Re-Key-Free, Risky-Free: Adaptable Model Usage Control

Zihan Wang*†◇, Zhongkui Ma*, Xinguo Feng*, Chuan Yan*, Dongge Liu§, Ruoxi Sun†, Derui Wang†, Minhui Xue†¶, Guangdong Bai‡
*The University of Queensland, Australia
†CSIRO's Data61, Australia
‡City University of Hong Kong, China
§Google LLC, Australia
¶Responsible AI Research (RAIR) Centre, Adelaide University, Australia

***Abstract*—Deep neural networks (DNNs) have become valuable intellectual property of model owners, due to the substantial resources required for their development. To protect these assets in the deployed environment, recent research has proposed model usage control mechanisms to ensure models cannot be used without proper authorization. These methods typically lock the utility of the model by embedding an access key into its parameters. However, they often assume static deployment, and largely fail to withstand continual post-deployment model updates, such as fine-tuning or task-specific adaptation.**

In this paper, we propose ADALOC, to endow key-based model usage control with *adaptability* during model evolution. It strategically selects a subset of weights as an intrinsic access key, which enables all model updates to be confined to this key throughout the evolution lifecycle. ADALOC enables using the access key to restore the keyed model to the *latest* authorized states without redistributing the entire network (*i.e.,* adaptation), and frees the model owner from full re-keying after each model update (*i.e.,* lock preservation). We establish a formal foundation to underpin ADALOC, providing crucial bounds such as the errors introduced by updates restricted to the access key. Experiments on standard benchmarks, such as CIFAR-100, Caltech-256, and Flowers-102, and modern architectures, including ResNet, DenseNet, and ConvNeXt, demonstrate that ADALOC achieves high accuracy under significant updates while retaining robust protections. Specifically, authorized usages consistently achieve strong task-specific performance, while unauthorized usage accuracy drops to near-random guessing levels (*e.g.,* 1.01% on CIFAR-100), compared to up to 87.01% without ADALOC. This shows that ADALOC can offer a practical solution for adaptive and protected DNN deployment in evolving real-world scenarios.

I. INTRODUCTION

Major AI and cloud providers such as Google, Microsoft, and Amazon have increasingly emphasized hardware-backed execution as a foundational component of secure AI deployment. For example, Google's recent "Private AI Compute" places models such as Gemini within TPU-based enclaves, which harden the computational environment against external compromise [1]. Although these trusted execution infrastructures substantially improve platform security, they secure only the runtime environment and not the model artifact itself. Once a high-value model is distributed, whether through licensing, integration into downstream pipelines, or deployment on customer-managed infrastructure, a fully functional copy can leave the enclave boundary. At that point, intellectual-property control becomes structurally fragile because unrestricted copies may propagate across organizations or jurisdictions, and dishonest parties can continue operating, reselling, or embedding the model at full fidelity [2]–[5]. Such leakage compromises the model owner's competitive advantage, devalues the substantial computational and data investment embodied in the model, and exposes an inherent limitation in relying solely on hardware isolation or access control. In summary, hardware-centric protections do not mitigate misuse once the model is no longer confined to the trusted platform.

This gap indicates a fundamental requirement that goes beyond enclave-style infrastructure, namely a *model-intrinsic mechanism for usage control* that can restrict a model's utility even when the hardware boundary is no longer present. Our work adopts this perspective by regulating the functional capacity of the model itself rather than relying solely on control over the execution environment. We demonstrate that hardware enclaves and model-level usage control are complementary layers. Enclaves provide secure storage and execution of secret material, while usage-control mechanisms ensure that the model remains nonfunctional without proper authorization. This layered view forms the central motivation of our work and establishes ADALOC as a missing component in the contemporary AI model-protection stack.

Recent research has approached model usage control through *model keying* [2], [6]–[9]. It ensures that without the correct access key,[1] the accuracy of the model collapses to near-random guessing, making unauthorized use effectively worthless. Data-based keying mechanisms, such as Pyone *et al.* [6] and Chen *et al.* [7], embed a secret key into the training data during preprocessing, and train the model to operate only on *key-preprocessed* inputs. Model-based mechanisms, such as Xue *et al.* [8] and Zhou *et al.* [2] embed the key directly into internal model parameters, undermining the utility of the entire model unless the key is known by the model controller for neutralization. These approaches mainly target the *accessibility* aspect of model usage control, and have demonstrated their effectiveness in locking model unity in *static* deployment. However, they largely assume the model

◇ Zihan Wang is supported by the Google PhD Fellowship.

[1]Throughout this paper, the term *key* denotes a usage-control primitive rather than a cryptographic key.

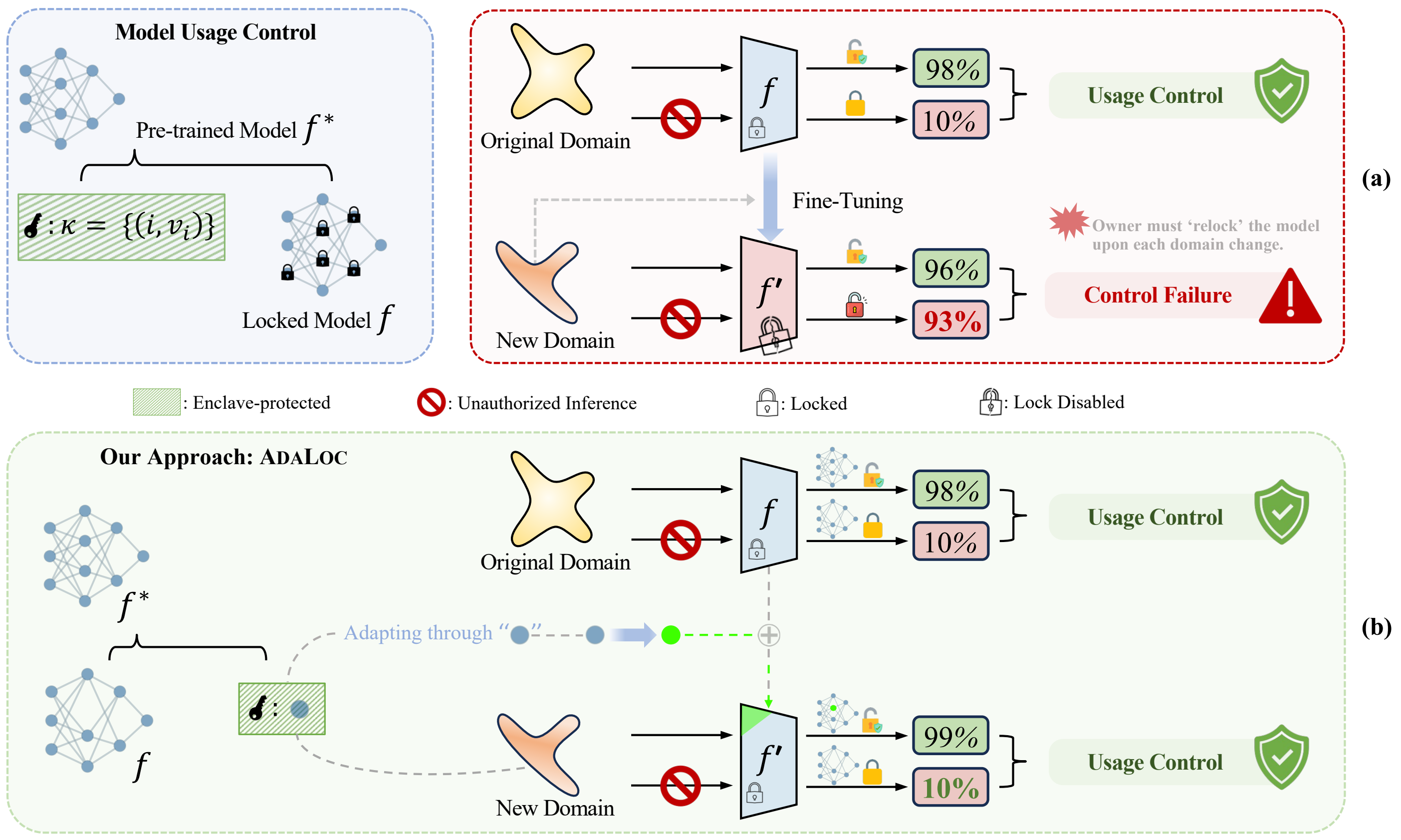

Fig. 1: **General model usage control (a).** *Locking*: a pretrained network f^* is converted to a locked model f plus a securely stored *key* κ (selected weight indices and values). Authorized inference returns normal outputs; unauthorized inference renders the model completely unusable (*e.g.,* random-guess level for 10-class classification task). *Update & failure*: adapting the model to a new domain ($f \to f'$) invalidates the lock, restoring full access for attackers and forcing costly re-keying and model redistribution. **Our approach (b).** Instead of relying on any "external" locking mechanism, we designate a compact block of neurons themselves as κ, and restrict all updates to that block. Because the rest of the network never changes, the lock remains intact and f' is still unusable without the *key*.

parameters remain unchanged after deployment.

This assumption fails to hold in practical machine learning workflows, where models are frequently fine-tuned or incrementally updated to adapt to new tasks, data distributions, or environments. Indeed, a recent study [10] reports that 44% of organizations piloting AI models in 2024 relied on frequent updates, often weekly or monthly, to fine-tune established backbones (*e.g.,* Vision Transformers [11]). Since even modest amounts of fine-tuning can significantly alter the parameter space, continual model refinement can easily overwrite the protection provided by existing keying schemes (see Figure 1(a)), making costly "re-keying" necessary. This reveals a fundamental limitation in existing model keying mechanisms: *accessibility* is not complemented by *adaptability*. An important research problem remains largely unexplored: *how to enable model usage control under rapid and continual adaptation cycles*?

Our work. We develop an adaptable model usage control framework that unifies accessibility and adaptability, named AdaLoc (adaptable lock). As illustrated in Figure 1(b), it effectively ensures *(i)* that model usability is strictly conditioned on possessing the *access key*, thereby blocking unauthorized usage, and *(ii)* that authorized users can efficiently update the keyed model, eliminating the costly need to redistribute the entire network after each update. During model updates, AdaLoc confines the updates exclusively to the *access key* (hereafter, the *key*), avoiding retraining the model or altering other parts of the model. At deployment, restoring the *key* reinstates the network's core functionality, enabling robust performance on newly introduced tasks or domains. AdaLoc is both *training data-agnostic* and *retraining-free*, making it directly applicable to off-the-shelf pre-trained models and easily integrated across diverse architectures.

AdaLoc adopts a simple yet effective strategy that leverages a minimal subset of the model's high-impact neurons as the *key*. This solution builds on the fundamental tendency known as *impact concentration* [12], [13], where a network's predictive power consistently hinges on a small set of decisive weights. The minimal subset ensures the *key* is compact and easily manageable (*i.e., key scalability*). Through AdaLoc, the model owner adapts the model across updates by modifying only these *key* weights without touching others (*i.e., effective domain adaptation*). Removing the *key* induces an accuracy drop to a completely unusable level, whereas reinserting it revives full performance in a single step (*i.e., restoration disparity*). Moreover, reconstructing the missing weights is computationally infeasible (*i.e., complex unauthorized restoration*). It may take an IBM Summit supercomputer millions of years to crack a multi-layer network [14].

To establish a solid formal foundation for AdaLoc, we for the first time formalize a comprehensive notion of model usage control that integrates accessibility and adaptability. Accessibility ensures that a locked model f behaves like an ideally unusable model f^0 in the absence of a valid *key* (*i.e.,* $f \approx f^0$), and adaptability requires this guarantee to persist after any authorized model update. We establish crucial bounds for these two properties, which define the allowable parameter changes among the original model f^*, the locked model f, and the *key*-only updated model f', in order to guarantee that the model is unusable without the *key* and returns to full performance once the *key* is restored. Using layer-wise Lipschitz bounds and sub-Gaussian tail bounds, we bound the exact region in weight-space where a *key*-only update can move while the network's output stays indistinguishable from full fine-tuning. Conversely, we prove that removing the *key* part collapses the network's output variance to that of an unusable reference model f^0.

We evaluate AdaLoc on a wide range of image and text benchmarks using standard convolutional and transformer backbones: DenseNet-121, ResNet-152, and ConvNeXt-V2 for vision tasks, and RoBERTa, BERT, and DeBERTa for language tasks. The evaluation address two questions: *(i)* can a model adapt when updates are confined to the *key*, and *(ii)* does AdaLoc's usage control remain effective across datasets and architectures. Our results demonstrate that in every benchmark, the authorized model matches or even exceeds full fine-tuning accuracy with the *key*, while removing the *key* drives accuracy to near-random levels, such as 1.02% on CIFAR-100 and 0.47% on Caltech-256. All existing schemes still expose around 85.14% accuracy on average under the same setup. These results confirm that AdaLoc enables lightweight continual adaptation while enforcing strict usage control.

Contributions. We summarize our main contributions as follows:

- **An adaptable usage control paradigm.** We introduce the concept of *adaptable* model usage control, which enables frequent updates (*e.g.,* fine-tuning) to the keyed model while preserving the validity and functionality of the original keying mechanism. This paradigm bridges the gap between *adaptability* and *accessibility*, addressing scenarios where models must evolve while retaining secure usage restrictions.
- **A solid formal foundation for usage control.** We establish a formal foundation for analyzing model usage control by, for the first time, formalizing a comprehensive notion that unifies *accessibility* and *adaptability*. We further derive crucial theoretical bounds that guarantee the efficacy of AdaLoc, providing provable assurances for secure and adaptable AI model usage.
- **An empirical evaluation.** We evaluate AdaLoc on representative datasets and real-word models. Across all settings, AdaLoc *consistently* delivers performance fully comparable to full fine-tuning for authorized users, while models without the *key* degrade to a completely unusable level. These results demonstrate that AdaLoc enforces robust usage control while supporting seamless model adaptation across diverse datasets and architectures, making usage control practical in modern machine learning workflows.

TABLE I: Main notations used in this paper.

Notation	Description
$\mathcal{D}^*$	Original dataset. The superscript * indicates the original domain. $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ is a data-label pair from $\mathcal{D}^*$.
$\hat{\mathcal{D}}$	New dataset used for fine-tuning. $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$ is a data-label pair sampled from $\hat{\mathcal{D}}$.
$\boldsymbol{\theta}^*$	Post-training parameters on $\mathcal{D}^*$, producing output $\boldsymbol{y}^*$.
$\hat{\boldsymbol{\theta}}$	Parameters obtained by *full* fine-tuning on $\hat{\mathcal{D}}$.
$\tilde{\boldsymbol{\theta}}$	Parameters obtained by *key-only* (partial) fine-tuning.
$\square_\ell$	The *locked* version of an object, *e.g.,* $\boldsymbol{\theta}^*_\ell$.

II. Background

This section introduces the foundational concepts required to understand this work. Section II-A formalizes neural network training, and Section II-B formulates the problem of model usage control.

A. Neural Networks

To facilitate understanding, Table I lists the main notations used throughout the paper. Unless otherwise stated, a symbol without superscripts denotes a generic case.

A neural network is represented as a function $f(\boldsymbol{x}; \boldsymbol{\theta})$, where $\boldsymbol{x} \in \mathbb{R}^m$ is the input and $\boldsymbol{\theta} \in \mathbb{R}^d$ denotes the parameters; m and d are the input and parameter dimensions, respectively. The choice of $\boldsymbol{\theta}$ governs the model's behavior and performance. We use $\boldsymbol{y}^*$ to denote the target output for a given input $\boldsymbol{x}^*$. Because our focus is on training, we assume inputs and outputs originate from the training distribution. Below we formalize model training and model functionality.

Definition 1 (Parameter Space)**.** *The* parameter space *of $f(\boldsymbol{x}; \boldsymbol{\theta})$ is the d-dimensional set $\{\boldsymbol{\theta} \in \mathbb{R}^d\}$; each point specifies a unique parameter configuration.*

Definition 2 (Model Training)**.** Model training *updates an initial parameter vector $\boldsymbol{\theta}$ to $\boldsymbol{\theta}^*$ by minimizing a loss function over a sequence of steps.*

Definition 3 (Parameter Update)**.** *The* parameter update *is $\Delta\boldsymbol{\theta}^* = \boldsymbol{\theta}^* - \boldsymbol{\theta}$, the difference between trained and initial parameters.*

Training halts once the loss $\mathcal{L}(f(\boldsymbol{x}^*; \boldsymbol{\theta}), \boldsymbol{y}^*)$ drops below a threshold or stops improving significantly. The resulting parameters are typically locally optimal yet adequate for practical metrics such as accuracy or F1-score.

Definition 4 (Model Functionality)**.** *Given $\mathcal{D}^*$, a model $f(\boldsymbol{x};\boldsymbol{\theta}^*)$ satisfies* functionality *if*

$$\mathcal{M}(f(\boldsymbol{x}^*;\boldsymbol{\theta}^*),\boldsymbol{y}^*) \leq \epsilon, \quad \forall(\boldsymbol{x}^*,\boldsymbol{y}^*) \in \mathcal{D}^*, \tag{1}$$

where $\mathcal{M}$ is a non-negative performance metric and ϵ is a small positive constant.

Below we formulate the problem of model usage control, focusing on key-based mechanisms central to our approach.

B. Model Usage Control

Model usage control protects a model from unauthorized exploitation. Commonly, a *key* specifies a subset of parameters whose values gate access.

Definition 5 (Key)**.** *A* key *κ^* is a finite set of index-value pairs*

$$\kappa^* = \{(i, v_i^*) \mid i \in \mathcal{S}^*,\, v_i^* \in \mathbb{R}\}, \tag{2}$$

where $\mathcal{S}^* \subseteq \{1,\ldots,d\}$ indexes the protected parameters.

Each index-value pair (i, v_i^*) specifies that the parameter θ_i^* takes the value v_i^* prior to locking the model with *key* κ^*. We denote by $\mathcal{K}$ the set of all possible keys. A valid *key* must satisfy the following properties:

- **Compactness**: The *key* covers only a minimal subset of parameters (Section II-B1), making it lightweight and suitable for practical deployment.
- **Unauthorized-use prevention**: Without the *key*, the model's performance degrades and becomes unusable (Section II-B2).
- **Restoration**: With the *key*, the model regains its original functionality (Section II-B3).

Fulfilling these requirements involves three sequential phases: *(i)* identifying sensitive parameters, *(ii)* locking the model by perturbing them, and *(iii)* unlocking the model by reinstating the stored values. We detail each phase below.

1) Key Localization: Key localization aims to identify a small subset of model parameters that are highly sensitive to perturbations, such that modifying them alone can significantly affect model behavior. These sensitive weights form the basis for constructing an effective *key*. The goal is to select a minimal set $\mathcal{S}^* \subseteq \{1,\ldots,d\}$, with $|\mathcal{S}^*| \ll d$, to ensure the resulting *key* remains compact and efficient while retaining strong control over model access.

2) Model Locking: To lock the model, the transformation

$$\Phi : \mathbb{R}^d \times \mathcal{K} \longrightarrow \mathbb{R}^d, \quad \boldsymbol{\theta}_\ell^* = \Phi(\boldsymbol{\theta}^*, \kappa^*), \tag{3}$$

is applied to produce the locked parameter vector $\boldsymbol{\theta}_\ell^*$ from the original weights $\boldsymbol{\theta}^*$ and *key* κ^*. For every protected index $i \in \mathcal{S}^*$, the weight θ_i^* is replaced by a *carefully crafted perturbation* that significantly degrades performance. Consequently, the locked model obeys

$$\mathcal{M}\big(f(\boldsymbol{x}^*;\boldsymbol{\theta}_\ell^*),\boldsymbol{y}^*\big) \;\geq\; M,$$

with M sufficiently large to render the model unusable without the *key*. Supplying the correct *key* invokes the inverse transformation, restores the original parameters, and reinstates full functionality.

3) Model Unlocking: Unlocking reverses the locking transformation with the same key. Let

$$\Psi : \mathbb{R}^d \times \mathcal{K} \longrightarrow \mathbb{R}^d, \quad \boldsymbol{\theta}^* = \Psi(\boldsymbol{\theta}_\ell^*, \kappa^*), \tag{4}$$

where Ψ restores, for every $i \in \mathcal{S}^*$, the original weight value v_i^* that was replaced during locking. The recovered parameter vector $\boldsymbol{\theta}^*$ reinstates full model functionality, enabling normal inference.

III. Problem Formulation

Figure 1 exposes a critical weakness in current usage control schemes. Locking a pre-trained network f^* with a *key* κ succeeds only while the model stays unchanged. Once updated (*e.g.,* fine-tuned), whether to add new features, serve a new client, or track shifting data, it becomes f', and even small weight changes can invalidate the original lock. Unauthorized users then regain full predictive power, forcing the owner to re-key the model and repeat the costly distribution and deployment process. In rapidly evolving production environments, where models may update daily or even hourly, this break-and-rekey cycle is unsustainable. A usage-control scheme whose *key* remains effective across routine updates is therefore indispensable.

A. Threat Model

This section outlines the scope of AdaLoc and specifies the attacks it defends against, in terms of the information adversaries possess and the operations they can perform on the obtained model.

We consider a model controller who deploys a high-value neural network and may use trusted infrastructure (*e.g.,* hardware-backed enclaves or TEEs) to manage access keys during normal operation. AdaLoc is designed to complement such platform-level protections by ensuring that, even if the locked model itself is leaked outside these environments, it remains unusable without the *key*.

The attacker seeks to acquire a complete copy of the locked neural-network model under the control of the model controller. Possession of such a model enables various abuses, such as monetizing it through unauthorized commercial services or generating adversarial examples [15], [16] against the model owner's legitimate offerings. The focus of our work is to endow model usage control with *adaptability* during model evolution, as defined in Section I, so that this protection persists across continual updates. Access keys themselves can be managed and released by hardware-assisted mechanisms [17] or TEE-based systems [9], [18], which we treat as trusted components responsible for key storage and release.
Attacker capabilities. The attacker is given *white-box* access to the leaked locked model: they can inspect and manipulate all weight parameters, and they know the exact network architecture used during training. This represents a conservative

assumption that favors the attacker and is realistic, since most industrial systems adopt well-published DNN designs with proven performance. The attacker may also perform arbitrary post-processing on the leaked model, including fine-tuning, distillation, pruning, or inserting adapters, using their own compute resources.

We assume the attacker has only limited access to valid training data drawn from the same distribution as the original training data; otherwise, they could simply train a competitive model themselves. However, they are free to combine this limited data with the leaked locked model in any way they choose. We do *not* consider attacks that compromise the underlying hardware or TEE itself (*e.g.,* breaking the enclave or extracting keys from a hardware root of trust); such attacks are orthogonal to our goal and are typically addressed by the platform-security layer. Under this threat model, AdaLoc aims to ensure that the leaked model remains practically unusable without the *key*, even under white-box access and adaptive post-processing.

B. Usage Control for Adaptive AI Systems

When a locked model is fine-tuned, its weights drift away from the version used to generate the original *key*. If the *key* is regenerated every time, deployment becomes expensive and fragile. The challenge is therefore to maintain a *single, compact key* that *(i)* continues to disable the model after *any* update, and *(ii)* still restores full utility for authorized users.

Formally, let $f(\boldsymbol{x}^*;\boldsymbol{\theta}^*)$ be a locked model with *key* κ^* protecting indices $\mathcal{S}^*$. After full fine-tuning we obtain parameters $\hat{\boldsymbol{\theta}}$, as well as the outputs $\hat{\boldsymbol{y}}$ on the new dataset $\hat{\mathcal{D}}$. We seek an *key* $\hat{\kappa}$ such that:

Problem (Adaptable usage control)**.** *Requiring that the performance metric $\mathcal{M}(f(\hat{\boldsymbol{x}};\hat{\boldsymbol{\theta}}),\hat{\boldsymbol{y}}) \leq \epsilon$, where $(\hat{\boldsymbol{x}},\hat{\boldsymbol{y}}) \in \hat{\mathcal{D}}$, we aim to find a key $\hat{\kappa}$ such that*

- *The model remains unusable without the new key $\hat{\kappa}$, i.e., $\mathcal{M}(f(\hat{\boldsymbol{x}};\hat{\boldsymbol{\theta}}_\ell),\hat{\boldsymbol{y}}) \geq M$;*
- *The model functionality can be fully restored with the new key $\hat{\kappa}$ presented, i.e., $\mathcal{M}(f(\hat{\boldsymbol{x}};\Psi(\hat{\boldsymbol{\theta}}_\ell,\hat{\kappa})),\hat{\boldsymbol{y}}) \leq \epsilon$.*

Solving this problem is challenging, as it demands maintaining a stable *key* structure while accommodating continuous model evolution. In the following section, we introduce AdaLoc, our adaptable usage-control framework designed specifically to meet these requirements.

IV. Our Approach: AdaLoc

Modern deep networks typically contain more parameters than necessary. Extensive pruning studies consistently reveal that a small subset of weights significantly determines model behavior, while the remaining parameters contribute minimally [19]–[21]. AdaLoc leverages this intrinsic property by keeping the majority of parameters untouched, and restricting updates to a *carefully selected, small subset* (designated as the *key*) whenever the model needs to be fine-tuned. Specifically, the *key* must satisfy two critical criteria:

(*i*) Removal of the *key* drastically reduces model accuracy.

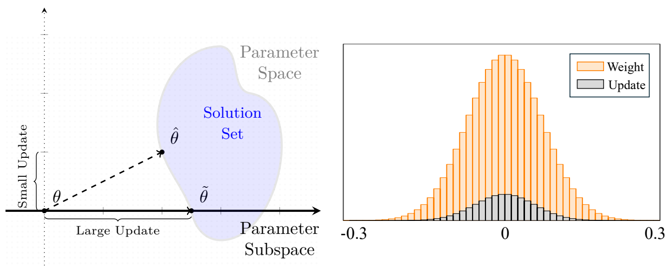

Fig. 2: Heuristic illustration. Updating only a few directions with the largest parameter changes, spanning a subspace $\tilde{\boldsymbol{\theta}}$, can often reach the same solution set as full fine-tuning $\hat{\boldsymbol{\theta}}$. These high-impact directions generally align with large-magnitude weights (see the histogram in Figure 4). Theorems 2-5 provide formal guarantees for both observations, and Section VI empirically confirms them.

(*ii*) Updating the *key* alone achieves performance fully comparable to full fine-tuning.

Both criteria naturally point toward selecting high-magnitude weights. Such weights significantly influence forward activations and typically accumulate larger gradient updates. Moreover, removing these influential weights generally *incapacitates* the model, directly satisfying condition *(i)*. Fine-tuning within this lower-dimensional subspace ($\tilde{\boldsymbol{\theta}}$) effectively steers the model towards the practical solution region near the fully tuned parameter vector $\hat{\boldsymbol{\theta}}$ (please refer to the heuristic illustration in Figure 2).

A. Key Localization

AdaLoc thus adopts a straightforward yet effective heuristic by ranking parameters according to their ℓ_1-norm and selecting the top $\rho\%$ (default $\rho = 5$) as the *key*. The chosen indices form the set $\mathcal{S}$, with their initial parameter values constituting the initial adaptation *key*.

The ℓ_1-norm efficiently captures the magnitude of parameters and their relative importance in influencing network predictions. This method is computationally lightweight, inherently data-agnostic, and broadly applicable across various network architectures. Figure 3 visually demonstrates the efficacy of this approach: filters exhibiting higher ℓ_1-norms generally encode richer and more diverse features, whereas filters with lower magnitudes tend to capture narrower or redundant information. Thus, selecting $\mathcal{S}$ as a concise yet influential subset of parameters effectively supports impactful model adaptation.

Formally, the process of extracting the top n filters or neurons based on the ℓ_1-norm from the i-th layer involves the following steps: *(i)* compute the sum of absolute weights for each filter or neuron; *(ii)* rank them in ascending order according to their computed sums; *(iii)* identify and remove the n filters or neurons with the highest sum values, along with their corresponding feature maps; *(iv)* subsequently, remove filters or neurons in the following layer connected to the discarded feature maps. Filters and neurons identified and

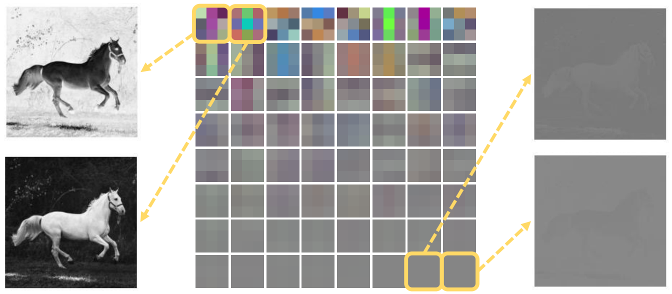

Fig. 3: Visualization of feature maps (the top and bottom two) and corresponding filters (all 64 filters) from the first convolutional layer of a trained VGG model.

removed through this method are cataloged in the *key*, clearly marking the critical parameters required for efficient and controlled model updates.

B. Adapting through the Key

After localizing the *key* with index set $\mathcal{S}$, AdaLoc updates the model exclusively through these parameters.

Given an initial parameter vector $\boldsymbol{\theta}^*$ and a new dataset $\hat{\mathcal{D}} = \{(\hat{\boldsymbol{x}}^{(i)}, \hat{\boldsymbol{y}}^{(i)}) \mid 1 \leq i \leq n\}$, the *key* parameters are fine-tuned by minimizing the loss function $\mathcal{L}$ over the dataset to adapt the model to the new task. Specifically,

$$\tilde{\boldsymbol{\theta}} = \underset{\{\theta_i^* | i \in \mathcal{S}\}}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(\hat{\boldsymbol{x}}^{(i)}; \boldsymbol{\theta}^*), \hat{\boldsymbol{y}}^{(i)}),$$

where $\tilde{\boldsymbol{\theta}}$ represents the partially updated parameters.

During each optimization step t, we restrict parameter updates to the coordinates in the adaptation set $\mathcal{S}$. For every $j \in \mathcal{S}$, the parameters are updated according to $\hat{\theta}_j^{(t+1)} = \tilde{\theta}_j^{(t)} - \eta \frac{\partial}{\partial \tilde{\theta}_j^{(t)}} [\frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(\hat{\boldsymbol{x}}^{(i)}; \tilde{\boldsymbol{\theta}}^{(t)}), \hat{\boldsymbol{y}}^{(i)})]$, where η is the learning rate. All other weights remain at their pre-trained values, *i.e.*, $\hat{\theta}_j^{(t+1)} = \tilde{\theta}_j^*$ for $j \notin \mathcal{S}$.

In practice, the goal is usually to meet a specified performance threshold rather than minimize the loss to absolute optimality. Thus, we define the practical solution set as the parameter configurations sufficiently close to the fully fine-tuned optimal parameters $\hat{\boldsymbol{\theta}}$ as

$$\mathcal{N}(\hat{\boldsymbol{\theta}}) = \{\boldsymbol{\theta}' \mid \|\boldsymbol{\theta}' - \hat{\boldsymbol{\theta}}\| \leq \epsilon\}, \quad (5)$$

where $\epsilon > 0$ represents an acceptable performance deviation. Partial updates are effective when the adapted parameters $\tilde{\boldsymbol{\theta}}$ remain within this practical solution set. To quantify how closely such partial solutions approximate full fine-tuning, we measure

$$\|\hat{\boldsymbol{\theta}} - \tilde{\boldsymbol{\theta}}\| = \sqrt{\textstyle\sum_{i \notin \mathcal{S}} (\hat{\theta}_i - \tilde{\theta}_i)^2},$$

which explicitly captures the impact of frozen parameters on adaptation effectiveness and clarifies the trade-off between updating fewer parameters and maintaining strong performance. Subsequent sections theoretically (Section V) and empirically (Section VI) show that AdaLoc's *key*-updated parameters $\tilde{\boldsymbol{\theta}}$ remain within this practical solution set, demonstrating both robust authorization control and strong adaptation performance.

Remark: AdaLoc *is highly usable and deployable*

By designating a small set of the model's own neurons as the access key, AdaLoc *enables parameter updates to occur entirely within this subset. This allows the lock to persist across continual adaptation, satisfying the model owner's need for adaptability without sacrificing the integrity of the control mechanism. In practice, this makes it possible to share, update, and license models securely and efficiently in dynamic, real-world environments.*

V. Theoretical Analysis

This section provides formal guarantees that a *key* produced by AdaLoc satisfies two essential properties: *(i) model accessibility*, ensuring the model becomes unusable without the *key* (Section V-A), and *(ii) model adaptability*, where updating only *key*-indexed weights yields performance comparable to full fine-tuning (Section V-B).

Design sketch. The proof proceeds in four logical steps. First, Theorem 1 shows that removing the highimpact weights selected by the *key* significantly collapses output variance, pushing the network toward a constantoutput reference model f^0 with null utility. Next, Theorems 2 and 3 establish that, provided the *key*updated parameters stay within explicit distance (or standarddeviation) bounds, the model's predictions match those of a fully finetuned counterpart. Theorems 4 and 5 then explain why parameters outside the *key* remain stable: their smaller incoming weight norms give them proportionally smaller accumulated gradients, so they drift negligibly during adaptation. Finally, Section V-C empirically verify that measured parameter gaps lie well inside these theoretical bounds, confirming the practical tightness of our formal foundation.

Throughout the proofs, we adopt the following norm conventions: For vectors, $\|\boldsymbol{x}\| = \sqrt{\sum_i^n x_i^2}$ where $\boldsymbol{x} \in \mathbb{R}^n$ denotes the Euclidean norm; for matrices $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\|\boldsymbol{A}\| = \max_{\|\boldsymbol{x}\|=1} \|\boldsymbol{A}\boldsymbol{x}\|$ is the spectral norm; and for random variables X, the sub-Gaussian norm is $\|X\|_{\varphi_2} = \inf\{c > 0 : \mathbb{E}\exp(X^2/c^2) \leq 2\}$ [22].

A. Model Accessibility

A network is unusable if it produces almost constant outputs and cannot discriminate inputs. We formalize this by introducing an *ideally locked* reference model.

Reference model f^0. Let f^0 be obtained from a pre-trained network f by zeroing all weights:

$$f^0(\boldsymbol{x}) = \boldsymbol{c}, \quad \forall \boldsymbol{x}, \quad (6)$$

where $\boldsymbol{c}$ equals the bias of the last layer. When the test set contains uniformly distributed classes, the accuracy of f^0 approaches the reciprocal of the class count; in practice, its performance resembles random guessing.

The following theorem bounds output variance in terms of parameter variance, showing that shrinking a small set of high-magnitude weights drives the network toward f^0.

Theorem 1 (Output variance bounded by parameter variance)**.** *Let $f(\boldsymbol{x};\boldsymbol{\theta})$ be a fully connected neural network with*

- *L hidden layers and N neurons per layer;*
- *activation function σ that is Lipschitz continuous,* i.e., *$|\sigma(a)-\sigma(b)| \le B_L|a-b|$ for all $a,b \in \mathbb{R}$;*
- *weight matrices $\boldsymbol{W}^{(m)} \in \mathbb{R}^{N\times N}$ and bias vectors $\boldsymbol{b}^{(m)} \in \mathbb{R}^N$ for $m=1,\dots,L$.*

For the m-th layer, where $m=1,\dots,L$, assume that[2]

(i) the weights $\{W_{ij}^{(m)}\}$ are i.i.d., have zero mean, and common variance $\mathrm{Var}(\boldsymbol{W}^{(m)})$;

(ii) the biases $\{b_i^{(m)}\}$ are i.i.d., have zero mean, and common variance $\mathrm{Var}(\boldsymbol{b}^{(m)})$;

Then, for any deterministic input $\boldsymbol{x}\in\mathbb{R}^N$, $\mathrm{Var}(f(\boldsymbol{x};\boldsymbol{\theta}))$ is upper bounded by

$$\begin{aligned}&\|\boldsymbol{x}\|_2^2(B_L^2N)^L\prod_{i=1}^{L}\mathrm{Var}(\boldsymbol{W}^{(i)}) + B_L^2N\cdot\mathrm{Var}(\boldsymbol{b}^{(L)})\\ &+B_L^2\sum_{i=1}^{L-1}\{N\cdot\mathrm{Var}(\boldsymbol{b}^{(i)})\prod_{j=i+1}^{L}[B_L^2N\cdot\mathrm{Var}(\boldsymbol{W}^{(j)})]\}.\end{aligned} \tag{7}$$

Proof. We bound the output variance layer by layer, using the Lipschitz property of σ, and then propagate the bound forward. For the first hidden layer, we have $\boldsymbol{W}_i^{(1)}\boldsymbol{x}+b_i^{(1)}$ of linear transformation for one neuron. Because the weights and biases are *i.i.d.* with zero mean,

$$\begin{aligned}\mathrm{Var}(\boldsymbol{W}_i^{(1)}\boldsymbol{x}+b_i^{(1)}) &= \mathrm{Var}(\textstyle\sum_{j=1}^N W_{i,j}^{(1)}x_j + b_i^{(1)})\\ &= \mathrm{Var}(\textstyle\sum_{j=1}^N W_{i,j}^{(1)}x_j) + \mathrm{Var}(b_i^{(1)})\\ &= \textstyle\sum_{j=1}^N x_j^2\mathrm{Var}(W_{i,j}^{(1)}) + \mathrm{Var}(b_i^{(1)})\\ &\le \mathrm{Var}(\boldsymbol{W}^{(1)})\|\boldsymbol{x}\|_2^2 + \mathrm{Var}(\boldsymbol{b}^{(1)}).\end{aligned}$$

By the Lipschitz property of the activation function, $y_i^{(1)} = \sigma(\boldsymbol{W}_i^{(1)}\boldsymbol{x}+b_i^{(1)}) \le B_L(\boldsymbol{W}_i^{(1)}\boldsymbol{x}+b_i^{(1)})$, and then,

$$\begin{aligned}\mathrm{Var}(y_i^{(1)}) &\le \mathrm{Var}[L(\boldsymbol{W}_i^{(1)}\boldsymbol{x}+b_i^{(1)})]\\ &\le B_L^2\mathrm{Var}(\boldsymbol{W}_i^{(1)}\boldsymbol{x}+b_i^{(1)})\\ &\le B_L^2[\mathrm{Var}(\boldsymbol{W}^{(1)})\|\boldsymbol{x}\|_2^2+\mathrm{Var}(\boldsymbol{b}^{(1)})].\end{aligned}$$

Summing over the N neurons yields

$$\begin{aligned}\mathrm{Var}(\boldsymbol{y}^{(1)}) &= \textstyle\sum_{j=1}^N \mathrm{Var}(y_i^{(1)})\\ &\le B_L^2N[\mathrm{Var}(\boldsymbol{W}^{(1)})\|\boldsymbol{x}\|_2^2+\mathrm{Var}(\boldsymbol{b}^{(1)})].\end{aligned}$$

Similarly, for the m-th layer,

$$\mathrm{Var}(\boldsymbol{y}^{(m)}) \le B_L^2N^{(m-1)}[(\sigma_w^{(m)})^2\|\boldsymbol{y}^{(m-1)}\|^2+(\sigma_b^{(m)})^2],$$

and it concludes the theorem by iteratively applying the inequalities. □

[2]Our assumptions across this study are practical and commonly used in many theoretical works of neural networks (*e.g.*, pruning [12], [23]).

Remark: AdaLoc *ensures strong usage controllability*

By Theorem 1, AdaLoc's locking strategy sharply reduces the parameter variances $\mathrm{Var}(\boldsymbol{W}^{(m)})$. This, in turn, compresses the model's output variance and leads it to behave like the unusable reference model f^0 that produces nearly constant outputs, effectively blocking unauthorized usage.

B. Model Adaptability

We prove that updating only the AdaLoc's *key*-indexed coordinates retains the accuracy that full fine-tuning would achieve. Throughout this subsection, we continue to use $\hat{\boldsymbol{\theta}}$ to denote the fully fine-tuned parameter vector, $\tilde{\boldsymbol{\theta}}$ the *key-only* adaptation one, and $\epsilon>0$ the target accuracy tolerance.

1) Performance Under Small Parameter Distance: The following theorem establishes a distance-based sufficient condition under which the *key-only* updated model performs as well as the fully fine-tuned one.

Theorem 2 (Performance under bounded parameter distance)**.** *Let $f(\boldsymbol{x};\boldsymbol{\theta})$[3] be a pre-trained network and let $\hat{\boldsymbol{\theta}}$ be a* practical *solution obtained by fine-tuning on $\hat{\mathcal{D}}$. Write $\boldsymbol{\theta} = \{\boldsymbol{\theta}^{(1)},\dots,\boldsymbol{\theta}^{(L)}\}$ with $\boldsymbol{\theta}^{(l)}\in\mathbb{R}^{m^{(l)}\times n^{(l)}}$. Assume*

(i) The activation function $\sigma(\cdot)$ is Lipschitz continuous: $|\sigma(a)-\sigma(b)|\le B_\sigma|a-b|$;

(ii) The weight matrices $\hat{\boldsymbol{\theta}}^{(l)}$ are bounded by $\|\hat{\boldsymbol{\theta}}^{(l)}\| = s1(\hat{\boldsymbol{\theta}}^{(l)}) \le B_\theta$, where $s1(\cdot)$ is the maximum singular value of the matrix and $B_\theta>0$ is a constant; Each $\tilde{\boldsymbol{\theta}}^{(l)}$ follows a sub-Gaussian distribution.

(iii) Inputs obey $\|\boldsymbol{x}\|\le B_x$, where $B_x>0$ is a constant.

If the key update $\tilde{\boldsymbol{\theta}}$ obeys

$$\|\tilde{\boldsymbol{\theta}}-\hat{\boldsymbol{\theta}}\| \le \epsilon/(B_\sigma^{L-1}B_\theta^L B_x) - B_\sigma B_\theta, \tag{8}$$

then $\tilde{\boldsymbol{\theta}}$ falls within the practical solution set $\mathcal{N}(\hat{\boldsymbol{\theta}})$ in Formula (5), such that $\|f(\boldsymbol{x};\tilde{\boldsymbol{\theta}})-f(\boldsymbol{x};\hat{\boldsymbol{\theta}})\|\le\epsilon$ with a probability of

$$\Pr(\tilde{\boldsymbol{\theta}}\in\mathcal{N}(\hat{\boldsymbol{\theta}})) = (1-2\exp(-t^2))^L, \tag{9}$$

where the term $B_\theta = CK^{(l)}(\sqrt{m^l}+\sqrt{n^l}+t)$, $K^{(l)} = \max_{i,j}\|\boldsymbol{\theta}_{i,j}^{(l)}\|_{\varphi_2}$, and C is a universal constant.

Proof. We bound the prediction difference between the fully fine-tuned model $f(\boldsymbol{x};\hat{\boldsymbol{\theta}})$ and the *key-only* updated model $f(\boldsymbol{x};\tilde{\boldsymbol{\theta}})$ by recursively applying Lipschitz continuity and norm inequalities across layers. Consider

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{\theta}^{(L)}\sigma(\boldsymbol{\theta}^{(L-1)}\sigma(\cdots\sigma(\boldsymbol{\theta}^{(1)}\boldsymbol{x}))),$$

[3]The bias terms are merged into $\boldsymbol{\theta}$, and $\boldsymbol{\theta}=\{\boldsymbol{\theta}^{(1)},\boldsymbol{\theta}^{(2)},\dots,\boldsymbol{\theta}^{(L)}\}$ and $\boldsymbol{\theta}^{(l)}$ denotes the weight matrix of the l-th layer.

let $\boldsymbol{y}^{(l)} = \boldsymbol{W}^{(L)}\boldsymbol{x}^{(l)}$ and $\boldsymbol{x}^{(l)} = \sigma(\boldsymbol{y}^{l-1})$ for $l = 1, 2, \ldots, L$. Then, we have

$$\begin{aligned} &\|f(\boldsymbol{x};\tilde{\boldsymbol{\theta}}) - f(\boldsymbol{x};\hat{\boldsymbol{\theta}})\| \\ =& \|\tilde{\boldsymbol{y}}^{(L)} - \hat{\boldsymbol{y}}^{(L)}\| \\ =& \|\tilde{\boldsymbol{\theta}}^{(L)}\tilde{\boldsymbol{x}}^{(L)} - \hat{\boldsymbol{\theta}}^{(L)}\hat{\boldsymbol{x}}^{(L)}\| \\ =& \|(\tilde{\boldsymbol{\theta}}^{(L)} - \hat{\boldsymbol{\theta}}^{(L)})\hat{\boldsymbol{x}}^{(L)} + \hat{\boldsymbol{\theta}}^{(L)}(\tilde{\boldsymbol{x}}^{(L)} - \hat{\boldsymbol{x}}^{(L)})\| \\ \le& \|\tilde{\boldsymbol{\theta}}^{(L)} - \hat{\boldsymbol{\theta}}^{(L)}\|\|\hat{\boldsymbol{x}}^{(L)}\| + \|\hat{\boldsymbol{\theta}}^{(L)}\|\|\tilde{\boldsymbol{x}}^{(L)} - \hat{\boldsymbol{x}}^{(L)}\| \\ \le& \|\tilde{\boldsymbol{\theta}}^{(L)} - \hat{\boldsymbol{\theta}}^{(L)}\|\|\hat{\boldsymbol{x}}^{(L)}\| + B_\theta\|\tilde{\boldsymbol{x}}^{(L)} - \hat{\boldsymbol{x}}^{(L)}\|. \end{aligned}$$

Since σ is B_σ-Lipschitz and $\|\hat{\boldsymbol{\theta}}^{(l)}\|_2 \le B_\theta$,

$$\begin{aligned} \|\hat{\boldsymbol{x}}^{(L)}\| &= \|\sigma(\hat{\boldsymbol{y}}^{(L-1)})\| \\ &\le B_\sigma\|\hat{\boldsymbol{y}}^{(L-1)}\| \\ &= B_\sigma\|\hat{\boldsymbol{\theta}}^{(L-1)}\hat{\boldsymbol{x}}^{(L-1)}\| \\ &\le B_\sigma B_\theta\|\|\hat{\boldsymbol{x}}^{(L-1)}\| \\ &\le \cdots \\ &\le B_\sigma^{L-1}B_\theta^L\|\boldsymbol{x}\| \\ &= B_\sigma^{L-1}B_\theta^L B_x, \end{aligned}$$

whereas for the difference of post-activations,

$$\begin{aligned} \|\tilde{\boldsymbol{x}}^{(L)} - \hat{\boldsymbol{x}}^{(L)}\| &= \|\sigma(\tilde{\boldsymbol{y}}^{(L-1)}) - \sigma(\hat{\boldsymbol{y}}^{(L-1)})\| \\ &\le B_\sigma\|\tilde{\boldsymbol{y}}^{(L-1)} - \hat{\boldsymbol{y}}^{(L-1)}\|. \end{aligned}$$

By repeatedly applying this nested bound across layers, we obtain

$$\begin{aligned} &\|f(\boldsymbol{x};\tilde{\boldsymbol{\theta}}) - f(\boldsymbol{x};\hat{\boldsymbol{\theta}})\| \\ =& B_\sigma^{L-1}B_\theta^L B_x\|\tilde{\boldsymbol{\theta}}^{(L)} - \hat{\boldsymbol{\theta}}^{(L)}\| + B_\sigma B_\theta\|\tilde{\boldsymbol{y}}^{(L-1)} - \hat{\boldsymbol{y}}^{(L-1)}\| \\ =& B_\sigma^{L-1}B_\theta^L B_x\textstyle\sum_{l=1}^{2}\|\tilde{\boldsymbol{\theta}}^{(l)} - \hat{\boldsymbol{\theta}}^{(l)}\| + B_\sigma^2 B_\theta^2\|\tilde{\boldsymbol{y}}^{(L-2)} - \hat{\boldsymbol{y}}^{(L-2)}\| \\ =& \cdots \\ =& B_\sigma^{L-1}B_\theta^L B_x\textstyle\sum_{l=1}^{L}\|\tilde{\boldsymbol{\theta}}^{(l)} - \hat{\boldsymbol{\theta}}^{(l)}\| + B_\sigma^L B_\theta^L\|\boldsymbol{x}\| \\ \le& B_\sigma^{L-1}B_\theta^L B_x\textstyle\sum_{l=1}^{L}\|\tilde{\boldsymbol{\theta}}^{(l)} - \hat{\boldsymbol{\theta}}^{(l)}\| + B_\sigma^L B_\theta^L B_x. \end{aligned}$$

Hence, demanding $\|f(\boldsymbol{x};\tilde{\boldsymbol{\theta}}) - f(\boldsymbol{x};\hat{\boldsymbol{\theta}})\| \le \epsilon$ gives

$$\|\tilde{\boldsymbol{\theta}} - \hat{\boldsymbol{\theta}}\| \le \textstyle\sum_{l=1}^{L}\|\tilde{\boldsymbol{\theta}}^{(l)} - \hat{\boldsymbol{\theta}}^{(l)}\| \le \epsilon/(B_\sigma^{L-1}B_\theta^L B_x) - B_\sigma B_\theta.$$

For sub-Gaussian weights, Vershynin's matrix concentration (*Theorem 4.4.5.* in [22]) implies that, for any $t > 0$,

$$B_\theta = \|\boldsymbol{\theta}^{(l)}\| \le CK^{(l)}(\sqrt{m^{(l)}} + \sqrt{n^{(l)}} + t),$$

with probability at least $1 - 2\exp(-t^2)$. Substituting this into the above bound yields the inequality (8), and the probability statement in Formula (9) follows from the presence of B_θ^L in that expression. □

The bound in Theorem 2 is intentionally conservative. In practice, activation functions generally satisfy $B_\sigma \le 1$, inputs are normalized so that $B_x \le 1$, and model parameters often have sufficiently small variance to yield $B_\theta \le 1$. As a result, the factor $B_\sigma^{L-1}B_\theta^L B_x$ in (8) decays exponentially with the network depth L, ensuring that the upper bound on $|\tilde{\boldsymbol{\theta}} - \hat{\boldsymbol{\theta}}|$ is satisfied in typical deployments.

The next theorem relaxes the requirement to one involving only the standard deviation of the parameter difference, showing that an even weaker condition still secures full performance.

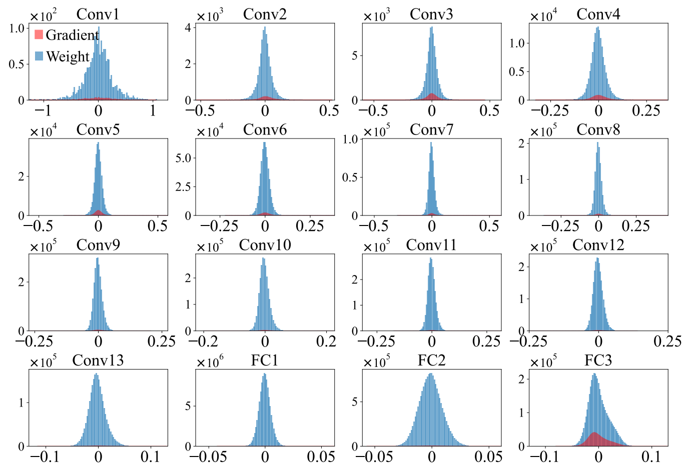

Fig. 4: Layer-wise histograms of VGG-16 weights and their accumulated gradient updates during adaptation, where across all layers the vast majority of values sit at or near zero.

Theorem 3 (Performance under bounded standard deviation)**.** *Given model parameters $\hat{\boldsymbol{\theta}}$, $\tilde{\boldsymbol{\theta}} \in \mathbb{R}^d$, where $\hat{\boldsymbol{\theta}}$ is fully updated and $\tilde{\boldsymbol{\theta}}$ is key-only (partially) updated, if*

$$\mathrm{Std}(\tilde{\boldsymbol{\theta}} - \hat{\boldsymbol{\theta}}) \le \epsilon/(B_\sigma^{L-1}B_\theta^L B_x) - B_\sigma B_\theta, \tag{10}$$

then $\tilde{\boldsymbol{\theta}} \in \mathcal{N}(\hat{\boldsymbol{\theta}})$ with probability $(1 - 2\exp(-t^2))^{L+1}$.

Proof. Theorem 4.4.5 in [22] gives, for any $t > 0$,

$$\|\tilde{\boldsymbol{\theta}} - \hat{\boldsymbol{\theta}}\| \le CK_\theta(\sqrt{d} + t),$$

with a probability of at least $1 - 2\exp(-t^2)$, where $K_\theta = \max_i\|\tilde{\theta}_i - \hat{\theta}_i\|_{\varphi_2}$. Combining this with the bound (8) from Theorem 2 yields

$$\begin{aligned} \|\tilde{\boldsymbol{\theta}} - \hat{\boldsymbol{\theta}}\| &\le CK_\theta(\sqrt{d} + t) \\ &\le \epsilon/(B_\sigma^{L-1}B_\theta^L B_x) - B_\sigma B_\theta, \end{aligned}$$

Then, we get the upper bound of the standard deviation of the parameter difference as

$$\begin{aligned} \mathrm{Std}(\tilde{\boldsymbol{\theta}} - \hat{\boldsymbol{\theta}}) &\le \max\|\tilde{\theta}_i - \hat{\theta}_i\|_{\varphi_2} \\ &= K_\theta \\ &\le [\epsilon/(B_\sigma^{L-1}B_\theta^L B_x) - B_\sigma B_\theta]/C(\sqrt{d} + t). \end{aligned}$$

□

Remark: ***The bounded adaptability requirements***

Theorems 2 and 3 together ensure that, as long as the weight shift respects the bound in Formula (8), the key-only update lies within the practical solution set defined in Section IV-B, i.e., it delivers performance fully comparable to full fine-tuning.

Both theorems above rely on the assumption that parameters outside the *key* remain nearly unchanged during model updates. Empirical evidence across diverse architectures and tasks confirms this behavior: fine-tuning generally causes only minor shifts in most weights (see Figure 4). As a result, the difference between these parameters and their updated counterparts remains small, ensuring that the distance and deviation bounds in Theorems 2 and 3 are typically satisfied in practice.

To ground this observation and justify the above theorems, the next two theorems prove that neurons (or filters) with smaller incoming weight norms receive proportionally smaller gradients. As a result, the weights not selected by the *key* naturally stay close to their starting values throughout fine-tuning, providing theoretical support for the stability assumption underlying our results.

2) Connecting Gradient Behavior to Parameter Stability: The two theorems below establish that neurons (or convolutional filters) whose incoming weight vectors have the smallest ℓ_1-norm receive the smallest back-propagated gradients and therefore undergo the least drift during the entire update process.

Theorem 4 (Parameter stability in FCN)**.** *Consider a fully connected neural network with ReLU activation, loss function $\mathcal{L}$, and fully updated parameters $\hat{\boldsymbol{\theta}}$. For neurons j and k in layer $(l-1)$, if*

$$\|\hat{\boldsymbol{\theta}}_j^{(l-1)}\|_1 \leq \|\hat{\boldsymbol{\theta}}_k^{(l-1)}\|_1,$$

then for any neuron i in layer l,

$$\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{\theta}}_i^{(l)}} \leq \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{\theta}}_j^{(l)}}.$$

Thus, neurons with smaller incoming ℓ_1-norm weights receive smaller gradients.

Proof. Consider the gradient of parameter $\theta_{i,j}^{(l)}$:[4]

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \hat{\theta}_{i,j}^{(l)}} &= \frac{\partial \mathcal{L}}{\partial \sigma(\boldsymbol{y}^{(l+1)})} \frac{\partial \sigma(\boldsymbol{y}^{(l+1)})}{\partial y_i^{(l)}} \frac{\partial y_i^{(l)}}{\partial \hat{\theta}_{i,j}^{(l)}} \\ &= \frac{\partial \mathcal{L}}{\partial \sigma(\boldsymbol{y}^{(l+1)})} \hat{\boldsymbol{\theta}}_{:,i}^{(l+1)} \sigma'(y_i^{(l)}) y_j^{(l-1)}\end{aligned}$$

Similarly, we have the gradient of $\theta_{i,k}^{(l)}$

$$\frac{\partial \mathcal{L}}{\partial \hat{\theta}_{i,k}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \sigma(\boldsymbol{y}^{(l+1)})} \hat{\boldsymbol{\theta}}_{:,i}^{(l+1)} \sigma'(y_i^{(l)}) y_k^{(l-1)}$$

If $\|\hat{\boldsymbol{\theta}}_j^{(l-1)}\|_1 \leq \|\hat{\boldsymbol{\theta}}_k^{(l-1)}\|_1$, then clearly we have $\|\hat{\boldsymbol{\theta}}_j^{(l-1)} \boldsymbol{y}^{(l-2)}\|_1 \leq \|\hat{\boldsymbol{\theta}}_k^{(l-1)} \boldsymbol{y}^{(l-2)}\|_1$. Since $y_j^{(l-1)} = \sigma(\hat{\boldsymbol{\theta}}_j^{(l-1)} \boldsymbol{y}^{(l-2)})$ and $y_k^{(l-1)} = \sigma(\hat{\boldsymbol{\theta}}_k^{(l-1)} \boldsymbol{y}^{(l-2)})$, and $\sigma(\cdot)$ is the ReLU activation, we have $y_j^{(l-1)} \leq y_k^{(l-1)}$. Thus, it follows that $\frac{\partial \mathcal{L}}{\partial \hat{\theta}_{i,j}^{(l)}} \leq \frac{\partial \mathcal{L}}{\partial \hat{\theta}_{i,k}^{(l)}}$. ☐

While the theorem specifically addresses fully connected layers, the following theorem extends the discussion to convolutional layers, emphasizing the unique symmetry and structure inherent to these layers.

Theorem 5 (Parameter stability in CNN)**.** *Consider a convolutional neural network $f(\boldsymbol{x}; \boldsymbol{\theta})$ with ReLU activations, a loss*

[4]Here, we omit the special cases of infeasible notation involving l.

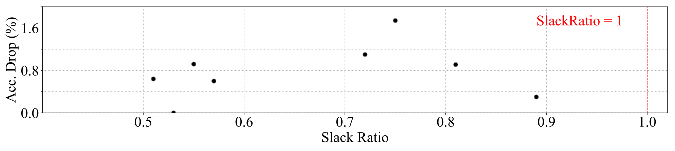

Fig. 5: Accuracy drop versus *slack ratio* for VGG-16 and VGG-19 on MNIST, FashionMNIST, CIFAR-10, and CIFAR-100, showing empirical slack below 1 while accuracy stays close to full fine-tuning.

function $\mathcal{L}$, and training data $\mathcal{D}$. Let $\hat{\boldsymbol{\theta}}$ denote fully fine-tuned parameters. Suppose two filters s and s' in layer $(l-1)$ satisfy

$$\|\hat{\boldsymbol{\theta}}_s^{(l-1)}\|_1 \leq \|\hat{\boldsymbol{\theta}}_{s'}^{(l-1)}\|_1.$$

Then, the gradients satisfy

$$\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{\theta}}_{:,s}^{(l)}} \leq \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{\theta}}_{:,s'}^{(l)}},$$

indicating that filters with lower ℓ_1-norms induce smaller gradients in subsequent layers.

Proof. Assume each filter parameter $\hat{\boldsymbol{\theta}}_{:,s}^{(l)}$ has dimension $\mathbb{R}^{c_{\text{out}} \times (c_{\text{in}} \times k \times k)}$, where c_{out} and c_{in} are the output and input channel counts, respectively, and k is the kernel size. The gradient of $\hat{\boldsymbol{\theta}}_{:,s}^{(l)}$ can be written as

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{\theta}}_{:,s}^{(l)}} &= \frac{\partial \mathcal{L}}{\partial \sigma(\boldsymbol{y}^{(l+1)})} \frac{\partial \sigma(\boldsymbol{y}^{(l+1)})}{\partial y_{c,i,j}^{(l)}} \frac{\partial y_{c,i,j}^{(l)}}{\partial \hat{\boldsymbol{\theta}}_{:,s}^{(l)}} \\ &= \frac{\partial \mathcal{L}}{\partial \sigma(\boldsymbol{y}^{(l+1)})} \hat{\boldsymbol{\theta}}^{(l+1)} \sigma'(y_{:,i,j}^{(l)}) y_{s,i:i+k,j:j+k}^{(l-1)}.\end{aligned}$$

Similarly, we have the gradient of $\hat{\boldsymbol{\theta}}_{:,s'}^{(l)}$

$$\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{\theta}}_{:,s'}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \sigma(\boldsymbol{y}^{(l+1)})} \hat{\boldsymbol{\theta}}^{(l+1)} \sigma'(y_{:,i,j}^{(l)}) y_{s',i:i+k,j:j+k}^{(l-1)}.$$

Like Theorem 4, $\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{\theta}}_{:,s}^{(l)}} \leq \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{\theta}}_{:,s'}^{(l)}}$ if $\|\hat{\boldsymbol{\theta}}_s^{(l-1)}\|_1 \leq \|\hat{\boldsymbol{\theta}}_{s'}^{(l-1)}\|_1$. ☐

We conclude that the gradient for each filter is proportional to its post-activation output, which is smaller for filters with lower incoming weight norms.

Remark: AdaLoc *offers strong adaptability*

Theorems 4 and 5 provide strong evidence that AdaLoc *will typically meet the parameter distance requirements, thereby supporting the performance guarantees in Theorems 2 and 3 throughout adaptation. Taken together, they offer a strong theoretical guarantee that* AdaLoc ***always*** *delivers performance fully comparable to full fine-tuning.*

C. Empirical Tightness of Theoretical Bounds

We assess the empirical tightness of Theorems 2 and 3 by computing the actual parameter distance $\|\tilde{\theta} - \hat{\theta}\|$ across various datasets. For the theoretical bound, we compute $\epsilon/(B_\sigma^{L-1} B_\theta^L B_x) - B_\sigma B_\theta$ using constants estimated from network statistics (*e.g.,* $\mathbb{E}[B_\theta]$) following the numerical setup in [12]. We report the resulting *slack ratio*

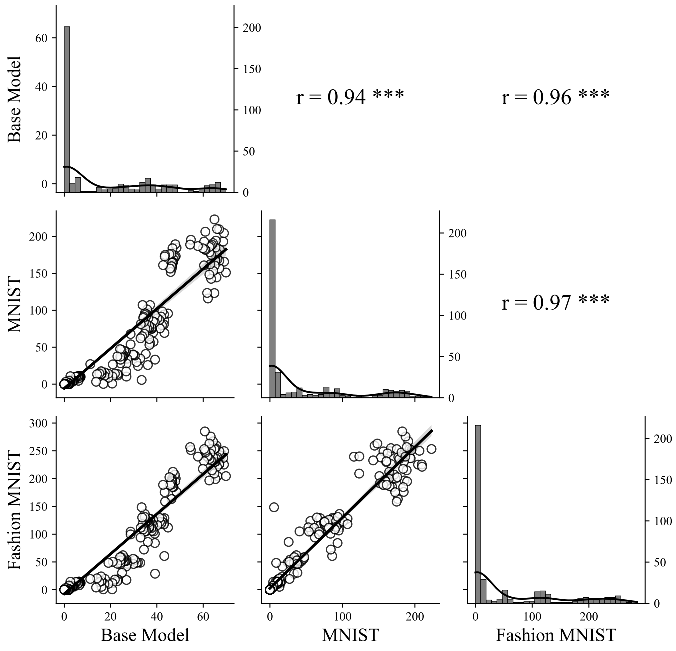

Fig. 6: Visualization of the correlation between the ℓ_1-norms of the base model (ImageNet-pretrained) and the fine-tuned models (on MNIST and Fashion-MNIST).

$\frac{\|\tilde{\boldsymbol{\theta}}-\hat{\boldsymbol{\theta}}\|_2}{\epsilon/(B_\sigma^{L-1}B_\theta^L B_x)-B_\sigma B_\theta}$ together with accuracy drop in Figure 5 for VGG-16 and VGG-19. For instance, on CIFAR-100 with VGG-16 the empirical distance is 0.51 while the theoretical threshold is 0.96, giving a slack ratio of 0.53. Across all tasks, the empirical distance remained within 50-90% of the theoretical threshold, suggesting that the bound is safe but not excessively pessimistic. Notably, models consistently retained performance within 2% of fully fine-tuned baselines, highlighting a consistent pattern of empirical slack. This slack arises because the theoretical bounds are derived under worst-case assumptions, such as compounding layer-wise Lipschitz constants and uniformly tight parameter sensitivity, which overestimate actual performance degradation. In practice, sparse activation patterns, overparameterization, and flat regions in the loss landscape enable networks to absorb moderate parameter deviations with negligible performance loss. Additionally, ADALOC's strategy of selectively updating high-impact weights acts as an implicit regularizer, often improving generalization on smaller target datasets. Together, these factors explain why ADALOC delivers robust adaptation performance even beyond the strict guarantees of our theoretical bounds.

VI. EXPERIMENTS

In this section we proceed in three steps. First, we verify that updating only the *key* matches the accuracy of full fine-tuning on new tasks, demonstrating efficient localization and effective domain adaptation (Section VI-A). Then, we show that removing those *key* in a static setting drops the model to a random guess level, confirming usage control. Finally, we compare ADALOC with baseline methods and highlight its ability to preserve the same control even after successive model updates (Section VI-B).

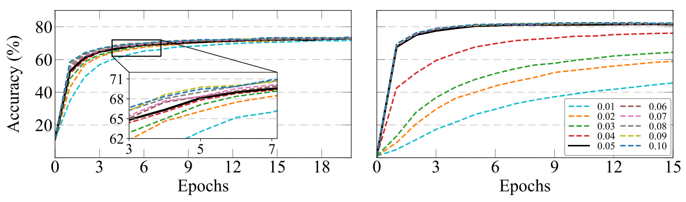

Fig. 7: Performance of ResNet-18 on CIFAR-10 (left) and DenseNet-121 on CIFAR-100 (right) when restricting updates to 1%-10% of neurons as the *key*.

A. Model Adaptation

We begin by empirically verifying the performance guarantees discussed in the previous Section. Specifically, we show that a compact *key*, selected by focusing on large-magnitude parameters, consistently preserves most of the performance gains of a fully fine-tuned model.

Experimental setup. We evaluate ADALOC on several commonly used datasets, spanning both image classification and language classification tasks. For image classification, we use MNIST [24], a dataset of 60,000 training and 10,000 testing grayscale images of size 28×28 across 10 classes; Fashion-MNIST [25], a similar dataset with images of clothing items; CIFAR-10 and CIFAR-100 [26], RGB datasets containing 50,000 training and 10,000 testing images of size 32×32, where CIFAR-10 contains 10 classes and CIFAR-100 has 100 classes; Caltech-256 [27], a dataset with 256 object categories for diverse classification tasks; and Flowers-102 [28], a fine-grained dataset of 102 flower species. For language classification, we use QNLI [29], a question-answer inference task from the GLUE benchmark; SST-2 [30], a sentiment classification task; and TweetEval [31], a dataset focusing on sentiment analysis in social media text. The models evaluated include DenseNet, ResNet, and ConvNeXt-V2 for image classification, and BERT, RoBERTa, and DeBERTa for language classification. All image classification models are initialized with ImageNet-pre-trained weights, while language models are initialized with their respective pre-trained checkpoints.

1) Key Selection: We analyze the weight distribution of a base model (ImageNet-pre-trained) before and after fine-tuning on MNIST and Fashion-MNIST datasets to explore the correlation between weight magnitudes and their gradient updates. Specifically, we examine how accumulated gradient updates during fine-tuning relate to the initial weight magnitudes. The correlation graphs in Figure 6 show scatter plots, with neurons sorted by the base model neurons' ℓ_1-norm, revealing a strong proportional relationship between the magnitude of initial neuron weights and their corresponding updates during fine-tuning. This observation supports the theoretical analysis in Section V-B, highlighting that large-magnitude neuron weights dominate updates during adaptation. Furthermore, the

TABLE II: Effectiveness of AdaLoc's *key* selection versus three baselines: full fine-tuning, random selection, and smallest-magnitude selection. With the *key* present (authorized use), AdaLoc matches or exceeds full fine-tuning accuracy; when the *key* is withheld (entries in green), accuracy drops to random-guess levels.

	CIFAR-100			Caltech-256			Flowers-102		
	DenseNet-121	ResNet-152	ConvNeXt-V2	DenseNet-121	ResNet-152	ConvNeXt-V2	DenseNet-121	ResNet-152	ConvNeXt-V2
Baseline	**81.97%**	87.35%	88.23%	84.79%	**85.19%**	92.03%	87.64%	87.84%	97.75%
Random	51.90%	28.57%	76.47%	38.30%	61.10%	81.82%	20.20%	8.33%	54.12%
Bottom	28.42%	10.31%	44.37%	16.45%	23.68%	17.67%	4.41%	1.57%	8.24%
AdaLoc	81.14%(1.02%)	**88.88%**(1.00%)	**90.17%**(1.07%)	**86.52%**(0.67%)	84.39%(0.47%)	**93.37%**(0.52%)	**88.14%**(1.42%)	**89.41%**(1.19%)	**98.92%**(1.08%)

TABLE III: Comparison of AdaLoc's *key-only* update and full fine-tuning on language classification tasks.

	QNLI			SST2			TweetEval		
	RoBERTa-base	BERT-base	DeBERTa-base	RoBERTa-base	BERT-base	DeBERTa-base	RoBERTa-base	BERT-base	DeBERTa-base
Baseline	60.81%	61.38%	60.61%	91.74%	92.66%	94.61%	73.50%	72.80%	74.50%
AdaLoc	62.21%	62.68%	61.65%	90.71%	90.56%	92.89%	67.45%	71.20%	66.20%

histograms (with the vertical axis on the right representing the number of neurons) demonstrate that such significant neuron weights are relatively sparse, with most parameters being near zero. These findings validate the heuristic of prioritizing large-magnitude weights for effective and efficient adaptation.

Next, we assess the fraction of weights required to achieve performance comparable to full fine-tuning. To maintain a compact *key*, we tested unlocking 1% to 10% of the neurons with the highest magnitudes (ℓ_1-norm). Our experiments show that 5% consistently achieves strong performance across tasks, while 1-4% suffices for simpler datasets like CIFAR-10 but underperforms on more complex tasks. Increasing beyond 5% provides only marginal improvements. Balancing performance with key size, we adopt 5% as the standard, as it performs reliably across all settings.

Figure 7 illustrates two representative scenarios. For ResNet-18 on CIFAR-10 (left), high performance is observed across all tested fractions, with 5% of neurons offering an optimal balance between accuracy and convergence speed. For DenseNet-121 on CIFAR-100 (right), an extreme case, performance drops significantly when using only 1-4% of neurons, underscoring the robustness of selecting 5% for domain adaptation. These results reaffirm the effectiveness of focusing updates on a small, high-impact subset of weights, ensuring efficient and reliable adaptation in diverse tasks.

2) Effective Model Adaptation: We assess adaptation under a strict 5% parameter budget, where only 5% of weights are used as the *key*. Specifically, we compare full fine-tuning (baseline) and two alternative 5% strategies that select neurons randomly or by small magnitude (bottom) against AdaLoc, which prioritizes large-magnitude weights. These comparisons show that focusing updates on large-magnitude weights is markedly more effective under the same budget.

Results. Table II presents the results of image classification across various models and datasets. The proposed approach, AdaLoc, achieves performance comparable to full fine-tuning, attaining an accuracy of up to 98.92%. This result reaffirms that AdaLoc effectively identifies the critical neurons that drive model adaptation. Compared to approaches that select neurons randomly or based on small-magnitude weights, AdaLoc delivers substantial accuracy improvements, with gains of up to 90.68%. As shown in Figure 8, AdaLoc also maintains convergence rates on par with or exceeding those of full fine-tuning, confirming its efficiency and effectiveness across diverse tasks.

For language classification, a similar trend is observed as shown in Table III, with AdaLoc achieving an accuracy of up to 92.89%, which is on par with full fine-tuning. This further validates AdaLoc's effectiveness in identifying key neurons essential for model adaptation across different architectures and tasks.

In some cases, AdaLoc even outperforms full fine-tuning. This could happen for two reasons. First, both approaches rely on finding local optima, and by focusing on a smaller subset of critical parameters, AdaLoc may occasionally land on a better local solution. Second, for smaller fine-tuning datasets, updating fewer parameters can act as a form of regularization, helping reduce overfitting and improve generalization. This behavior underscores the advantage of selectively fine-tuning high-impact parameters.

Takeaway

By tuning only the key-designated weights, chosen for their strong capacity to absorb updates, AdaLoc reaches full fine-tuning performance for authorized users while leaving the remainder of the network untouched, as guaranteed by Theorems 2-5.

B. Adaptable Usage Control

To evaluate the adaptability and effectiveness of AdaLoc, we compare it with three active model usage control baselines: AdvParams [8], NNSplitter [2], and CoreLocker [9]. Each method follows the same workflow in which a pretrained

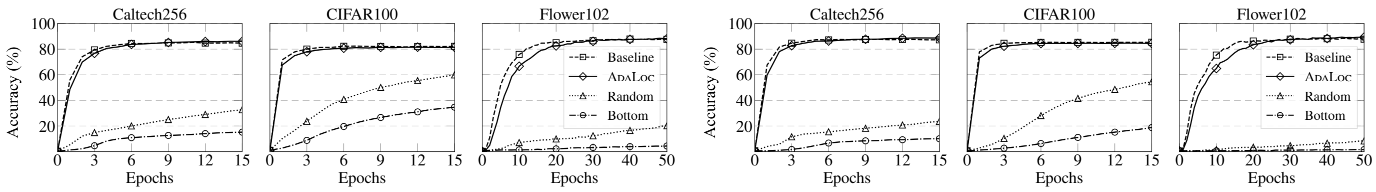

Fig. 8: Training convergence for DenseNet-121 (left) and ResNet-152 (right) with ADALOC *key-only* updates, full fine-tuning, and two baselines (random and smallest-magnitude) across datasets.

TABLE IV: ADALOC with relaxed neuron selection on image classification: selecting the top 5% neurons generally yields the best accuracy, while sampling 5% from the top 10% or 15% remains nearly as effective.

	CIFAR-100			Caltech-256			Flowers-102		
	DenseNet-121	ResNet-152	ConvNeXt-V2	DenseNet-121	ResNet-152	ConvNeXt-V2	DenseNet-121	ResNet-152	ConvNeXt-V2
ADALOC $_{\text{top 5\%}}$	81.14%	88.88%	90.17%	86.52%	84.39%	93.37%	88.14%	89.41%	98.92%
ADALOC $_{\text{5\% from top 10\%}}$	79.26%	80.94%	90.06%	81.06%	82.08%	92.75%	84.41%	84.61%	98.43%
ADALOC $_{\text{5\% from top 15\%}}$	77.71%	78.31%	89.89%	79.57%	78.03%	93.25%	85.29%	80.29%	98.24%

TABLE V: Relaxed neuron selection on language classification shows that selecting the top 5% neurons or sampling 5% from the top 10% or 15% all yield similarly effective performance.

	QNLI			SST2			Tweet		
	RoBERTa-base	BERT-base	DeBERTa-base	RoBERTa-base	BERT-base	DeBERTa-base	RoBERTa-base	BERT-base	DeBERTa-base
ADALOC $_{\text{top 5\%}}$	62.21%	62.68%	61.65%	90.71%	90.56%	92.89%	67.45%	71.20%	66.20%
ADALOC $_{\text{5\% from top 10\%}}$	62.29%	62.29%	60.72%	90.94%	91.05%	91.97%	69.15%	71.60%	57.55%
ADALOC $_{\text{5\% from top 15\%}}$	62.84%	62.80%	61.50%	91.28%	91.97%	92.66%	72.30%	73.55%	66.95%

DNN is converted into a *locked model* and an authorization key that record selected weight indices and their original values. The locked model, stored in normal memory, provides only limited functionality, while the secrets are kept in a protected environment that attackers cannot reach. Once the locked model is combined with its secrets inside this secure enclave, full accuracy is restored. AdvParams injects carefully crafted adversarial perturbations into chosen weights so that, without the corrective key, the network's outputs become unusable; NNSplitter uses reinforcement learning to partition the network into a public portion and a small secret block that must be recombined for normal inference; and CoreLocker disables influential channels and relies on reinstating them to recover full performance.

1) Key Pool: To demonstrate the flexibility of ADALOC's *key* selection, we introduce an *key* pool, which relaxes the strict requirement of selecting the top 5% of neurons by ℓ_1 magnitude. Instead, neurons are sampled from broader subsets, specifically the top 10% and 15% of weights, to accommodate scenarios where precise ranking might be constrained by hardware or other mechanisms.

Results. Table IV and V present the results of relaxed sampling. Across all models and datasets, selecting the top 5% of neurons consistently yields the highest accuracy, reaching up to 98.92%. Although sampling from a broader set, such as selecting 5% from the top 10% or 15%, results in a slight performance degradation, it still achieves an accuracy of up to 98.43%, closely matching the performance of the top 5%.

Takeaway

These findings validate ADALOC's efficient localization property: even with a relaxed access-key criterion, adaptation remains nearly as effective, demonstrating robustness and practical flexibility.

2) Static Usage Control: We first verify that ADALOC enforces usage control in a static setting, *i.e.,* without any further model updates. For all models in Table II, we strip the *key* after adaptation and record the top-1 accuracy. Across every datasets the locked models collapse to random-guess performance. On CIFAR-100 (100 classes) all backbone drops from 81.97-88.23% down to near 1.0%; on Caltech-256 (256 classes) accuracy drops below 0.7%. These values are highlighted in green in Table II.

To demonstrate practicality, we also test widely used architectures beyond our main suite. A Vision Transformer [11] trained on CIFAR-100 falls from 81.4% to 1.2% when the *key* is removed, and a BERT on AG News [32] (4 classes) drops from 87.2% to 25.3%, effectively the random level. Because these drops occur regardless of architecture or dataset size, we reaffirm that ADALOC is model-agnostic: removing the key reliably degrades inference to random guessing, thereby maintaining strict static usage control.

3) Usage Control during Model Adaptation: We next assess whether this usage control property holds during model adap-

TABLE VI: Authorized vs. unauthorized performance of ResNet-152 adapted from MNIST, Fashion-MNIST, CIFAR-10, and CIFAR-100. AdaLoc consistently preserves usage control, delivering full performance for authorized users while forcing unauthorized usage to completely unusable levels.

.6Method	MNIST		Fashion-MNIST		CIFAR-10		CIFAR-100		.6Usage Control
	Authorized	Unauthorized	Authorized	Unauthorized	Authorized	Unauthorized	Authorized	Unauthorized	
AdvParams [8]	98.29%	10.62%	93.07%	87.19%	94.27%	91.80%	87.35%	87.01%	✗
CoreLocker [9]	98.29%	10.00%	93.07%	81.71%	94.27%	80.07%	87.35%	76.26%	✗
NNSplitter [2]	98.29%	10.00%	93.07%	90.29%	94.27%	93.40%	87.35%	86.20%	✗
AdaLoc	98.29%	10.00%	92.60%	10.01%	91.26%	10.00%	88.88%	1.02%	✓

tation to new tasks. To simulate dynamic adaptation scenarios, we begin with MNIST as the base dataset where the *key* is generated, and fine-tune the locked model on target datasets including Fashion-MNIST, CIFAR-10, and CIFAR-100. This setup allows us to evaluate whether AdaLoc can maintain effective usage control while enabling legitimate updates and preventing unauthorized exploitation.

Results. Table VI compares AdaLoc with three representative usage control baselines across Fashion-MNIST, CIFAR-10, and CIFAR-100. After fine-tuning, the baselines still let adversaries recover up to 93.40% accuracy, leaving protection almost ineffective. Under the same conditions, AdaLoc holds unauthorized accuracy near random guess level (10.02% on Fashion-MNIST, 10.00% on CIFAR-10, 1.01% on CIFAR-100), while authorized users retain high performance. The small accuracy drop for legitimate users is a worthwhile trade-off for the much stronger usage control delivered by AdaLoc.

Takeaway

AdaLoc *enables continual model updates across tasks and datasets while still blocking unauthorized use, making it a scalable choice for secure DNN deployment in evolving AI systems.*

4) Discussion of Key Management: Effective key management is central to the success of AdaLoc, as the framework relies on securely handling the *key*. The *key* ensures the core functionality of the model remains locked and enables updates to accommodate new tasks or domains. Managing it poses several challenges, particularly in scenarios involving large-scale deployments or dynamic updates. First, securely storing and transmitting keys is critical to prevent unauthorized access. AdaLoc addresses this by integrating secure hardware mechanisms, such as trusted execution environments (TEEs), which isolate key storage from external access, reducing the risk of leaks. Second, distributing keys to authorized users requires robust authentication protocols. Techniques such as public-key cryptography and multi-factor authentication can enhance this process, ensuring that only verified users can retrieve and use the keys.

Additionally, key scalability becomes a concern when adapting models across multiple environments or frequent updates. To address this, the *key* is designed to be compact, targeting only high-impact parameters, which minimizes storage and distribution overhead. The framework could further benefit from hierarchical keying systems, where a master key governs subsets of the *key*, reducing the complexity of key distribution. Finally, dynamic re-keying mechanisms ensure that old keys become obsolete after updates, preventing exploitation of previously exposed keys. By combining these strategies, AdaLoc offers a secure and scalable approach to key management, maintaining robust usage control while enabling seamless model evolution in real-world applications.

Takeaway

With these key-management measures in place, every model revision stays locked to outsiders, while an authorized controller can always recover full, up-to-date accuracy by presenting the current access key.

VII. Related Work

Model IP protection. Model IP protection aims to shield a network's intellectual property from unauthorized replication and monetization. Passive safeguards such as watermarking [33]–[36] support post-hoc ownership verification, but offer limited control over how a disclosed model is used and cannot, by themselves, prevent misuse once leakage occurs. To address this, more proactive approaches encrypt or obfuscate model parameters, forcing attackers to contend with distorted weight representations. For instance, Chakraborty *et al.* [17], [37] rely on specialized hardware and a key-dependent back-propagation process to obscure weights, rendering unauthorized deployments on untrusted hardware functionally ineffective. Although effective, this strategy depends on hardware modifications and is not trivially applicable to pre-trained networks. Fan *et al.* [38] introduce a "passport layer" to preserve the network's accuracy only when the correct passport is provided, curbing ambiguity attacks. Other methods insert a secret key into training data [6], [7], making inference accurate solely with key-preprocessed inputs, though retraining per key can be laborious. Recent works add adversarial perturbations into the model [2], [8], yet still require cautious handling of secure memory and carefully configured perturbations to maintain robustness. In contrast, our work introduces a new focus by enabling *adaptable, secure model updates* that address the dynamic needs of modern applications while preserving essential protections.

Trusted execution environment. Trusted execution environments (TEEs), such as ARM TrustZone on mobile devices [39], offer a promising solution for active model protection. TEEs provide hardware-based isolation, partitioning memory into a secure (trusted) world and a normal (untrusted) world. Communication between these environments occurs through secure monitor calls [40], ensuring that only authorized users can access the trusted world while blocking attackers. Prior research has demonstrated the efficacy of TEEs in protecting models [41], [42], and we adopt this implementation approach as described in [42]. However, a detailed exploration of TEE vulnerabilities, such as side-channel attacks, lies beyond the scope of this work.

A key challenge of TEE-based approaches is the limited secure memory available for trusted applications, typically around 10 MB [42]. Meanwhile, modern DNNs are increasingly large, with state-of-the-art models like ResNet-101 containing over 155 million parameters [43]. Our method bridges this gap by confining updates to a tiny *access key* subset of weights, keeping the enclave's memory footprint manageable while still delivering strong usage control.

Partial fine-tuning. Partial fine-tuning has gained traction as a way to adapt large models by updating only a subset of parameters [44], [45], reducing the computation and preserving most of the original network. Approaches such as adapters [46], LoRA layers [47], or bias-only tuning [48] have proven effective, particularly when models must frequently handle *practical dynamic scenarios* (*e.g.,* domain shifts or personalized tasks). These methods focus on optimizing resource usage and retaining initial performance, yet they typically lack formal guarantees and do not address the need for locking critical parameters to prevent malicious reconfiguration or extraction. By contrast, our method deliberately disables the model for unauthorized users until the designated weight subset (*i.e.,* the *key*) is restored. This reframes partial tuning not as an efficiency trick but as a usage-control mechanism that enables *adaptability*. We formalize this perspective and prove that the locking step, together with subsequent key-only updates, delivers both strong protection and reliable adaptation.

VIII. Conclusion and Future Work

We propose a novel usage control paradigm in which a model remains securely locked yet fully adaptable by confining every update to a compact *access key*; possession of this key restores the model's full adapted capabilities, while its absence reduces performance to a completely unusable level. We for the first time formalize a comprehensive notion of model usage control that integrates *accessibility* and *adaptability*, and derive rigorous theoretical bounds that underpin AdaLoc's effectiveness. Validation across diverse tasks and architectures confirms that this approach enables seamless adaptation for legitimate stakeholders while enforcing robust protection against unauthorized use. AdaLoc thus delivers a lightweight, practical solution for safeguarding model intellectual property in real-world machine learning workflows.

A possible avenue for future exploration lies in addressing scenarios where the *key* is partially exposed, such as through collusion attacks where unauthorized entities combine fragments of the *key* to circumvent restrictions. Developing robust mechanisms to tolerate such leakage while maintaining the model's performance and security would significantly enhance the resilience of AdaLoc. These advancements could involve designing layered security protocols, redundancy strategies, or adaptive locking mechanisms to mitigate risks associated with partial exposure. By tackling these challenges, AdaLoc could offer strengthened protection in dynamic, adversarial, and security-sensitive environments, ensuring its practical viability across diverse real-world applications.

Acknowledgement

We thank Chi Wang for his insightful discussions and generous experimental support, and Zhiyong Ma for his valuable assistance with the experiments.

References

[1] J. Yagnik, "Private ai compute: our next step in building private and helpful ai," https://blog.google/technology/ai/google-private-ai-compute/, Nov. 2025, google Blog.
[2] T. Zhou, Y. Luo, S. Ren, and X. Xu, "Nnsplitter: An active defense solution for dnn model via automated weight obfuscation," in *Proceedings of the 40th International Conference on Machine Learning (ICML)*. JMLR, 2023.
[3] T. Nayan, Q. Guo, M. A. Duniawi, M. Botacin, S. Uluagac, and R. Sun, "SoK: All you need to know about On-Device ML model extraction - the gap between research and practice," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 5233–5250. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/nayan
[4] C. Yan, R. Ren, M. H. Meng, L. Wan, T. Y. Ooi, and G. Bai, "Exploring chatgpt app ecosystem: Distribution, deployment and security," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 1370–1382.
[5] Z. Wang, Z. Ma, Z. Ma, S. Liu, A. Liu, D. Wang, M. Xue, and G. Bai, "Catch-only-one: Non-transferable examples for model-specific authorization," 2025. [Online]. Available: https://arxiv.org/abs/2510.10982
[6] A. Pyone, M. Maung, and H. Kiya, "Training dnn model with secret key for model protection," in *2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)*, 2020, pp. 818–821.
[7] M. Chen and M. Wu, "Protect your deep neural networks from piracy," in *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, 2018, pp. 1–7.
[8] M. Xue, Z. Wu, Y. Zhang, J. Wang, and W. Liu, "AdvParams: An active DNN intellectual property protection technique via adversarial perturbation based parameter encryption," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 3, pp. 664–678, 2023.
[9] Z. Wang, Z. Ma, X. Feng, R. Sun, H. Wang, M. Xue, and G. Bai, "Corelocker: Neuron-level usage control," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 2497–2514.
[10] M. Malec, "Generative ai statistics: Insights and emerging trends for 2025," December 2024. [Online]. Available: https://hatchworks.com/blog/gen-ai/generative-ai-statistics/
[11] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *9th International Conference on Learning Representations (ICLR)*, 2021.
[12] X. Qian and D. Klabjan, "A probabilistic approach to neural network pruning," in *International Conference on Machine Learning (ICML)*. PMLR, 2021, pp. 8640–8649.
[13] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" *Proceedings of machine learning and systems*, vol. 2, pp. 129–146, 2020.
[14] N. Lin, X. Chen, H. Lu, and X. Li, "Chaotic weights: A novel approach to protect intellectual property of deep neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 7, pp. 1327–1339, 2021.

[15] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in *International Conference on Learning Representations (ICLR)*, 2015.

[16] X. Yuan, P. He, Q. Zhu, and X. Li, “Adversarial examples: Attacks and defenses for deep learning,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[17] A. Chakraborty, A. Mondai, and A. Srivastava, “Hardware-assisted intellectual property protection of deep learning models,” in *57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[18] Z. Sun, R. Sun, C. Liu, A. R. Chowdhury, L. Lu, and S. Jha, “Shadownet: A secure and efficient on-device model inference system for convolutional neural networks,” in *2023 IEEE Symposium on Security and Privacy (IEEE S&P)*, 2023, pp. 1596–1612.

[19] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” *arXiv preprint arXiv:1608.08710*, 2016.

[20] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning convolutional neural networks for resource efficient inference,” in *International Conference on Learning Representations (ICLR)*, 2017.

[21] H. Lim, S.-D. Roh, S. Park, and K.-S. Chung, “Robustness-aware filter pruning for robust neural networks against adversarial attacks,” in *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*, 2021, pp. 1–6.

[22] R. Vershynin, *High-dimensional probability: An introduction with applications in data science*. Cambridge University Press, 2018, vol. 47.

[23] E. Malach, G. Yehudai, S. Shalev-Schwartz, and O. Shamir, “Proving the lottery ticket hypothesis: Pruning is all you need,” in *International Conference on Machine Learning*. PMLR, 2020, pp. 6682–6691.

[24] L. Deng, “The mnist database of handwritten digit images for machine learning research,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[25] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms,” *arXiv preprint arXiv:1708.07747*, 2017.

[26] A. Krizhevsky, G. Hinton *et al.*, “Learning multiple layers of features from tiny images,” 2009.

[27] G. Griffin, A. Holub, P. Perona *et al.*, “Caltech-256 object category dataset,” Technical Report 7694, California Institute of Technology Pasadena, Tech. Rep., 2007.

[28] M.-E. Nilsback and A. Zisserman, “Automated flower classification over a large number of classes,” in *2008 Sixth Indian conference on computer vision, graphics & image processing*. IEEE, 2008, pp. 722–729.

[29] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, “Glue: A multi-task benchmark and analysis platform for natural language understanding,” in *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[30] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, “The stanford sentiment treebank,” *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1631–1642, 2013.

[31] F. Barbieri, J. Camacho-Collados, L. Espinosa Anke, and L. Neves, “TweetEval: Unified benchmark and comparative evaluation for tweet classification,” in *Findings of the Association for Computational Linguistics: EMNLP 2020*, T. Cohn, Y. He, and Y. Liu, Eds. Online: Association for Computational Linguistics, Nov. 2020, pp. 1644–1650. [Online]. Available: https://aclanthology.org/2020.findings-emnlp.148/

[32] X. Zhang, J. Zhao, and Y. LeCun, “Character-level convolutional networks for text classification,” *Advances in neural information processing systems*, vol. 28, 2015.

[33] J. Guo and M. Potkonjak, “Watermarking deep neural networks for embedded systems,” in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

[34] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, “Embedding watermarks into deep neural networks,” in *Proceedings of ACM on International Conference on Multimedia Retrieval (ICMR)*, 2017, p. 269277.

[35] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, “Watermarking techniques for intellectual property protection,” in *Proceedings of the 35th Annual Design Automation Conference (DAC)*, 1998, pp. 776–781.

[36] Z. Wang, O. Byrnes, H. Wang, R. Sun, C. Ma, H. Chen, Q. Wu, and M. Xue, “Data hiding with deep learning: A survey unifying digital watermarking and steganography,” *IEEE Transactions on Computational Social Systems*, 2023.

[37] Z. Wang, Z. Ma, X. Feng, Z. Mei, E. Ma, D. Wang, M. Xue, and G. Bai, “Ai model modulation with logits redistribution,” in *Proceedings of the ACM on Web Conference 2025*, ser. WWW ’25. New York, NY, USA: Association for Computing Machinery, 2025, p. 46994709. [Online]. Available: https://doi.org/10.1145/3696410.3714737

[38] L. Fan, K. W. Ng, and C. S. Chan, “Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[39] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin, “Trustzone explained: Architectural features and use cases,” in *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*. IEEE, 2016, pp. 445–451.

[40] M. Ye, J. Sherman, W. Srisa-An, and S. Wei, “Tzslicer: Security-aware dynamic program slicing for hardware isolation,” in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2018, pp. 17–24.

[41] H. Chen, C. Fu, B. D. Rouhani, J. Zhao, and F. Koushanfar, “Deepattest: an end-to-end attestation framework for deep neural networks,” in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2019, pp. 487–498.

[42] Z. Sun, R. Sun, C. Liu, A. Chowdhury, L. Lu, and S. Jha, “Shadownet: A secure and efficient on-device model inference system for convolutional neural networks,” in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 1489–1505.

[43] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[44] Z. Shen, Z. Liu, J. Qin, M. Savvides, and K.-T. Cheng, “Partial is better than all: Revisiting fine-tuning strategy for few-shot learning,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 11, 2021, pp. 9594–9602.

[45] P. Ye, Y. Huang, C. Tu, M. Li, T. Chen, T. He, and W. Ouyang, “Partial fine-tuning: A successor to full fine-tuning for vision transformers,” *arXiv preprint arXiv:2312.15681*, 2023.

[46] J. Pan, Z. Lin, X. Zhu, J. Shao, and H. Li, “St-adapter: Parameter-efficient image-to-video transfer learning,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 26 462–26 477, 2022.

[47] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “Lora: Low-rank adaptation of large language models,” *arXiv preprint arXiv:2106.09685*, 2021.

[48] E. B. Zaken, S. Ravfogel, and Y. Goldberg, “Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models,” *arXiv preprint arXiv:2106.10199*, 2021.

Appendix

Ethics Considerations

This research focuses on developing a framework for securely updating AI models under strict usage control. It relies solely on open-source datasets and publicly available pre-trained models, involve no human subjects, sensitive data, and touches no proprietary systems, ensuring compliance with ethical standards. The work is focused on theoretical analysis and experimental validation in controlled environments, without direct application to real-world services. While AdaLoc improves model usage control and adaptation, there is a potential risk if the *key* is leaked or mismanaged. In such a scenario, unauthorized parties could bypass the intended update restrictions. To mitigate such risks, AdaLoc is intended for deployment in controlled environments and should integrate robust access management to prevent unauthorized usage.

Open Science Policy

This work complies with the open science policy by providing detailed descriptions of methodologies and experiments to ensure reproducibility. Our code is publicly available at: https://github.com/MLresearchAI/ADALOC for community access and verification while adhering to data privacy and security guidelines.

LLM Usage Considerations

Large language models were used solely for grammar and spelling correction, and all resulting text was manually reviewed by the authors for accuracy.