer stability, which can degrade accuracy. Thus, Δt provides a direct trade-off between measurement overhead and the responsiveness of QL.

4.3 Main Results

Given these observations, we now evaluate whether QL translates this stability signal into practical efficiency gains. We choose QL_ρ with $\Delta t = 10$ for comparison against baselines, as it provides an explicit upper budget constraint while reducing the overhead (Fig. 4b).

Convolutional Models. We first compare QL with SU on MCUNet. Following the original SU configurations, we derive the corresponding proportion of trainable parameters and use the same values as the budget for QL_ρ. The results are reported in Table 1.
Under matched trainable-parameter budgets, QL achieves higher average accuracy than SU across both evaluated settings. At approximately 2.6% trainable parameters, QL improves the average accuracy from 52.2% to 53.4%, while at approximately 11.3%, the improvement increases from 54.4% to 56.3%. SU achieves lower FLOPs by relying on a sparse update pattern obtained through an expensive pre-deployment evolutionary search. In contrast, QL determines which layers to update online from the observed training dynamics, without requiring a task-specific search or modifying the internal structure of individual layers.

Table 1: MCUNet fine-tuning results under matched trainable-parameter budgets. Blue denotes task-agnostic methods, and red denotes task-dependent methods.

%TRAIN	METH.	CIFAR-10		CIFAR-100		CUB		FLOWERS		PETS		AVERAGE	
		$\delta_{\text{FLOPS}}\downarrow$	ACC.↑	$\delta_{\text{FLOPS}}\downarrow$	ACC.↑	$\delta_{\text{FLOPS}}\downarrow$	ACC.↑	$\delta_{\text{FLOPS}}\downarrow$	ACC.↑	$\delta_{\text{FLOPS}}\downarrow$	ACC.↑	$\delta_{\text{FLOPS}}\downarrow$	ACC.↑
2.61±0.01	SU	0.03	87.8	0.03	57.6	0.03	12.4	0.03	32.3	0.03	70.8	0.03	52.2
	QL	0.21	87.6	0.26	58.8	0.27	15.5	0.50	35.2	0.28	70.0	0.30	53.4
11.31±0.02	SU	0.11	87.8	0.11	58.4	0.11	19.2	0.11	34.9	0.11	71.4	0.11	54.4
	QL	0.34	88.4	0.38	61.2	0.46	21.7	0.57	39.5	0.46	70.8	0.44	56.3
100	VANI.	1.00	91.1	1.00	65.9	1.00	33.1	1.00	45.6	1.00	73.8	1.00	61.9

Table 2: ViT fine-tuning results under matched trainable-parameter budgets. We compare QL_ρ and LoRA by setting LoRA ranks to match the proportion of trainable parameters induced by QL_ρ, along with vanilla backpropagation.

%TRAIN	METH.	$\delta_{\text{FLOPS}}\downarrow$	$\delta_{\text{EXTRA}}\downarrow$	ACC.↑				
				CIFAR-10	CIFAR-100	CUB	FLOWERS	PETS
4.42±0.25	LORA	0.10	9.4	97.2	84.3	53.3	81.7	88.4
	QL	0.04	0.0	97.2	85.7	52.8	83.3	88.4
8.35±0.02	LORA	0.18	17.0	97.2	85.8	53.7	84.9	88.3
	QL	0.08	0.0	97.3	85.8	53.7	83.9	89.4
100	VANI.	1.00	0.0	97.3	85.7	50.8	85.7	89.1

Transformer Models. We next evaluate QL on ViT-B/32 and compare it with LoRA. To ensure a comparable training budget, we first apply QL_ρ with predefined values of ρ and then select the LoRA rank that most closely matches the resulting fraction of trainable parameters. The results are reported in Table 2.

At matched trainable-parameter budgets, QL requires substantially fewer FLOPs than LoRA while maintaining comparable accuracy. At approximately 4.4% trainable parameters, QL reduces the FLOPs ratio from 0.10 to 0.04, while at approximately 8.4%, it reduces the ratio from 0.18 to 0.08. Moreover, LoRA introduces up to 17% additional parameter memory for the adapters, whereas QL introduces no additional model parameters. These results indicate that directly skipping updates to quasi-stable layers can provide a more favorable compute-memory trade-off when the underlying model can be selectively frozen.

Overall, these results show that QL provides a favorable accuracy-efficiency trade-off across both convolutional and transformer-based models. Compared with SU, QL avoids pre-deployment search and adapts online to downstream training dynamics; compared with LoRA, it achieves larger FLOPs savings without introducing additional memory overhead.

Generalization to Pretraining. Although QL is primarily designed for resource-constrained fine-tuning, its formulation naturally extends to training from scratch. We therefore evaluate whether the same stability criterion can also reduce computation during training from scratch. We apply QL_τ to pretrain MCUNet and ViT-B/32 on ImageNet-1K using $\tau \in \{0.01, 0.005\}$ and compare against vanilla backpropagation.

As shown in Table 3, QL retains accuracy close to vanilla training while substantially reducing the average per-epoch FLOPs. With $\tau = 0.005$, MCUNet reaches 54.7% accuracy compared with 54.9% for vanilla training while using only 36% of the FLOPs, corresponding to a 64% reduction. On ViT-B/32, QL achieves 39.6% accuracy compared with 39.5% for vanilla while using 73% of FLOPs.

Comparison with Other Layer-Freezing Methods. We further compare QL with other layer-freezing baselines, including PipeTransformer, AutoFreeze, and FreezeOut. We fine-tune MCUNet under the same benchmark setting and use the 11.3% trainable-parameter configuration of QL as the target budget. Since these baselines start from full-model training and progressively freeze layers during optimization, we report their trainable-parameter ratios averaged over all training epochs.

As shown in Table 4, QL achieves the highest average accuracy while using the smallest average trainable-parameter budget among the competitive methods. QL reaches 56.3% average accuracy with 11.3% trainable parameters, compared with 54.7% for PipeTransformer at 11.6% and 55.3% for AutoFreeze at 13.9%. Notably, these progressive-freezing methods require full-model training at the

Table 3: Pretraining on ImageNet-1K with QL_τ. We report the fraction of trainable parameters (%Train), average per-epoch FLOPs relative to vanilla backpropagation (δ_{FLOPs}), and Top-1 accuracy.

METHOD	τ	MCUNET			VIT-B/32		
		%TRAIN	$\delta_{\text{FLOPs}}\downarrow$	ACC.↑	%TRAIN	$\delta_{\text{FLOPs}}\downarrow$	ACC.↑
QL	0.01	4.4	0.12	52.5	44.1	0.46	37.9
	0.005	22.2	0.36	54.7	69.3	0.73	**39.6**
VANILLA BP	–	100	1.00	**54.9**	100	1.00	39.5

Table 4: Comparison with other layer-freezing baselines on MCUNet.

METHOD	CIFAR-10		CIFAR-100		CUB		FLOWERS		PETS		AVERAGE	
	%TRAIN.	ACC.↑	%TRAIN.	ACC.↑	%TRAIN.	ACC.↑	%TRAIN.	ACC.↑	%TRAIN.	ACC.↑	%TRAIN.	ACC.↑
QL	**11.3**	88.4	**11.3**	**61.2**	**11.3**	21.7	**11.3**	**39.5**	**11.3**	70.8	**11.3**	**56.3**
PIPETRANSFORMER	11.6	88.3	11.6	60.7	11.6	**21.8**	11.6	33.3	11.6	69.4	11.6	54.7
AUTOFREEZE	16.0	**89.2**	11.8	61.0	15.0	20.7	14.5	35.9	12.0	**71.0**	13.9	55.3
FREEZEOUT	64.0	10.7	64.0	1.2	64.0	0.9	63.7	1.8	63.9	3.8	63.9	3.7

beginning of optimization, whereas QL satisfies the prescribed trainable-parameter budget from the outset. This distinction is particularly important for resource-constrained devices, where full-model training may be infeasible even temporarily. Moreover, AutoFreeze and PipeTransformer rely on gradient accumulation, which introduces additional overhead.[2]

4.4 ON-DEVICE PROCESSING TIME

In this section, we evaluate whether the FLOPs savings of QL translate into wall-clock speedups on real resource-constrained hardware. For practical on-device deployment, QL_ρ is used to impose an explicit computational budget while fine-tuning MCUNet on CIFAR-10 under different ρ. The stability measurement is performed at every epoch by setting $\Delta t = 1$, and the resulting backward latency is compared against vanilla backpropagation on a Raspberry Pi 5.

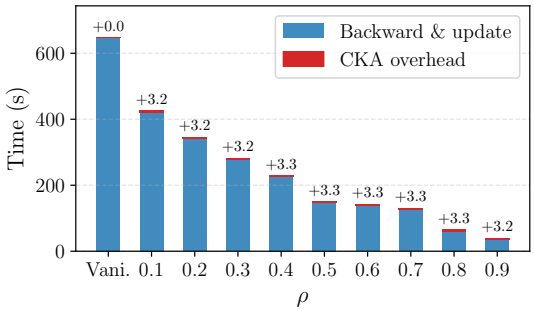


Figure 5: Backward time per epoch to fine-tune MCUNet on CIFAR-10 with QL under different budgets ρ and $\Delta t = 1$ on a Raspberry Pi 5.

Fig. 5 reports the average time required for gradient computation, weight updates, and CKA evaluation per epoch. As ρ increases, fewer parameters remain trainable, leading to a consistent reduction in processing time. At $\rho = 0.9$ ($\approx 10\%$ trainable parameters), QL_ρ achieves a $16.7\times$ speedup over vanilla backpropagation on the Raspberry Pi 5.

Further results on GPU can be found in Appendix B.2

5 CONCLUSION

In this work, we introduce Quasi-stable Layer Freezing (QL) to reduce redundant backpropagation under resource constraints. QL estimates layer-wise transformation stability online from streaming training data and temporarily skips updates to quasi-stable layers. This reduces gradient computation and parameter-update costs without requiring a reference model, extra forward passes, or memory-intensive training statistics.

QL reduces up to 96% of backward FLOPs during fine-tuning without sacrificing accuracy. Beyond fine-tuning, QL is also effective during pretraining, where it saves up to 64% of redundant computation while preserving model performance. In real-device experiments, QL achieves up to a $16.7\times$ speedup on a Raspberry Pi 5, demonstrating its effectiveness in practical deployment.

[2]Further discussion of gradient-based and reference-model-based methods are provided in Appendix B.2.

AI USE STATEMENT

In this work, we used generative AI tools for assisting in the writing of proofs, providing feedback on research experiments, implementing methods, and assisting with translation. We have not used generative AI tools for the rest of the required disclosure tasks. Additionally, we used generative AI tools for creating or editing software code, sourcing/searching for information, and editing a research paper to improve readability. We have reviewed all AI-assisted work. All AI-assisted code and any manuscript sections edited with the help of generative AI were reviewed and verified by at least two authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The complete experimental setup, including the models, datasets, and evaluation environment, is detailed in Sec. 4.1. Further implementation details are provided in Appendix B.1. The code will be released publicly upon acceptance.

REFERENCES

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*, 2020.

Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18 (153):1–43, 2018.

Andrea Bragagnolo, Enzo Tartaglione, and Marco Grangetto. To update or not to update? neurons at equilibrium in deep models. *Advances in neural information processing systems*, 35:22149–22160, 2022.

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Freezeout: Accelerate training by progressively freezing layers. *arXiv preprint arXiv:1706.04983*, 2017.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

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

Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Advances in neural information processing systems*, 27, 2014.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

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

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

Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJl-b3RcF7`.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. *Advances in neural information processing systems*, 30, 2017.

Kelam Goutam, S Balasubramanian, Darshan Gera, and R Raghunatha Sarma. Layerout: Freezing layers in deep neural networks. *SN Computer Science*, 1(5):295, 2020.

Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*, 2024.

Chaoyang He, Shen Li, Mahdi Soltanolkotabi, and Salman Avestimehr. Pipetransformer: Automated elastic pipelining for distributed training of transformers. *arXiv preprint arXiv:2102.03161*, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

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

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, pp. 3519–3529. PMlR, 2019.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. *arXiv preprint arXiv:1804.08838*, 2018.

Dongyue Li and Hongyang Zhang. Improved regularization and robustness for fine-tuning in neural networks. *Advances in Neural Information Processing Systems*, 34:27249–27262, 2021.

Sheng Li, Geng Yuan, Yue Dai, Youtao Zhang, Yanzhi Wang, and Xulong Tang. Smartfrz: An efficient training framework using attention-based layer freezing. *arXiv preprint arXiv:2401.16720*, 2024.

Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. On-device training under 256kb memory. *Advances in Neural Information Processing Systems*, 35:22941–22954, 2022.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024.

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

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

Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. *Advances in neural information processing systems*, 31, 2018.

MG Sarwar Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, and Faraz Hussain. Machine learning at the network edge: A survey. *ACM Computing Surveys (CSUR)*, 54(8):1–37, 2021.

Le-Trung Nguyen, Aël Quélennec, Enzo Tartaglione, Samuel Tardieu, and Van-Tam Nguyen. Activation map compression through tensor decomposition for deep learning. *Advances in Neural Information Processing Systems*, 37:130384–130407, 2024.

Le-Trung Nguyen, Aël Quélennec, Van-Tam Nguyen, and Enzo Tartaglione. Beyond low-rank decomposition: A shortcut approach for efficient on-device learning. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=YWUYqN2laV.

Le-Trung Nguyen, Enzo Tartaglione, and Van-Tam Nguyen. Efficient resource-constrained training of transformers via subspace optimization. In *The Fourteenth International Conference on Learning Representations*, 2026.

Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, 2008.

Danilo Pietro Pau and Fabrizio Maria Aymone. Suitability of forward-forward and pepita learning to mlcommons-tiny benchmarks. In *2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, pp. 1–6. IEEE, 2023.

D. C. Plaut, S. J. Nowlan, and G. E. Hinton. Experiments on learning back propagation. Technical Report CMU–CS–86–126, Carnegie–Mellon University, Pittsburgh, PA, 1986.

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*, pp. 685–694, 2024.

Evani Radiya-Dixit and Xin Wang. How fine can fine-tuning be? learning efficient language models. In *International Conference on Artificial Intelligence and Statistics*, pp. 2435–2443. PMLR, 2020.

Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. *Advances in neural information processing systems*, 30, 2017.

Pooneh Safayenikoo and Ismail Akturk. Weight update skipping: Reducing training time for artificial neural networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 11(4): 563–574, 2021.

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

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in neural information processing systems*, 31, 2018.

Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset, 2011.

Yiding Wang, Decang Sun, Kai Chen, Fan Lai, and Mosharaf Chowdhury. Egeria: Efficient dnn training with knowledge-guided layer freezing. In *Proceedings of the eighteenth European conference on computer systems*, pp. 851–866, 2023.

Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, Xiaopeng Zhang, and Qi Tian. Qa-lora: Quantization-aware low-rank adaptation of large language models. *arXiv preprint arXiv:2309.14717*, 2023.

Geng Yuan, Yanyu Li, Sheng Li, Zhenglun Kong, Sergey Tulyakov, Xulong Tang, Yanzhi Wang, and Jian Ren. Layer freezing & data sieving: Missing pieces of a generic framework for sparse training. *Advances in Neural Information Processing Systems*, 35:19061–19074, 2022.

Hui Zhang, Shenglong Zhou, Geoffrey Ye Li, and Naihua Xiu. 0/1 deep neural networks via block coordinate descent. *arXiv preprint arXiv:2206.09379*, 2022.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*, 2023.

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

A Additional Theoretical Details

A.1 Details of Weight Updates in Backpropagation for Linear Layers

Let i denote the index of a linear layer in a transformer-based model, characterized by a weight tensor $\mathcal{W}_i \in \mathbb{R}^{O_i \times I_i}$, where I_i and O_i are the number of input and output channels, respectively. This layer processes an input activation tensor $\mathcal{A}_i \in \mathbb{R}^{B \times N_i \times I_i}$ and produces an output tensor $\mathcal{A}_{i+1} \in \mathbb{R}^{B \times N_i \times O_i}$. Here, B denotes the batch size, while N_i the sequence length or number of tokens.

During the backward pass, gradients are computed via the chain rule as

$$\frac{\partial \mathcal{L}}{\partial \mathcal{W}_i} = \frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}}^{\top} \cdot \frac{\partial \mathcal{A}_{i+1}}{\partial \mathcal{W}_i} = \frac{\partial \mathcal{L}}{\partial \mathcal{A}_{i+1}}^{\top} \cdot \mathcal{A}_i, \tag{8}$$

where $\top$ denotes the transpose over the last two dimensions. The weights $\mathcal{W}_i$ are then updated according to Eq. 2.

A.2 Details of QL Algorithm

Algorithm 1 summarizes the two variants of QL. Both variants periodically measure the change in inter-layer representation similarity on the first mini-batch every Δt epochs.

QL$_\tau$ freezes layers whose subspace variation $\mathcal{D}_i$ falls below a predefined threshold τ. In contrast, QL$_\rho$ first estimates a freezing parameter budget from the target freeze ratio ρ, and then greedily selects the most stable layers, i.e., those with the lowest $\mathcal{D}_i$, while keeping the total number of frozen parameters within this budget.

Error Bound. We provide a simple perturbation analysis to characterize the error induced by QL. Consider a network $f_\theta = f_{L-1} \circ \cdots \circ f_0$, and let $\mathcal{I}^{(t)}$ denote the set of layers frozen at epoch t. We compare the QL-frozen model $\tilde{\theta}$ with a fully updated model θ^{full}, which follows vanilla backpropagation and keeps all layers trainable.

For each frozen layer $i \in \mathcal{I}^{(t)}$, we define the local freezing error as

$$\varepsilon_i^{(t)} = \mathbb{E}_{x \sim \mathcal{D}} \left[\left\| f_i(A_i; \theta_i^{\text{full}}) - f_i(A_i; \tilde{\theta}_i) \right\|_2 \right], \tag{9}$$

where x is sampled from the data distribution $\mathcal{D}$ and A_i denotes the input representation of layer i. This quantity measures the average local output perturbation introduced by freezing layer i, compared with the output that would have been produced if the layer had remained trainable.

Following standard Lipschitz-based perturbation analyses of neural networks (Szegedy et al., 2013; Virmaux & Scaman, 2018), we assume that each subsequent layer f_j is locally K_j-Lipschitz with respect to its input along the training trajectory. By the composition property of Lipschitz functions, the perturbation introduced at layer i is amplified by at most $\prod_{j=i+1}^{L-1} K_j$ when propagated to the network output. Summing the contributions of all frozen layers gives

$$\mathbb{E}_{x \sim \mathcal{D}} \left[\left\| f_{\theta^{\text{full}}}(x) - f_{\tilde{\theta}}(x) \right\|_2 \right] \leq \sum_{i \in \mathcal{I}^{(t)}} \varepsilon_i^{(t)} \prod_{j=i+1}^{L-1} K_j, \tag{10}$$

where $f_{\theta^{\text{full}}}(x)$ denotes the prediction of the fully updated model trained with vanilla backpropagation, and $f_{\tilde{\theta}}(x)$ denotes the prediction of the QL-frozen model.

QL freezes layers based on the temporal stability of their input-output transformation. As defined in Eq. 6 and Eq. 7, this stability is quantified by the subspace variation $D_i^{(t)}$. We assume that the local freezing error is controlled by this temporal variation:

$$\varepsilon_i^{(t)} \leq \alpha_i D_i^{(t)}, \tag{11}$$

where α_i is a layer-dependent regularity constant. This assumption formalizes the intuition behind QL: when the transformation of a layer changes only marginally across epochs, freezing that layer induces only a small local perturbation.

Algorithm 1 Quasi-stable Layer Freezing (QL)

Require: Measurement interval Δt, and either threshold τ or freeze ratio ρ

if using QL_ρ **then**
  Let P_i denote the number of parameters in layer i
  Set the frozen-parameter budget $\mathcal{B} \leftarrow \rho \sum_{i=0}^{L-1} P_i$
end if
for each training epoch e **do**
  if $e \bmod \Delta t = 0$ **then**
    if $e = 0$ **then**
      Evaluate two consecutive mini-batches
      for $i = 0$ to $L-1$ **do**
        $s_i^{(1)} \leftarrow \mathrm{CKA}(A_i^{(1)}, A_{i+1}^{(1)})$
        $s_i^{(2)} \leftarrow \mathrm{CKA}(A_i^{(2)}, A_{i+1}^{(2)})$
        $\mathcal{D}_i \leftarrow |s_i^{(2)} - s_i^{(1)}|$
        $S_i^{\mathrm{prev}} \leftarrow s_i^{(2)}$
      end for
    else
      Evaluate the first mini-batch
      for $i = 0$ to $L-1$ **do**
        $s_i \leftarrow \mathrm{CKA}(A_i, A_{i+1})$
        $\mathcal{D}_i \leftarrow |s_i - S_i^{\mathrm{prev}}|$
        $S_i^{\mathrm{prev}} \leftarrow s_i$
      end for
    end if
    if using QL_τ **then**
      $\mathcal{I} \leftarrow \{i : \mathcal{D}_i \le \tau\}$
    else
      Sort layers by increasing $\mathcal{D}_i$
      Greedily construct $\mathcal{I}$ such that $\sum_{i \in \mathcal{I}} P_i \le \mathcal{B}$
    end if
    Freeze layers in $\mathcal{I}$ and unfreeze all remaining layers
  end if
  Train with the current layer configuration
end for

For QL_τ, a layer is frozen only when $D_i^{(t)} \le \tau$. Substituting Eq. 11 into Eq. 10 gives

$$\mathbb{E}_{x\sim\mathcal{D}}\left[\left\|f_{\theta^{\mathrm{full}}}(x) - f_{\tilde{\theta}}(x)\right\|_2\right] \le \tau \sum_{i\in\mathcal{I}_\tau^{(t)}} \alpha_i \prod_{j=i+1}^{L-1} K_j. \tag{12}$$

Therefore, a smaller threshold τ imposes a stricter stability criterion and directly tightens the upper bound on the freezing-induced output deviation.

On the other hand, QL_ρ does not directly impose an upper threshold on each layer subspace variation $D_i^{(t)}$. Instead, it controls the total frozen parameter budget. Let $\mathcal{I}_\rho^{(t)}$ denote the set of layers selected by QL_ρ at epoch t under the budget constraint

$$\sum_{i\in\mathcal{I}_\rho^{(t)}} P_i \le \rho \sum_{i=0}^{L-1} P_i,$$

where P_i is the number of parameters in layer i. Given this budget, QL_ρ greedily selects layers with the smallest subspace variations $D_i^{(t)}$ (Algorithm 1). Substituting Eq. 11 into Eq. 10, the output deviation induced by QL_ρ is bounded by the accumulated contributions of the selected frozen layers:

$$\mathbb{E}_{x\sim\mathcal{D}}\left[\left\|f_{\theta^{\mathrm{full}}}(x) - f_{\tilde{\theta}}(x)\right\|_2\right] \le \sum_{i\in\mathcal{I}_\rho^{(t)}} \alpha_i D_i^{(t)} \prod_{j=i+1}^{L-1} K_j. \tag{13}$$

Each term in the summation is non-negative: $D_i^{(t)}$ is an absolute difference, α_i is a non-negative regularity constant, and the Lipschitz constants K_j are non-negative. Therefore, decreasing ρ reduces the frozen parameter budget, which limits the size of $\mathcal{I}_\rho^{(t)}$ and removes some non-negative error contributions from the summation. This yields a smaller accumulated freezing error bound. Conversely, increasing ρ allows more layers to be frozen, which can provide larger backpropagation FLOPs savings but may increase the accumulated error bound.

Finally, if the loss function $\mathcal{L}$ is $K_{\mathcal{L}}$-Lipschitz with respect to the model prediction, the corresponding loss deviation satisfies

$$\left|\mathcal{L}(\tilde{\theta}) - \mathcal{L}(\theta^{\text{full}})\right| \leq K_{\mathcal{L}} \sum_{i \in \mathcal{I}^{(t)}} \alpha_i D_i^{(t)} \prod_{j=i+1}^{L-1} K_j. \tag{14}$$

This result formalizes the trade-off controlled by QL. Smaller values of τ or ρ make freezing more conservative, reducing the accumulated freezing-induced error and better preserving accuracy, but also lowering FLOPs savings. Conversely, larger values of τ or ρ allow QL to freeze more layers, which improves FLOPs savings and wall-clock speedups but may increase the deviation from full backpropagation. This trade-off is consistent with our empirical results in Sec. 4.

A.3 Details of Computational Analysis

Overhead. The only computational overhead introduced by our method stems from the evaluation of CKA. Consider two matrices $X \in \mathbb{R}^{n \times p_1}$ and $Y \in \mathbb{R}^{n \times p_2}$, representing the activations of n samples, either extracted from two different layers or from the same layer at two distinct training epochs. The cost of constructing the centering matrices H for each sample X and Y is

$$\Theta_{\text{centering}} = \Theta_{\text{centering-}X} = \Theta_{\text{centering-}Y} = 6n^2. \tag{15}$$

Applying the RBF kernel to X (Eq. 5) incurs a cost of

$$\Theta_{\text{RBF-}X^\dagger} = 2n^2 p_1 + 7n^2. \tag{16}$$

The cost of computing the HSIC between the two samples via Eq. 4 is therefore

$$\begin{aligned} \Theta_{\text{HSIC-}X^\dagger Y^\dagger} &= 2\Theta_{\text{centering}} + \Theta_{\text{RBF-}X^\dagger} + \Theta_{\text{RBF-}Y^\dagger} + \Theta_{\text{tr}(X^\dagger, Y^\dagger)} \\ &= 12n^2 + (2n^2 p_1 + 7n^2) + (2n^2 p_2 + 7n^2) + 2n^2 \\ &= 2n^2(p_1 + p_2) + 28n^2. \end{aligned} \tag{17}$$

Finally, the total overhead of computing CKA using Eq. 3 is given by

$$\begin{aligned} \Theta_{\text{CKA}} &= \Theta_{\text{HSIC-}X^\dagger Y^\dagger} + \Theta_{\text{HSIC-}X^\dagger X^\dagger} + \Theta_{\text{HSIC-}Y^\dagger Y^\dagger} + 2\Theta_{\sqrt{(\cdot)}} \\ &= 2\Theta_{\text{centering-}X} + \Theta_{\text{RBF-}X^\dagger} + \Theta_{\text{RBF-}Y^\dagger} + \Theta_{\text{tr}(X^\dagger, X^\dagger)} + \Theta_{\text{tr}(Y^\dagger, Y^\dagger)} + \Theta_{\text{tr}(X^\dagger, Y^\dagger)} \\ &= 2n^2(p_1 + p_2) + 32n^2. \end{aligned} \tag{18}$$

Convolutional Layers. For a convolutional layer i, before to computing CKA, we reshape the activation tensors $\mathcal{A}_i \in \mathbb{R}^{B \times C_i \times H_i \times W_i}$ and $\mathcal{A}_{i+1} \in \mathbb{R}^{B \times C_i' \times H_i' \times W_i'}$ into $A_i \in \mathbb{R}^{B \times C_i H_i W_i}$ and $A_{i+1} \in \mathbb{R}^{B \times C_i' H_i' W_i'}$, respectively (Raghu et al., 2017). The corresponding cost of performing CKA is then obtained by substituting $n = B$, $p_1 = C_i H_i W_i$, and $p_2 = C_i' H_i' W_i'$ to Eq. 18, yielding

$$\Theta_{\text{CKA-CONV}} = 2B^2(C_i H_i W_i + C_i' H_i' W_i' + 16)\,. \tag{19}$$

The cost of gradient computation and weight update in Eqs. 1 and 2 is

$$\Theta_{\text{WEIGHT-CONV}} = D_i^2 C_i C_i' B H_i' W_i', \tag{20}$$

For simplicity, we assume that the input, output activations and the kernel have identical dimensions, i.e., $C_i = C_i' = H_i = W_i = H_i' = W_i' = D_i = S$. The ratio between the CKA overhead and the cost of weight updates can therefore be expressed as

$$\begin{aligned} \mathcal{R} &= \frac{\Theta_{\text{CKA-CONV}}}{\mathcal{N}\,\Theta_{\text{WEIGHT-CONV}}} \\ &= \frac{4B}{\mathcal{N}} \left(\frac{1}{S^3} + \frac{8}{S^6}\right) \times 100\,(\%), \end{aligned} \tag{21}$$

where $\mathcal{N}$ denotes the total number of mini-batches over the entire training process. It is immediate that $\mathcal{R}$ decreases as the layer size S increase. In our analysis, we set $B = 128$. Figure 6 reports the theoretical evolution of $\mathcal{R}$ as a function of the layer size S.

Fig. 6 predicts the ratio $\mathcal{R}$ between the overhead and the cost of updating one layer by computing Eqs. 1 and 2 in percentage.

As the number of training mini-batches grows and layer size becomes larger, which is typically the case in modern deep learning models (Kaplan et al., 2020; Vaswani et al., 2017; Keskar et al., 2016; Shoeybi et al., 2019), the overhead becomes increasingly negligible compared to the cost of updating the layer. Moreover, once a subset of layers is frozen, this overhead is easily amortized. This allows us to compute CKA without increasing the total computational cost.

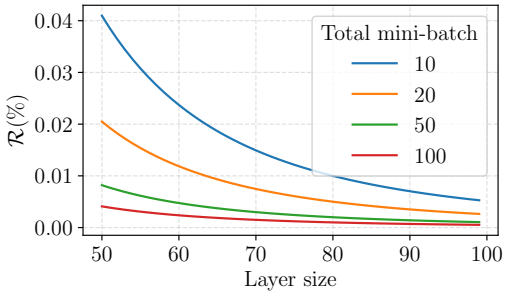


Figure 6: Theoretical FLOPs overhead of CKA relative to gradient-based weight updates as the layer size increases.

Linear Layers. Analogously, before computing CKA for a linear layer, we reshape $\mathcal{A}_i \in \mathbb{R}^{B\times N_i\times I_i}$ and $\mathcal{A}_{i+1} \in \mathbb{R}^{B\times N_i\times O_i}$ into $A_i \in \mathbb{R}^{B\times N_i I_i}$ and $A_{i+1} \in \mathbb{R}^{B\times N_i O_i}$, respectively. Under this representation, the overhead of computing CKA is

$$\Theta_{\text{CKA-LINEAR}} = 2B^2(N_iI_i + N_iO_i + 16)\,. \tag{22}$$

The cost of gradient computation and weight update in Eqs. 8 and 2 is

$$\Theta_{\text{WEIGHT-LINEAR}} = 2BN_iI_iO_i. \tag{23}$$

For standard transformer architectures, it is common to have $O_i = 4I_i$ (Vaswani et al., 2017), yielding

$$\begin{aligned}\mathcal{R} &= \frac{\Theta_{\text{CKA-LINEAR}}}{\mathcal{N}\,\Theta_{\text{WEIGHT-LINEAR}}} \\ &= \frac{B}{\mathcal{N}I_i}\left(\frac{5}{4} + \frac{4}{N_iI_i}\right) \times 100\,(\%),\end{aligned} \tag{24}$$

Memory Overhead. Consider a network with L layers. Applying QL only requires storing one scalar transformation similarity score S_i (Eq. 6) for each layer $i \in \{0, \ldots, L-1\}$. Using 32-bit floating-point values, the additional memory overhead is therefore

$$\Delta_{\text{mem}} = \frac{32L}{8 \times 1024^2} = \frac{4L}{1024^2}\ \text{(MB)}. \tag{25}$$

This overhead is negligible compared with the memory required to store model parameters, activations, and optimizer states. For example, even for a network with $L = 100$ layers, QL only introduces approximately 3.8×10^{-4} MB of additional memory. CKA additionally requires temporary workspace during measurement, which is released immediately afterward and does not increase peak memory in our measured settings.

B Additional Experimental Details

B.1 Details of Experimental Setup

Fine-tuning Setup. We initialize all networks from ImageNet-1K pretrained weights and then adapt them to each downstream task using an $80\%/20\%$ train–validation split over 50 epochs. We train the models with the cross-entropy objective and optimize them using stochastic gradient descent. The learning rate is initialized at 0.05 and gradually reduced following a cosine annealing schedule. We set the momentum coefficient to 0 and fix the weight decay at 1×10^{-4}. To improve training stability, we apply L_2 gradient clipping with a maximum norm of 2.0. For data preprocessing and augmentation, we employ random resizing, horizontal flipping, normalization, and a mini-batch size of 128 throughout all experiments.

Table 5: Comparison between linear and RBF kernels for HSIC computation under QL_ρ when fine-tuning ResNet18. Results are reported for $\rho \in \{0.7, 0.9\}$ across five datasets.

METHOD	KERNEL	CIFAR-10		CIFAR-100		CUB		FLOWERS		PETS		AVERAGE	
		$\delta_{\text{FLOPs}}\downarrow$	ACC.↑	$\delta_{\text{FLOPs}}\downarrow$	ACC.↑	$\delta_{\text{FLOPs}}\downarrow$	ACC.↑	$\delta_{\text{FLOPs}}\downarrow$	ACC.↑	$\delta_{\text{FLOPs}}\downarrow$	ACC.↑	$\delta_{\text{FLOPs}}\downarrow$	ACC.↑
$\text{QL}_{\rho=0.7}$	RBF	0.31	95.1	0.32	78.2	0.37	56.3	0.59	80.8	0.39	89.8	0.40	80.0
	LINEAR	0.32	95.2	0.31	78.5	0.34	55.7	0.41	81.0	0.34	88.6	0.34	79.8
$\text{QL}_{\rho=0.9}$	RBF	0.11	94.8	0.11	76.6	0.16	54.7	0.38	79.9	0.19	88.0	0.19	78.8
	LINEAR	0.10	94.6	0.12	76.3	0.13	55.4	0.19	78.8	0.13	87.7	0.13	78.6
VANI.	–	1.00	95.4	1.00	78.8	1.00	55.8	1.00	83.0	1.00	89.5	1.00	80.5

Pretraining Setup. Following the same protocol, all pretraining runs are conducted for 50 epochs and use L_2 gradient clipping with a maximum norm of 2.0. Unlike fine-tuning, pretraining starts from randomly initialized models. We optimize the networks using AdamW, with an initial learning rate of 5×10^{-4}, momentum 0.9, and a weight decay of 1×10^{-3}. As in fine-tuning, we adopt a cosine annealing schedule over 50 epochs, without any warm-up phase.

B.2 ADDITIONAL RESULTS

Post-hoc Comparison with Final Representations. We fine-tune ViT-B/32 on Flowers. The training data is randomly shuffled at every epoch. At each epoch, we use CKA to measure the representational similarity between each layer and its counterpart at the final epoch. The measurement is performed using only the first mini-batch. Due to data shuffling, this mini-batch is randomly sampled at each epoch. A higher CKA value indicates that the layer is closer to its destination subspace, and a score of 1.0 corresponds to full convergence.

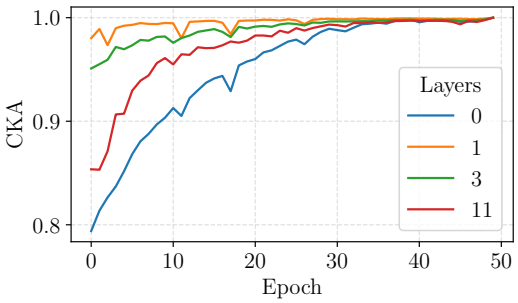


Figure 7: Evolution of subspace similarity (CKA) between layer representations at each training epoch and their final representations, for ViT-B/32 layers 0 (first from the input), 1, 3, and 11.

As shown in Fig. 7 different layers approach their final representations at different rates, further supporting heterogeneous representation dynamics during training. The trajectories are non-monotonic, consistent with the temporal fluctuations observed in Sec. 4.2.

Linear Kernel vs. RBF Kernel. We fine-tune ResNet18 with QL_ρ at $\rho \in \{0.7, 0.9\}$ and compare the use of a linear kernel and an RBF kernel for computing HSIC (Eq. 4). The results are reported in Tab. 5.

Consistent with the original findings of Kornblith et al. (2019), we observe similar accuracy between the two kernels across datasets. In contrast, the linear kernel yields slightly lower FLOPs overhead than the RBF variant in most cases. In our main experiments, we nevertheless adopt the RBF kernel to demonstrate that even with a more expensive kernel, our method does not increase the overall training cost.

MobileNetV2 Fine-tuning. We conduct the same fine-tuning experiments described in Sec. 4.3 on MobileNetV2.

The observed behavior in Tab. 6 closely mirrors the trends discussed in Sec. 4.3. In particular, QL_ρ maintains accuracy close to vanilla training while reducing FLOPs by up to 80%, without requiring the task-specific pre-deployment search used by SU.

Swint-T on Different Random Seeds. We follow the same experimental setup as in Sec. 4.3 and compare QL, LoRA, and vanilla training when fine-tuning Swin-T under three different random seeds (233, 234, and 235). The results are summarized in Table 7. Across all seeds, QL consistently achieves performance comparable to LoRA, while reducing the FLOPs ratio from 0.55 to 0.42 and introducing no additional memory overhead. Importantly, the results remain consistent across seeds,

Table 6: Comparison between SU, QL_ρ, and vanilla backpropagation under matched trainable-parameter budgets when fine-tuning MobileNetV2.

%TRAIN	METH.	CIFAR-10		CIFAR-100		CUB		FLOWERS		PETS	
		$\delta_{\text{FLOPS}}\downarrow$	ACC.↑	$\delta_{\text{FLOPS}}\downarrow$	ACC.↑	$\delta_{\text{FLOPS}}\downarrow$	ACC.↑	$\delta_{\text{FLOPS}}\downarrow$	ACC.↑	$\delta_{\text{FLOPS}}\downarrow$	ACC.↑
15.07±0.01	SU	0.11	94.8	0.11	77.9	0.11	57.6	0.11	85.2	0.11	89.2
	QL	0.39	94.7	0.31	78.4	0.30	57.0	0.48	78.7	0.32	90.0
100	VANI.	1.00	95.3	1.00	78.4	1.00	57.4	1.00	80.2	1.00	89.8

Table 7: SwinT fine-tuning results under matched trainable-parameter budgets over three random seeds (233, 234, and 235). We compare QL and LoRA under the same trainable-parameter budget, together with vanilla backpropagation.

%TRAIN	METH.	$\delta_{\text{FLOPS}}\downarrow$	$\delta_{\text{EXTRA}}\downarrow$	ACC.↑				
				CIFAR-10	CIFAR-100	CUB	FLOWERS	PETS
11.2±0.1	LORA	0.55	23.7	97.7±0.1	86.7±0.2	78.4±1.1	90.0±1.2	95.4±1.0
	QL	0.42	0.0	97.8±0.0	86.8±0.2	78.3±1.2	90.3±0.3	95.6±1.4
100	VANI.	1.00	0.0	98.0±0.0	87.0±0.2	76.9±0.9	89.8±0.4	95.4±1.1

suggesting that the observed trends are not sensitive to the random seed. Based on this observation, we use seed 233 for the main experiments to reduce the overall experimental cost.

ResNet18 on ImageNet. Fig. 8a and Tab. 8 report the pretraining results of ResNet18 on ImageNet, following the same training policy described in Sec. 4.1.

Processing Time on NVIDIA A100. We repeat the experiments in Sec. 4.4 on an NVIDIA A100 GPU. As shown in Fig. 8b, QL consistently reduces processing time even on this high-end accelerator. These results indicate that QL is not only effective for on-device training, but also offers a general mechanism for eliminating redundant backpropagation computation across different hardware platforms.

TinyLlama on BoolQ. We further extend our evaluation to a language task. Specifically, we fine-tune TinyLlama-1B on the BoolQ dataset (Clark et al., 2019), and compare QL_ρ against vanilla backpropagation under two budgets, $\rho \in \{0.9, 0.8\}$. As reported in Tab. 9, QL is able to reduce the fraction of trainable parameters to as low as 10.6%, while saving up to 89% of FLOPs, with only a marginal drop in accuracy compared to vanilla backpropagation. Notably, under $\rho = 0.8$, QL reaches 64.8% accuracy compared with 64.5% for vanilla while using only 21% of the FLOPs, providing initial evidence that QL extends beyond vision models to language-model fine-tuning.

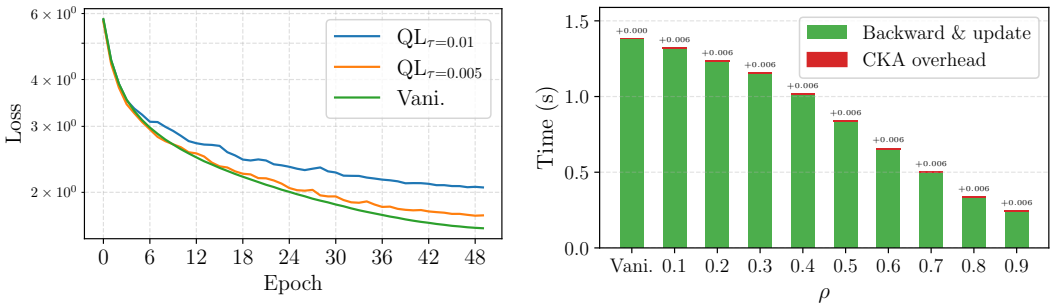
Comparison with Other Baselines. PipeTransformer was originally designed for distributed multi-GPU training. For a controlled single-device comparison, we adapt its freezing policy while removing the distributed pipeline and data-parallel components.

Comparison with Reference-based Methods. Since Egeria, SMARTFRZ, and SpFDE cannot be faithfully reproduced because public implementations are unavailable. To quantify the overhead associated specifically with maintaining a separate reference model, we construct a controlled proxy termed REFERENCE PASS. Every Δt training epochs, we release the activations, gradient buffers, and backward workspace, load a separate reference model into memory, perform a single inference pass, and release the reference model afterward. We compare its memory and latency overheads with those of QL.

We train ViT-B/32 on Flowers102 for 10 epochs with $\Delta t = 2$. Experiments are conducted on a single NVIDIA A100 GPU and a dual-socket AMD EPYC 9124 CPU server with 32 physical cores and 64 logical threads. We evaluate both CPU and GPU execution to characterize the overhead across hardware backends. For CPU experiments, memory usage is measured using the

Table 8: Pretraining on ImageNet-1K with QL_τ on ResNet18. We report the fraction of trainable parameters (%Train), average per-epoch FLOPs relative to vanilla backpropagation (δ_{FLOPs}), and Top-1 accuracy.

METH.	τ	%TRAIN	$\delta_{\text{FLOPs}} \downarrow$	ACC.↑
QL	0.01	31.8	0.42	49.0
	0.005	65.6	0.68	53.8
VANI.	–	100	1.00	55.7



(a) (b)

Figure 8: **(a)** Training loss across epochs for QL and vanilla backpropagation when pretraining ResNet18 from scratch on ImageNet-1K. **(b)** Average backward time per epoch to fine-tune MCUNet on CIFAR-10 with QL under different ρ on an NVIDIA A100 with $\Delta t = 10$.

process resident set size obtained from `/proc/self/status`; for GPU experiments, we use `torch.cuda.max_memory_allocated()`.

The latency of REFERENCE PASS includes loading the reference model from storage, transferring it to the target device, performing one inference pass, and releasing the model afterward. As shown in Table 10, REFERENCE PASS incurs substantially higher latency than QL, particularly on GPU. QL computes its stability metric directly from the training model already resident in memory, whereas REFERENCE PASS requires repeatedly loading and executing a separate model. Importantly, this comparison accounts only for the cost of the reference-model pass and excludes any additional layer-selection computation required by a complete freezing method.

C LIMITATION

QL relies on periodic CKA measurements from a mini-batch, which may become noisy in highly non-stationary or very small-data regimes as shown in experiments in Sec. 4.2. Its measurement overhead can also become less negligible for small layers or short training runs. Finally, QL operates at layer granularity, which limits the precision with which arbitrary parameter budgets can be matched.

Table 9: Fine-tuning TinyLlama-1B on BoolQ with QL_ρ under two budgets $\rho \in \{0.9, 0.8\}$, compared with vanilla backpropagation.

METH.	ρ	%TRAIN	$\delta_{\text{FLOPS}} \downarrow$	ACC.↑
QL	0.9	10.6	0.11	63.2
	0.8	21.2	0.21	64.8
VANI.	–	100	1.00	64.5

Table 10: Memory and latency overhead of QL and the reference-model baseline. PEAK MEM. denotes the maximum memory usage during training. RAW LAT. denotes the latency of a single overhead event, while AVG. LAT. denotes the latency amortized over all training steps.

METHOD	PEAK MEM. (MB)	RAW LAT. (MS)	AVG. LAT. (MS)
VANILLA (CPU)	6928.70	0	0
QL (CPU)	6928.70	6079.92	434.28
REFERENCE PASS (CPU)	6928.70	7100.76	507.20
VANILLA (GPU)	4420.95	0	0
QL (GPU)	4420.95	66.66	4.76
REFERENCE PASS (GPU)	4420.95	1558.32	111.31