UNMERGEABLE LORA: PROACTIVELY PROTECTING ADAPTERS FROM UNAUTHORIZED MERGING

Anonymous authors
Paper under double-blind review

ABSTRACT

Public LoRA adapters enable convenient capability sharing, but also allow unauthorized reuse of proprietary capabilities through model merging. Existing proactive defenses for full-model release do not transfer to the fixed-base LoRA setting, leaving released adapters effectively unprotected against unauthorized merging. In this work, we propose LoRA Preservation-Barrier Hardening (LoRA-PBH), a release-time defense that directly protects adapters against post-release merging while preserving their standalone performance. Instead of modifying model representations that become ineffective under LoRA merging, LoRA-PBH optimizes the released adapter to make unauthorized merged models retain little defender performance without requiring changes to the public base model or knowledge of future attacker adapters. Experiments on vision, language, and diffusion models demonstrate that LoRA-PBH preserves standalone performance while degrading merged performance across multiple backbones, merging methods, merge coefficients, and multi-adapter settings. Code is included in the supplementary material.

1 INTRODUCTION

Low-rank adaptation (LoRA) has become the dominant paradigm for distributing task-specific model capabilities. Instead of releasing an entire fine-tuned model, model owners can publish lightweight LoRA adapters on top of a public base model, making task-specific model customization significantly more efficient and accessible. As the LoRA ecosystem has matured, model merging (Ilharco et al., 2023) has become an increasingly common practice for combining multiple released LoRA adapters into a single model. By directly combining the parameter updates of multiple adapters in weight space without additional fine-tuning, a merged model can simultaneously inherit multiple capabilities, allowing users to integrate diverse functionalities into a single model rather than deploying multiple adapters independently. This workflow has already been widely adopted in practice, as evidenced by the large number of LoRA adapters publicly shared on repositories such as Hugging Face[1].

While model merging greatly improves the flexibility and reusability of released models, it also introduces a new security risk. Recent work represented by PaRaMS (Wei et al., 2025) showed that a free-rider can merge a released model with another model to reuse proprietary capabilities without authorization. The merged model inherits these capabilities while obscuring their origin, making attribution to the original model owner difficult. Although this was originally studied for full-model merging, it naturally extends to LoRA adapters, where lightweight adapters are publicly released and merged in the same way, as shown in Figure 1.

This threat raises the question of whether existing proactive defenses for full-model release can be directly transferred to the LoRA setting. To answer this question, we adapt representative defenses, including PaRaMS (Wei et al., 2025), MergeBarrier (Li et al., 2026) and MergeLock (Wang et al., 2025), to the LoRA release setting. However, our empirical study shows that these straightforward adaptations provide almost no protection against unauthorized LoRA merging. Our analysis reveals a structural mismatch: existing defenses preserve the functionality of the released adapter by transforming its internal LoRA factors, whereas LoRA merging operates only on the resulting low-rank weight updates. As a result, these transformations do not affect the merge process, leaving the merged model essentially unchanged.

[1] https://huggingface.co/

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.

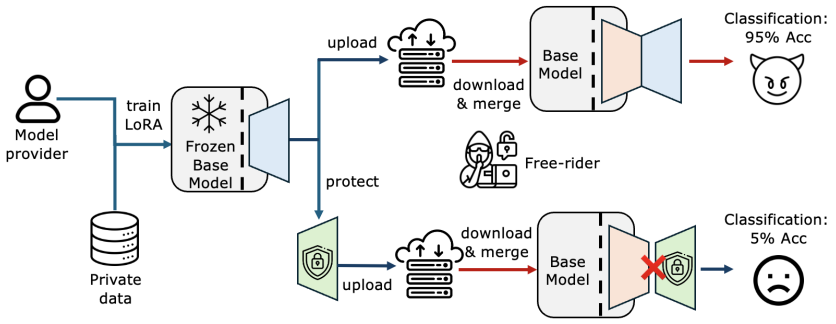


Figure 1: Unauthorized LoRA merging and its defense.

Motivated by this observation, we reformulate protection directly from the perspective of the merged model. We introduce the *Preservation Barrier*, which measures the increase in defender-task loss relative to the protected standalone model. A larger barrier means that merging produces a larger defender-task loss increase. At the same time, the protected adapter should retain its standalone performance. Balancing these two objectives leads to a max–min optimization problem, which forms the basis of our proposed defense. Based on this formulation, we propose *LoRA Preservation-Barrier Hardening* (LoRA-PBH), a release-time protection strategy for LoRA adapters. LoRA-PBH approximates the max–min objective by constructing representative perturbations induced by defender-generated surrogate adapters, optimizing the preservation barrier over these perturbations, and simultaneously preserving the standalone performance of the protected adapter. As a release-time strategy, LoRA-PBH can be applied directly to existing LoRA adapters without retraining or modifying the public base model.

Contributions. Our main contributions are as follows.

1. We show that existing proactive defenses fail in the fixed-base LoRA setting. We further identify the structural reason: transformations that preserve standalone functionality have no effect on the LoRA merging process, revealing a fundamental limitation of prior defenses.
2. We propose **LoRA Preservation-Barrier Hardening (LoRA-PBH)**, a release-time defense that protects LoRA adapters against unauthorized merging while preserving standalone performance.
3. We demonstrate the effectiveness of LoRA-PBH on vision, language, and diffusion models. Across diverse backbones, merging methods, merge coefficients, and multi-adapter settings, LoRA-PBH consistently suppresses merged performance while preserving standalone utility.

2 Related Work

Model merging and its shift to LoRA. Model merging combines multiple fine-tuned models into a single model by merging their parameters. Existing methods include parameter arithmetic (Ilharco et al., 2023), conflict-aware merging (Yadav et al., 2023; Yu et al., 2024), representation alignment (Stoica et al., 2025), and adaptive weighting (Yang et al., 2024; Touayouch et al., 2026), all aiming to preserve the constituent capabilities after merging.

As LoRA adapters have become the standard mechanism for distributing model updates, model merging has naturally evolved toward the adapter setting, where multiple LoRA adapters are combined on top of a shared base model. Recent methods include KnOTS, which aligns LoRA updates via singular value decomposition before merging (Stoica et al., 2025); LoRA-LEGO, which decomposes adapters into rank-wise units for fine-grained recombination (Zhao et al., 2025); and Core Space Merging, which formulates LoRA merging in a shared low-rank latent space to improve both accuracy and efficiency (Panariello et al., 2025).

Merging risks and defenses. While model merging provides a mechanism for composing capabilities, it also introduces a security risk: a released model can be merged without its owner's consent, enabling unauthorized reuse or extraction of its capabilities. To mitigate this threat, recent work has explored proactive defenses against unauthorized merging. In this setting, the model owner modifies the model before release so that any subsequent unauthorized merging is rendered ineffective.

PaRaMS (Wei et al., 2025), MergeBarrier (Li et al., 2026) and MergeLock (Wang et al., 2025) are representative proactive defenses proposed for full-parameter model release. These methods can be naturally adapted to the LoRA release setting by applying their transformations directly to the released LoRA factors. We empirically evaluate these natural adaptations in our setting and find that they provide almost no protection against unauthorized LoRA merging. A detailed empirical and theoretical analysis of this failure is presented in Section 5.

MergeGuard (Chen et al., 2026) adopts an importance-aware weight redistribution and pruning strategy that relies on the parameter redundancy of full-parameter models. Since LoRA restricts trainable parameters to fixed-rank low-rank updates, this mechanism is not applicable. Indeed, the original paper explicitly excludes parameter-efficient fine-tuning (PEFT), including LoRA, from its threat model. We therefore do not consider MergeGuard as a comparable baseline; further discussion is provided in Appendix A.4.

3 Preliminaries

3.1 LoRA in Weight Space

For a single weight matrix, let $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ denote the base weight. LoRA (Hu et al., 2022) freezes W and learns two low-rank matrices $B \in \mathbb{R}^{d_{\text{out}} \times r}$ and $A \in \mathbb{R}^{r \times d_{\text{in}}}$ with $r \ll \min(d_{\text{out}}, d_{\text{in}})$. A LoRA adapter is the factor pair $\phi = (A, B)$, defining the LoRA adapter update and the LoRA-adapted weight with scaling factor α as

$$\Delta W_\phi = (\alpha/r)BA, \qquad W_\phi = W + \Delta W_\phi \tag{1}$$

For notational simplicity, we formulate our method for a single LoRA-adapted weight matrix.

3.2 LoRA Adapter Merging

Consider n LoRA adapters trained from the same base matrix W, where adapter ϕ_j induces the weight update ΔW_{ϕ_j}. A LoRA merging method combines these updates into a single merged model,

$$W_{\text{merge}} = \mathcal{M}_\eta(W, \Delta W_{\phi_1}, \ldots, \Delta W_{\phi_n}), \tag{2}$$

where $\mathcal{M}_\eta$ denotes the merging method with configuration η. Since each adapter affects the model only through its update, we abbreviate $\mathcal{M}_\eta(W, \phi_1, \ldots, \phi_n)$ to $\mathcal{M}_\eta(W, \Delta W_{\phi_1}, \ldots, \Delta W_{\phi_n})$. For example, Task Arithmetic (TA) (Ilharco et al., 2023) computes

$$W_{\text{merge}} = W + \sum_{j=1}^{n} \lambda_j \Delta W_{\phi_j}, \tag{3}$$

where λ_j is the merge coefficient for adapter ϕ_j. Other merging methods modify the updates through reweighting, filtering, or alignment before constructing the merged model (Yadav et al., 2023; Yu et al., 2024; Stoica et al., 2025; Yang et al., 2024; Touayouch et al., 2026).

4 Problem Formulation

4.1 Definition of Defense

We consider the LoRA release setting, where a model owner publishes a task-specific LoRA adapter on top of a public base model W. Let ϕ_{def} denote the defender's original adapter trained on private data. Before release, the defender applies a protection procedure $\mathcal{D}$ to obtain the protected adapter $\tilde{\phi}_{\text{def}} = \mathcal{D}(\phi_{\text{def}})$. Unlike full-model release, the public base model W remains fixed throughout the entire process, and only the protected adapter is distributed.

After release, an attacker downloads the protected adapter, combines it with another LoRA adapter ϕ_{atk} using a merging method $\mathcal{M}_\eta$, and deploys the merged model $W_{\text{merge}} = \mathcal{M}_\eta\Big(W, \tilde{\phi}_{\text{def}}, \phi_{\text{atk}}\Big)$.

4.2 Defense Objective

Let $\text{Perf}_j(\cdot)$ denote the performance metric for task j, where larger values indicate better performance. Given adapters $\phi_{1:n} := (\phi_1, \ldots, \phi_n)$ and a merge configuration η, we define the defender-task performance of the standalone adapter and the merged model as

$$\text{P}_j^{\text{stand}}(\phi_j) = \text{Perf}_j(W_{\phi_j}), \qquad \text{P}_j^{\text{merge}}(\phi_{1:n}; \eta) = \text{Perf}_j(W_{\text{merge}}). \tag{4}$$

Based on these two quantities, a protected adapter should satisfy two requirements. (i) The released adapter should retain nearly the same standalone performance as the original adapter:

$$\text{P}_{\text{def}}^{\text{stand}}(\mathcal{D}(\phi_{\text{def}})) \geq \text{P}_{\text{def}}^{\text{stand}}(\phi_{\text{def}}) - \epsilon, \tag{5}$$

where $\epsilon > 0$ specifies the maximum acceptable degradation.

(ii) After merging with an external adapter, the merged model should retain little capability on the defender task:

$$\text{P}_{\text{def}}^{\text{merge}}((\tilde{\phi}_{\text{def}}, \phi_{\text{atk}}); \eta) \ll \text{P}_{\text{def}}^{\text{stand}}(\tilde{\phi}_{\text{def}}), \tag{6}$$

Combining these two requirements, the defender seeks a protection procedure that maximizes the gap between standalone performance and merged performance subject to Eq. (5):

$$\mathcal{D}^\star \in \arg\max_{\mathcal{D}} \quad \text{P}_{\text{def}}^{\text{stand}}(\mathcal{D}(\phi_{\text{def}})) - \text{P}_{\text{def}}^{\text{merge}}((\mathcal{D}(\phi_{\text{def}}), \phi_{\text{atk}}); \eta) \tag{7}$$

4.3 Threat Model

Defender. The defender has access to the public base model W, the private training data, and the original adapter ϕ_{def} trained on that data. However, the defender cannot modify the public base model W, does not know the attacker's adapter ϕ_{atk}, and does not know which merging method $\mathcal{M}_\eta$ or configuration η the attacker will use.

Attacker. After release, the attacker has access to the public base model W and the protected adapter $\tilde{\phi}_{\text{def}}$. The attacker can train or obtain another LoRA adapter ϕ_{atk} on separate data and can freely choose the merging method $\mathcal{M}_\eta$ and its configuration η. However, the attacker has no access to the defender's private training data, the original adapter ϕ_{def} or the defense procedure $\mathcal{D}$.

Attacker adaptivity. Unless otherwise specified, we consider a non-adaptive free-rider that is unaware of the protection mechanism and directly merges the released protected adapter with another LoRA adapter. We further evaluate a stronger adaptive attacker (Section 7.4) that follows Kerckhoffs's principle: the protection algorithm is assumed to be public, and the attacker optimizes its local copy of the protected adapter to circumvent the defense before merging, using only its own data and adapter.

5 Revisiting Defenses under LoRA Merging

PaRaMS (Wei et al., 2025), MergeBarrier (Li et al., 2026) and MergeLock (Wang et al., 2025) preserve the released model's standalone functionality by applying paired invertible transformations to coupled weight matrices. For two consecutive linear layers with weights W_i and W_{i+1}, an invertible transformation T gives

$$W_i' = TW_i, \quad W_{i+1}' = W_{i+1}T^{-1}, \qquad W_{i+1}'W_i' = W_{i+1}T^{-1}TW_i = W_{i+1}W_i. \tag{8}$$

Unlike the full-parameter setting, LoRA releases only the low-rank adapter while the pretrained base model remains fixed. Consequently, adapting the paired-transformation principle to LoRA leaves two natural choices.

Route I: Transforming the LoRA adapter updates. The first route applies the paired transformations directly to the induced LoRA adapter updates. However, this adaptation no longer preserves the standalone model, because the paired transformation no longer acts on the fixed pretrained weights.

Route II: Transforming the LoRA factors. The second route applies the paired transformation inside the LoRA factorization,

$$A' = TA, \quad B' = BT^{-1}, \qquad \Delta W' = \frac{\alpha}{r} B'A' = \frac{\alpha}{r} BT^{-1}TA = \frac{\alpha}{r} BA = \Delta W. \tag{9}$$

This construction preserves the standalone functionality because the effective update remains unchanged. Existing LoRA merging methods, including TA, TIES-Merging, and KnOTS, operate directly on ΔW rather than its low-rank factorization (Ilharco et al., 2023; Yadav et al., 2023; Stoica et al., 2025). Therefore, a performance degradation caused by merging does not occur in principle, and it does not function as a proactive defense at all.

We empirically confirmed the two corresponding failure modes. Detailed implementations and experimental results are provided in Appendix A.2.

6 METHOD

6.1 FROM MERGE VISIBILITY TO PRESERVATION BARRIER

The two requirements formalized by Eqs. (5) and (6) provide our starting point: a protection strategy should maximize the drop from standalone performance to merged performance on the defender task, while the protected adapter must retain standalone performance. Let $\tilde{\phi} := \mathcal{D}(\phi_{\text{def}})$. From the merged model defined in Section 4.1, we define the merge perturbation as

$$q := W_{\text{merge}} - W_{\tilde{\phi}} = \mathcal{M}_\eta(W, \tilde{\phi}, \phi_{\text{atk}}) - W_{\tilde{\phi}}. \tag{10}$$

Here, q is the complete change from the protected standalone model to the merged model, including the effects of the external adapter, merging method, and merge configuration. Because performance metrics such as accuracy are not differentiable, we maximize the barrier of the loss values, where $\mathcal{L}_{\text{def}}$ is the defender-task loss evaluated on defender data:

$$\mathcal{B}_{\tilde{\phi}}(q) := \mathcal{L}_{\text{def}}(W_{\tilde{\phi}} + q) - \mathcal{L}_{\text{def}}(W_{\tilde{\phi}}). \tag{11}$$

A larger barrier represents a greater reduction in merged performance on the defender task.

The defender does not know the attacker's adapter, merging method, or configuration and therefore does not know the q. Instead, defender can employ the worst case merge configuration that attains the least barrier as the surrogate. Let $\mathcal{Q}(\tilde{\phi})$ denote the set of all valid perturbations induced by these configurations. The protection objective is to maximize the *worst-case* barrier over this set. To preserve the standalone performance while degrading merged performance, we ultimately solve the following constrained maximization problem.

$$\max_{\mathcal{D}} \quad \inf_{q \in \mathcal{Q}(\mathcal{D}(\phi_{\text{def}}))} \mathcal{B}_{\mathcal{D}(\phi_{\text{def}})}(q) \qquad \text{s.t.} \quad \text{Perf}_{\text{def}}(W_{\mathcal{D}(\phi_{\text{def}})}) \geq \text{Perf}_{\text{def}}(W_{\phi_{\text{def}}}) - \epsilon. \tag{12}$$

Eq. (12) is not directly optimizable: $\mathcal{Q}(\tilde{\phi})$ is unavailable before release, the inner minimum is nonsmooth, and the performance-based standalone constraint is nondifferentiable. We introduce LoRA Preservation-Barrier Hardening (LoRA-PBH) to address these obstacles in the following.

6.2 APPROXIMATING THE UNKNOWN PERTURBATION

The defender has no knowledge about the attacker's potential adapters and merging methods. LoRA-PBH therefore constructs sampled perturbations using only defender-side information. Using dataset that the defender owns, it extracts defender features, forms pseudo tasks by clustering them, trains one surrogate adapter for each pseudo task, and retains the candidates that cause the smallest defender-task loss increase when merged with the defender adapter.

More specifically, the defender first prepares the adapter bank $\Psi = \{\psi_1, \ldots, \psi_J\}$, where J is the retained bank size and each retained surrogate adapter is trained from different subsets or variants of

the defender data. At each optimization step t, LoRA-PBH samples M retained surrogate adapters $\psi_{t,i}$ from Ψ and merging configurations $\eta_{t,i}$. Each pair produces

$$W_{\text{merge}}^{(t,i)} = \mathcal{M}_{\eta_{t,i}}(W, \tilde{\phi}, \psi_{t,i}), \qquad q_{t,i} = W_{\text{merge}}^{(t,i)} - W_{\tilde{\phi}}, \qquad i = 1, \ldots, M.$$

These perturbations form the indexed collection $\hat{\mathcal{Q}}_t = (q_{t,1}, \ldots, q_{t,M})$. LoRA-PBH instantiates $\mathcal{M}_{\eta_{t,i}}$ based on merging methods with randomly chosen $\lambda_{t,i}$ and obtains the corresponding perturbation $q_{t,i}$ at each optimization step.

6.3 Approximating the Worst-case Barrier

Replacing the unknown perturbation set $\mathcal{Q}(\tilde{\phi})$ in Eq. (12) by the sampled perturbations $\hat{\mathcal{Q}}_t$ changes the inner optimization into the sampled worst-case barrier

$$\min_{1 \le i \le M} \mathcal{B}_{\tilde{\phi}}(q_{t,i}).$$

This minimum is non-differentiable and therefore unsuitable for gradient-based optimization. LoRA-PBH instead optimizes the differentiable SoftMin

$$\mathcal{S}_\beta(\tilde{\phi}; \hat{\mathcal{Q}}_t) = -\frac{1}{\beta} \log \sum_{i=1}^{M} \exp\left(-\beta \mathcal{B}_{\tilde{\phi}}(q_{t,i})\right),$$

where $\beta > 0$ denotes the inverse temperature. The corresponding barrier loss is $\mathcal{L}_{\text{bar}} = -\mathcal{S}_\beta$. As $\beta \to \infty$, the SoftMin converges to the sampled minimum. Moreover,

$$0 \le \min_i \mathcal{B}_{\tilde{\phi}}(q_i) - \mathcal{S}_\beta \le \frac{\log M}{\beta},$$

so maximizing the SoftMin approximately maximizes the worst sampled preservation barrier.

For merge perturbations outside the sampled set, Proposition 1 establishes a lower bound on the worst-case barrier over the entire perturbation space under a standard local smoothness assumption. The proposition and proof are provided in Appendix C.1.

6.4 Training Objective

Eq. (5) requires the protected adapter to retain standalone performance on the defender task within the tolerance ϵ. LoRA-PBH keeps direct supervision on the defender data through standalone loss

$$\mathcal{L}_{\text{s}}(\tilde{\phi}) := \mathcal{L}_{\text{def}}(W_{\tilde{\phi}}), \tag{13}$$

and constrains how far its LoRA adapter update moves from the original defender update using the normalized weight-space anchor

$$\mathcal{A}_w(\tilde{\phi}) := \frac{\|\Delta W_{\tilde{\phi}} - \Delta W_{\phi_{\text{def}}}\|_F^2}{\|\Delta W_{\phi_{\text{def}}}\|_F^2}, \tag{14}$$

where the original defender update is assumed to be nonzero. Its normalization penalizes update drift and discourages barrier inflation through uncontrolled update-norm growth.

Combining the standalone loss and normalized weight anchor above with the barrier loss from Section 6.3 yields the final LoRA-PBH objective,

$$\mathcal{L}_{\text{LoRA-PBH}} = \mathcal{L}_{\text{s}} + \nu \mathcal{L}_{\text{bar}} + \mu_w \mathcal{A}_w, \tag{15}$$

where ν controls barrier hardening and μ_w controls update preservation. Optimization updates only the protected LoRA adapter; Appendix C gives the detailed algorithm.

7 Experiments

We evaluate LoRA-PBH through the following questions. **RQ1.** Does LoRA-PBH suppress unauthorized LoRA merging while preserving standalone performance? **RQ2.** Does LoRA-PBH generalize across merge coefficients, backbones, and multi-adapter settings? **RQ3.** How does each component contribute to the effectiveness? **RQ4.** How robust is LoRA-PBH against adaptive attacks? Section 7.2, Section 7.3, Appendix D.5, and Section 7.4 correspond to RQ1-4, respectively.

7.1 Experimental Setup

Evaluation scope and protocol. We evaluate LoRA-PBH on image classification, language understanding, and text-to-image generation. The image-classification evaluation uses ViT-B/32, ViT-B/16, and ViT-L/14 with adapters for Cars, DTD, EuroSAT, GTSRB, and RESISC45; the language evaluation uses Llama-2-7B, Mistral-7B, and Llama-3.1-8B (Touvron et al., 2023; Jiang et al., 2023; Grattafiori et al., 2024) with adapters for HellaSwag, ARC-Challenge, BoolQ, SST-2, and RTE; and the text-to-image evaluation uses SDXL 1.0 (Podell et al., 2024) with six public LoRA adapters. For each classification backbone, every task adapter serves as the defender against the other four adapters, giving 20 defender–free-rider combinations. Each defender is protected once using only defender-side data, and is then evaluated against all four unseen free-riders.

Evaluation metrics and optimization. For classification, $\mathrm{Perf}(\cdot)$ in Section 4.2 is instantiated as task accuracy. Following the definitions in Section 4.2, we refer to the performance of the merged model on the defender task as the *defender-task merged performance*. Analogously, we define the *free-rider-task merged performance* as the accuracy of the same merged model on the free-rider task. Together with the standalone performance, these metrics evaluate whether LoRA-PBH suppresses unauthorized preservation of the defender capability while maintaining standalone utility. We report accuracies in $\%$ and compute performance changes by subtraction on the same scale.

For optimization, unless otherwise stated, LoRA-PBH uses TA, with one shared defender–surrogate coefficient sampled uniformly from $[0.2, 1.0]$. Dataset, adapter, evaluation details and results of other merging methods used during optimization are provided in Appendices B and D.

7.2 Main Results Across Diverse Tasks

7.2.1 Image Classification

We evaluate LoRA-PBH under diverse LoRA merging methods. Due to space limitations, Table 1 reports the complete task-level results for Task Arithmetic (TA) (Ilharco et al., 2023), while Table 2 summarizes the average results over six additional representative merging methods. As shown in Table 1, LoRA-PBH preserves standalone performance, with an average change of only $+0.5\%$, while reducing defender-task merged performance from 74.0% to 3.0%.

For completeness, we additionally report the free-rider-task merged performance, which decreases from 74.0% to 11.7%. Detailed task-level results are provided in Appendix D.6. Overall, these results demonstrate that LoRA-PBH consistently suppresses unauthorized capability preservation while maintaining the standalone utility of the released adapter across diverse merging methods.

Table 2 demonstrates that LoRA-PBH generalizes well beyond the TA merging method used during optimization. Without any method-specific retraining, protected merged performance remains substantially lower than the unprotected baseline under all evaluated merging methods, with degradation of at least 63.1%. Even under LoRA-LEGO, which retains the highest merged performance after protection, the defender-task merged performance is only 11.7%, compared with 74.8% before

Table 1: ViT-B/32 results under TA ($c = 0.5$). Merged performance is averaged over the four free-rider adapters for each defender task. Performance is in $\%$; Δ denotes Protected minus Unprotected standalone performance.

Defender	Standalone Unprotected	Δ	Merged Unprotected	Merged Protected
Cars	58.5	-0.9	52.1	0.5
DTD	72.8	$+2.2$	66.2	2.1
EuroSAT	95.5	-0.1	83.9	9.3
GTSRB	93.6	$+0.8$	86.3	0.7
RESISC45	87.6	$+0.5$	81.5	2.5
Avg.	81.6	$+\mathbf{0.5}$	74.0	**3.0**

Table 2: Average defender-task merged performance on ViT-B/32 under six additional merging methods.

Merging method	Merged Unprotected	Merged Protected
TIES-Merging	74.4	3.7
DARE	73.6	5.3
KnOTS	74.0	3.0
DivMerge	73.9	5.3
LoRA-LEGO	74.8	11.7
Core Space	75.2	7.2

Table 3: Results on three LLM backbones. Standalone and base-model performance are averaged over five downstream tasks. Merged performance is averaged over all defender–free-rider combinations. Performance is in %; Δ denotes Protected minus Unprotected.

Backbone	Stand. Unprot.	Δ	Merged Unprot.	Merged Prot.	Δ	Base Perf.
Llama-2-7B	80.0	+0.2	72.8	57.1	−15.7	62.2
Mistral-7B	85.3	−0.1	80.8	62.4	−18.4	73.0
Llama-3.1-8B	80.7	+0.1	76.6	61.0	−15.6	72.4
Avg.	**82.0**	**+0.1**	**76.7**	**60.2**	**−16.6**	**69.2**

protection. These results demonstrate that the protection transfers effectively across structurally different LoRA merging methods.

Across vision backbones. We further evaluate LoRA-PBH on ViT-B/16 and ViT-L/14 using the same protocol. Across all three backbones, standalone performance changes by at most 0.6%, while protected defender-task merged performance remains between 3.0% and 3.4%, compared with 74.0%–79.5% before protection. These results demonstrate that the same protection strategy generalizes across different CLIP architectures and model scales. Complete backbone-level results are reported in Appendix D.6.

7.2.2 Language Models

We next evaluate LoRA-PBH on LoRA adapters of large language models. Table 3 summarizes the results on three representative LLM backbones. Standalone performance and base-model performance are averaged over the five downstream tasks, while merged performance is averaged over all defender–free-rider combinations under TA.

Table 3 shows that LoRA-PBH exhibits the same behavior on language models as on vision models. Across all three LLM backbones, standalone performance changes by at most 0.2%, whereas defender-task merged performance decreases by 15.6%–18.4%. These results demonstrate that LoRA-PBH generalizes across LLM architectures.

For completeness, we additionally measure merged performance on the free-rider task. This metric decreases from 76.7% to 68.8%, indicating that the merged model no longer fully preserves the performance of its constituent adapters. However, the degradation is smaller than that on the defender task, suggesting that LoRA-PBH primarily suppresses preservation of the protected performance while leaving considerably more of the free-rider performance intact. Per-backbone free-rider results are provided in Appendix D.6.

7.2.3 Diffusion Models

We finally evaluate LoRA-PBH on three public SDXL LoRA pairs from Hugging Face. Figure 2 compares the standalone and merged generations before and after protection under identical generation settings.

Across all three LoRA pairs, the protected standalone generations remain visually consistent with the original defender adapters, indicating that LoRA-PBH preserves standalone generation quality. In contrast, the protected merged generations no longer exhibit the expected composition of the constituent adapters, suggesting that unauthorized merging fails to preserve their combined capabilities.

7.3 Generalization Across Merge Configurations

Across all five defender tasks, protected merged performance remains consistently low throughout the evaluated merging-coefficient range from $c = 0.3$ to 0.8, and the same trend persists when increasing the number of merged adapters from two to five. During hardening, the coefficient is sampled uniformly from $[0.2, 1.0]$. These results demonstrate that LoRA-PBH is robust to the attacker's

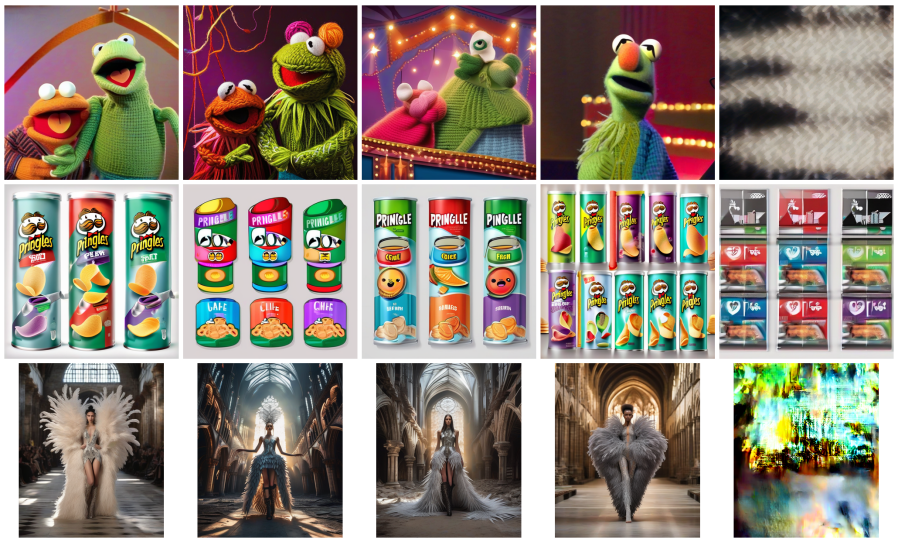


Figure 2: Qualitative comparison on three public SDXL LoRA pairs. Columns from left to right show the unprotected defender, free-rider, unprotected merged, protected defender, and protected merged output.

choice of merge strength and naturally extends from pairwise merging to multi-adapter merging. Complete results are provided in Appendix D.4.

We further conduct ablation studies to understand the contribution of each component of LoRA-PBH. The results show that the preservation barrier is the primary driver of merged-performance degradation, while $\mathcal{L}_{\mathrm{s}}$ preserves standalone functionality and the weight-space anchor $\mathcal{A}_w$ further stabilizes optimization by limiting parameter drift. Complete generalization and ablation results are provided in Appendix D.4 and Appendix D.5.

7.4 Evaluation Against Adaptive Attacks

We consider a white-box adaptive attacker with full knowledge of LoRA-PBH and access to the released protected adapter. Consistent with our threat model, the attacker has access only to its own free-rider data and never observes the defender's private training data. In practice, we approximate the adaptive objective by jointly optimizing the released adapter and the merge configuration using only free-rider data. Details are given in Appendix D.3.

On ViT-B/32, averaged over all defender–free-rider combinations, the unprotected merged model achieves 74.0% on both tasks. Adaptive optimization partially recovers defender-task merged performance from 3.0% to 34.4% and free-rider-task merged performance from 11.7% to 59.7%, compared with non-adaptive merging (Table 5 in Appendix D.3). Nevertheless, the recovered defender-task performance remains well below the unprotected baseline (34.4% vs. 74.0%), indicating that the absence of the defender's private data fundamentally limits the attacker's ability to remove the preservation barrier. LoRA-PBH remains effective against a strong white-box adaptive attacker.

8 Conclusion

We studied proactive protection against unauthorized LoRA adapter merging. We showed that existing defenses become ineffective in the fixed-base LoRA setting due to a structural mismatch with the LoRA merging interface. To address this limitation, we proposed *LoRA Preservation-Barrier Hardening* (LoRA-PBH), a release-time defense that suppresses unauthorized capability preservation while maintaining standalone utility. Experiments on vision, language, and diffusion models demonstrated its effectiveness across diverse merging methods and settings.

AI USE STATEMENT

We used generative AI tools to polish the language of the manuscript, including its grammar, wording, and clarity, and to assist with LaTeX formatting, reference verification, and consistency checks of the text and mathematical exposition. The authors take responsibility for the final manuscript, including its claims, results, and references.

REFERENCES

Wei-Jia Chen, Min-Yan Tsai, Cheng-Yi Lee, and Chia-Mu Yu. Defending unauthorized model merging via dual-stage weight protection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 27926–27935, June 2026. arXiv:2511.11851.

Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 105(10):1865–1883, 2017. doi: 10.1109/JPROC.2017.2675998.

Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3606–3613, 2014. doi: 10.1109/CVPR.2014.461.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of NAACL-HLT*, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, December 2023. URL https://zenodo.org/records/10256836.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. EuroSAT: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019. doi: 10.1109/JSTARS.2019.2918242.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *International Conference on Learning Representations (ICLR)*, 2023.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, et al. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023.

Jonathan Krause, Michael Stark, Jia Deng, and Fei-Fei Li. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 554–561, 2013. doi: 10.1109/ICCVW.2013.77.

Qinfeng Li, Miao Pan, Jintao Chen, Fu Teng, Zhiqiang Shen, Ge Su, Hao Peng, and Xuhong Zhang. Do not merge my model! safeguarding open-source LLMs against unauthorized model merging. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(37):31662–31670, 2026. doi: 10.1609/aaai.v40i37.40433. URL https://ojs.aaai.org/index.php/AAAI/article/view/40433.

Aniello Panariello, Daniel Marczak, Simone Magistri, Angelo Porrello, Bartłomiej Twardowski, Andrew D. Bagdanov, Simone Calderara, and Joost van de Weijer. Accurate and efficient low-rank model merging in core space. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. SDXL: Improving latent diffusion models for high-resolution image synthesis. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=di52zR8xgf.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012. doi: 10.1016/j.neunet.2012.02.016.

George Stoica, Pratik Ramesh, Boglarka Ecsedi, Leshem Choshen, and Judy Hoffman. Model merging with SVD to tie the KnOTS. In *International Conference on Learning Representations (ICLR)*, 2025.

Brahim Touayouch, Loïc Fosse, Géraldine Damnati, and Gwénolé Lecorvé. DivMerge: A divergence-based model merging method for multi-tasking. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7157–7180, Rabat, Morocco, March 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.eacl-long.337. URL https://aclanthology.org/2026.eacl-long.337/.

Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations (ICLR)*, 2019.

Zihao Wang, Enneng Yang, Lu Yin, Shiwei Liu, and Li Shen. Model unmerging: Making your models unmergeable for secure model sharing. *arXiv preprint arXiv:2509.01548*, 2025. URL https://arxiv.org/abs/2509.01548.

Junhao Wei, Zhe Yu, and Jun Sakuma. Disrupting model merging: A parameter-level defense without sacrificing accuracy. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17698–17707, 2025.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. TIES-merging: Resolving interference when merging models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Enneng Yang, Zhenyi Wang, Li Shen, Shiwei Liu, Guibing Guo, Xingwei Wang, and Dacheng Tao. AdaMerging: Adaptive model merging for multi-task learning. In *International Conference on Learning Representations (ICLR)*, 2024.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super Mario: Absorbing abilities from homologous models as a free lunch. In *International Conference on Machine Learning (ICML)*, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

Jinghan Zhang, Shiqi Chen, Junteng Liu, and Junxian He. Composing parameter-efficient modules with arithmetic operation. In *Advances in Neural Information Processing Systems*, volume 36, pp. 12589–12610, 2023.

Ziyu Zhao, Tao Shen, Didi Zhu, Zexi Li, Jing Su, Xuwu Wang, and Fei Wu. Merging LoRAs like playing LEGO: Pushing the modularity of LoRA to extremes through rank-wise clustering. In *International Conference on Learning Representations (ICLR)*, 2025.

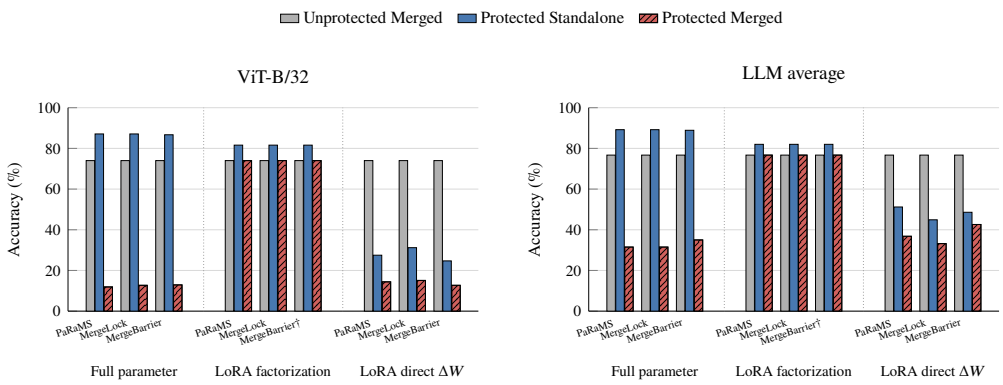


Figure 3: Comparison of PaRaMS, MergeLock, and MergeBarrier in the full-parameter and LoRA settings. Each method in each setting reports three defender-side quantities: Unprotected Merged is defender-task merged performance before applying the defense, Protected Standalone is standalone performance after applying the defense, and Protected Merged is defender-task merged performance after applying the defense. Gray, blue, and red hatched bars denote these quantities, respectively. Lower Protected Merged indicates stronger merge suppression, while Protected Standalone should remain high. † denotes the scoped factor-internal MergeBarrier construction.

Appendix Overview

This appendix provides additional analysis, method details, experimental configurations, and results that complement the main text:

- **Appendix A** presents the adapter-side constructions, structural analysis, and empirical comparison of prior defenses under the fixed-base LoRA merging interface.
- **Appendix B** describes LoRA background, the successful-merging criterion, the evaluated merging methods, vision and language tasks, and diffusion adapters, evaluation pairs, prompts, and generation settings.
- **Appendix C** provides the coverage result and proof, the intermediate defense objectives, the complete LoRA-PBH algorithm, and the full self-generated surrogate-adapter procedure.
- **Appendix D** reports task-adapter training and evaluation details, surrogate and hardening configurations, the adaptive-attacker configuration, coefficient and multi-adapter evaluations, objective ablations, the direct update-shift diagnostic, alternative methods for constructing q, and additional vision and language results.

A Prior Defenses under the LoRA adapter Merging Interface

This section collects all supplementary discussion of PaRaMS, MergeLock, MergeBarrier, and MergeGuard. It gives the exact adapter-side constructions used in Section 5, proves their structural properties, and explains why MergeGuard is not included in the main comparison.

Except for the explicitly separated PEM-TA experiment, all adapter-side evaluations and all claims of merge-output invariance below use the LoRA release setting and LoRA adapter merging interface defined in Section 3: the public base model W remains fixed and the merging methods are based on LoRA adapter updates. For the controlled comparison in Section 5, the evaluated adapter-side constructions also retain the original per-module rank and parameter budget; this is an evaluation constraint rather than a universal restriction on LoRA release.

A.1 Adapter-Side Constructions for PaRaMS, MergeLock, and MergeBarrier

Standalone preservation and merge compatibility. Eq. (8) shows the paired-transformation principle. Although the individual weight matrices change, their overall function remains unchanged

because the paired transformations cancel across adjacent layers. Previous defenses therefore preserve standalone performance while substantially modifying the released parameters. Since successful full-model merging relies on nearby parameters in weight space, these parameter changes reduce merge compatibility and degrade the merged performance.

PaRaMS. In the full-parameter setting, PaRaMS applies paired permutations and, where the activation and module structure permit, positive rescalings to compatible network weights. The permutation equivalence needed here is shown below.

Equivalence of the MLP permutation. For an MLP with an element-wise activation σ, write

$$f(x) = W_2\sigma(W_1x + b_1) + b_2.$$

PaRaMS chooses a permutation matrix P and transforms the two weights and the intermediate bias as

$$W_1' = PW_1, \qquad b_1' = Pb_1, \qquad W_2' = W_2P^\top.$$

Because an element-wise activation commutes with a coordinate permutation, $\sigma(Pz) = P\sigma(z)$. Therefore,

$$\begin{aligned} W_2'\sigma(W_1'x + b_1') + b_2 &= W_2P^\top\sigma(P(W_1x + b_1)) + b_2 \\ &= W_2P^\top P\sigma(W_1x + b_1) + b_2 \\ &= f(x). \end{aligned}$$

Thus, the paired MLP permutation preserves the standalone model's input–output function exactly and therefore preserves its standalone performance.

Our adapter-side PaRaMS construction applies the same transformation families to the latent coordinates of each LoRA factor pair. For every adapted module ℓ, choose an invertible $T^{(\ell)} = P^{(\ell)}D^{(\ell)} \in \mathbb{R}^{r\times r}$, where $P^{(\ell)}$ is a permutation matrix and $D^{(\ell)}$ is a positive diagonal matrix. Here $T^{(\ell)}$ acts in the LoRA rank space. We set

$$A_{\mathrm{P}}^{(\ell)} = T^{(\ell)}A^{(\ell)}, \qquad B_{\mathrm{P}}^{(\ell)} = B^{(\ell)}(T^{(\ell)})^{-1}. \tag{16}$$

The base, rank, scaling coefficient, adapted-module set, and all other model parameters remain unchanged.

The PaRaMS construction preserves every update visible to the merging method because

$$\begin{aligned} \Delta W_{\mathrm{P}}^{(\ell)} &= \frac{\alpha}{r}B^{(\ell)}(T^{(\ell)})^{-1}T^{(\ell)}A^{(\ell)} \\ &= \Delta W^{(\ell)}. \end{aligned} \tag{17}$$

This identity holds for every nonsingular choice of the permutation and scaling parameters. Consequently, the standalone model and every merged model produced by an LoRA adapter-only merging method remain exactly unchanged.

MergeLock. MergeLock (Wang et al., 2025) likewise protects a full-parameter model through paired function-preserving reparameterizations of coupled weights. Its direct adapter-side realization therefore follows the same construction as PaRaMS in Eq. (16): an invertible map is inserted between the two LoRA factors and canceled by its inverse. This leaves every LoRA adapter update unchanged by Eq. (17). As shown in Figure 3, its factor-level adaptation leaves both standalone and merged performance unchanged, while its direct update-level adaptation causes the loss of standalone performance.

Relation to PEM-TA. PaRaMS also evaluates a PEFT setting using PEM-based task arithmetic (PEM-TA) (Zhang et al., 2023). Unlike Eq. (2), PEM-TA treats the stored LoRA factors A and B as the objects of composition and combines corresponding factor tensors component-wise. Let d and f denote the defender and free-rider adapters, respectively. The following construction assumes that they adapt the same modules with matching factor shapes, rank r, and scaling α/r. For a two-adapter interpolation with $\lambda \in [0, 1]$,

$$\begin{aligned} A_{\mathrm{PEM}} &= \lambda A_{\mathrm{d}} + (1-\lambda)A_{\mathrm{f}}, \\ B_{\mathrm{PEM}} &= \lambda B_{\mathrm{d}} + (1-\lambda)B_{\mathrm{f}}. \end{aligned}$$

Table 4: Average defender-task merged performance under PEM-TA on ViT-B/32 with $\lambda = 0.5$, averaged across all defender–free-rider combinations. Unprotected denotes merging without defense, whereas Protected denotes merging after applying the listed defense.

Method	Merged Unprotected	Merged Protected
PaRaMS	80.7	26.3
LoRA-PBH	80.7	31.5

the LoRA adapter update is

$$\Delta W_{\mathrm{PEM}} = \frac{\alpha}{r} B_{\mathrm{PEM}} A_{\mathrm{PEM}},$$

whose factor product expands as

$$\begin{aligned} B_{\mathrm{PEM}} A_{\mathrm{PEM}} = \lambda^2 B_{\mathrm{d}} A_{\mathrm{d}} + (1-\lambda)^2 B_{\mathrm{f}} A_{\mathrm{f}} \\ + \lambda(1-\lambda)\left(B_{\mathrm{d}} A_{\mathrm{f}} + B_{\mathrm{f}} A_{\mathrm{d}}\right). \end{aligned} \tag{18}$$

Thus PEM-TA contains cross terms that are absent when already-induced updates are merged in weight space. Although $A'_{\mathrm{d}} = TA_{\mathrm{d}}$ and $B'_{\mathrm{d}} = B_{\mathrm{d}}T^{-1}$ preserve $B_{\mathrm{d}}A_{\mathrm{d}}$, they generally change $B_{\mathrm{d}}T^{-1}A_{\mathrm{f}} + B_{\mathrm{f}}TA_{\mathrm{d}}$. Consequently, factor reparameterization can affect PEM-TA without contradicting the LoRA adapter invariance established above. This interpolation convention is distinct from the shared-coefficient TA used in the main experiments. At $\lambda = c = 0.5$, the numerical input weights coincide, but PEM-TA applies them separately to A and B, yielding squared self-term coefficients and additional cross terms rather than the TA update $0.5\Delta W_{\mathrm{d}} + 0.5\Delta W_{\mathrm{f}}$.

To evaluate this distinct factor-level merge interface in our setting, we compare PaRaMS and LoRA-PBH under PEM-TA on ViT-B/32 in Table 4. The PaRaMS row uses the scaling-only special case of Eq. (16), with $P^{(\ell)} = I$, matching its original PEFT configuration. The distinguishing feature is that PEM-TA composes the stored factors rather than their induced updates.

Both defenses substantially reduce defender-task merged performance under PEM-TA, with PaRaMS attaining a value 5.2% lower than LoRA-PBH. This does not contradict the LoRA adapter invariance above: PEM-TA exposes factor reparameterization through the cross terms in Eq. (18), whereas the zero-change result in Section 5 applies when the merging method consumes only LoRA adapter updates.

Original MergeBarrier attention transformation. MergeBarrier uses different mechanisms for attention and FFN blocks. For an attention head, its attention component applies the same output-feature orthogonal matrix to the full query and key weights:

$$\overline{W}_q = PW_q, \qquad \overline{W}_k = PW_k, \qquad P^\top P = I. \tag{19}$$

This preserves the attention scores because $\overline{W}_q^\top \overline{W}_k = W_q^\top P^\top P W_k = W_q^\top W_k$. This display assumes bias-free projections; when query and key biases are present, the same P must also be applied to both biases. For multi-head attention, the method uses a block-diagonal P assembled from head-wise orthogonal matrices. MergeBarrier selects P through a full-weight displacement objective involving W_q and W_k, reflecting selected leading eigendirections to increase the distance induced by subsequent full-weight merging (Li et al., 2026). Thus, the original attention operation replaces both full effective weights; it is not an operation defined on a LoRA factor pair.

Why the original attention operation is not a direct LoRA port. LoRA does not inherently provide residuals for both W_q and W_k: low-rank factors exist only for the modules selected as LoRA targets. In our vision implementation, query, key, and value are stored in a fused `in_proj_weight` parameter rather than exposed as separate LoRA-targeted linear modules, so no query–key factor pair is released. The LLM checkpoints likewise do not uniformly target both projections. Therefore, the paired operation in Eq. (19) cannot be instantiated uniformly on the released factors.

Even when both projections are LoRA-targeted, fixed target coverage alone does not make the original operation representable. For a projection $s \in \{q, k\}$ that is actually targeted, write its residual as

$\Delta W_s = (\alpha_s/r_s)B_sA_s$. Let W_s denote the corresponding fixed public base-model weight. Reproducing the original transformed effective weight while keeping this base weight fixed would require

$$\begin{aligned}\Delta W_s^\star &= P(W_s + \Delta W_s) - W_s \\ &= (P - I)W_s + P\Delta W_s.\end{aligned} \tag{20}$$

The required update $\Delta W_s^\star$ is not guaranteed to have rank at most r_s: it contains the base-dependent term $(P - I)W_s$, for which neither a rank-r_s bound nor cancellation with $P\Delta W_s$ is guaranteed. Hence the original full-weight attention transformation is not, in general, exactly realizable at the prescribed rank under the LoRA release setting.

Original MergeBarrier FFN transformation. MergeBarrier's FFN component replaces the original activation–linear computation with a Taylor-polynomial parameterization (Li et al., 2026).

Why the FFN component is not a direct LoRA port. Let $y = V\,\mathrm{Act}(z + b) + c$, let z_0 be an expansion point, and let N be the polynomial order. With $\mathrm{Act}^{(n)}$ denoting the coordinate-wise nth derivative, MergeBarrier writes

$$\begin{aligned}y - c &= V\,\mathrm{Act}(z + b) \\ &\approx \sum_{n=0}^{N} \widehat{V}_n (z - z_0)^{\odot n}, \\ \widehat{V}_n &= V D_n, \\ D_n &= \mathrm{Diag}\left(\frac{\mathrm{Act}^{(n)}(z_0 + b)}{n!}\right).\end{aligned} \tag{21}$$

The original operation therefore introduces $N + 1$ polynomial branches and changes the FFN computation graph. A standard LoRA release only adds a low-rank residual to an existing linear layer and cannot introduce these branches. Even if one attempted to realize each coefficient with the original base W, substituting $V = W + (\alpha/r)BA$ would require the residual

$$\widehat{V}_n - W = W(D_n - I) + \frac{\alpha}{r} B(AD_n), \tag{22}$$

for which no rank-r bound follows: the base-dependent term $W(D_n - I)$ is unconstrained in rank, and no cancellation with the rank-r second term is guaranteed. Thus the computation-graph change already prevents a direct standard LoRA port, and exact coefficient realization is not guaranteed at the prescribed rank either.

Evaluated fixed-budget attention adaptation. Because neither original component is directly realizable across our released LoRA adapters, we define MergeBarrier† by transferring its paired, orthogonal, function-preserving attention principle to the two factors already present in each LoRA-targeted attention module. For every such module ℓ, choose $P_r^{(\ell)} \in \mathbb{R}^{r\times r}$ satisfying $P_r^{(\ell)}(P_r^{(\ell)})^\top = I$ and set

$$A_{\mathrm{MB}}^{(\ell)} = (P_r^{(\ell)})^\top A^{(\ell)}, \qquad B_{\mathrm{MB}}^{(\ell)} = B^{(\ell)} P_r^{(\ell)}.$$

Here $P_r^{(\ell)}$ acts in the LoRA rank space, whereas the original P in Eq. (19) acts in the attention feature space. Orthogonality preserves every LoRA adapter update:

$$\Delta W_{\mathrm{MB}}^{(\ell)} = \frac{\alpha}{r} B^{(\ell)} P_r^{(\ell)} (P_r^{(\ell)})^\top A^{(\ell)} = \Delta W^{(\ell)}. \tag{23}$$

Consequently, the result is independent of which rank-space orthogonal matrices are selected. The dagger in Figure 3 denotes precisely this attention adaptation.

Taken together, MergeBarrier does not successfully transfer to the LoRA release setting under the fixed-rank evaluation constraint: the original attention operation cannot generally be instantiated on the released target modules and rank budget, while the compatible adaptation above is invisible under the LoRA adapter merging interface and empirically provides no protection. This conclusion does not preclude a rank-expanded or architecture-changing redesign, which would fall outside the LoRA release setting evaluated here.

A.2 DIRECT LORA ADAPTER ADAPTATION

To evaluate the first adaptation route in Section 5, we apply the paired maps directly to compatible pairs of induced LoRA updates while keeping the public base weights fixed. For the consecutive-layer construction in Eq. (8), we set

$$\Delta W_i^{\mathrm{upd}} = T\Delta W_i, \qquad \Delta W_{i+1}^{\mathrm{upd}} = \Delta W_{i+1}T^{-1}, \tag{24}$$

where T acts on the full hidden coordinate shared by the two layers.

Writing $\Delta W_j = (\alpha_j/r_j)B_jA_j$, the transformed updates can be released without increasing either LoRA rank by setting

$$\begin{aligned} B_i^{\mathrm{upd}} &= TB_i, \quad A_i^{\mathrm{upd}} = A_i, \\ B_{i+1}^{\mathrm{upd}} &= B_{i+1}, \quad A_{i+1}^{\mathrm{upd}} = A_{i+1}T^{-1}. \end{aligned} \tag{25}$$

This construction changes the LoRA adapter updates supplied to the merging method, but the unchanged base weights prevent the paired maps from canceling in the standalone model.

Why transforming only the updates breaks the cancellation. For the same two-linear-map abstraction used in Eq. (8), applying the paired maps only to the LoRA updates of the fixed base-layer weights W_i and W_{i+1} gives

$$\widehat{W}_i = W_i + T\Delta W_i, \qquad \widehat{W}_{i+1} = W_{i+1} + \Delta W_{i+1}T^{-1}.$$

The difference between their composition and the original composition is

$$\begin{aligned} &\widehat{W}_{i+1}\widehat{W}_i - (W_{i+1} + \Delta W_{i+1})(W_i + \Delta W_i) \\ &\quad = W_{i+1}(T - I)\Delta W_i + \Delta W_{i+1}(T^{-1} - I)W_i, \end{aligned} \tag{26}$$

which is nonzero in general. The fixed base-weight terms therefore prevent the two maps from canceling at the effective-weight level.

Figure 3 confirms both limitations. In the full-parameter setting, PaRaMS, MergeLock, and MergeBarrier retain high protected standalone performance: 87.1%, 87.1%, and 86.7% on ViT-B/32 and 89.2%, 89.2%, and 88.9% on average across the three LLM backbones, respectively; while reducing protected defender-task merged performance to 11.9%, 12.7%, and 12.9% on ViT-B/32 and to 31.5%, 31.5%, and 35.0% on the LLMs, respectively. Thus, all three defenses effectively suppress merging in their original full-parameter setting.

When their transformations are applied within the LoRA factorization, protected standalone performance remains 81.6% on ViT-B/32 and 82.0% on the LLMs, but protected defender-task merged performance remains at the corresponding unprotected levels of 74.0% and 76.7%. This confirms that the factor-internal transformations cancel in the induced update and therefore provide no protection under the LoRA adapter merging interface.

Applying the transformations directly to ΔW does reduce protected defender-task merged performance to 14.4%, 15.1%, and 12.7% on ViT-B/32 and to 36.8%, 33.2%, and 42.6% on the LLMs. However, the same transformations change the effective network realized by the fixed base and adapter: protected standalone performance simultaneously falls to 27.5%, 31.2%, and 24.7% on ViT-B/32 and to 51.2%, 44.9%, and 48.6% on the LLMs. The apparent merge suppression therefore comes at the cost of severe standalone degradation and fails the standalone performance constraint in Eq. (5).

A.3 BASE-DEPENDENT CORRECTION FOR FULL-WEIGHT REALIZATION

To reproduce $T_{\mathrm{out}}(W + \Delta W)T_{\mathrm{in}}^{-1}$ while retaining the fixed base W with a rank-r LoRA update $\Delta W'$, the required base correction must satisfy

$$C_{\mathrm{base}} := T_{\mathrm{out}}WT_{\mathrm{in}}^{-1} - W = \Delta W' - T_{\mathrm{out}}\Delta W T_{\mathrm{in}}^{-1}.$$

Both terms on the right have rank at most r, so a necessary condition is $\mathrm{rank}(C_{\mathrm{base}}) \le 2r$. This condition is not guaranteed: for full-rank $W \in \mathbb{R}^{d\times d}$, $T_{\mathrm{out}} = \gamma I$, $T_{\mathrm{in}} = I$, $\gamma \neq 1$, and $d > 2r$, we obtain $C_{\mathrm{base}} = (\gamma - 1)W$ with rank $d > 2r$. Hence, reproducing the original full-weight

transformation may require a base-dependent correction that fixed-rank LoRA cannot represent; special cases remain possible only when the correction also satisfies the rank and adapted-module coverage constraints. Conversely, placing the compensating maps entirely within each LoRA factor pair preserves the induced update and is therefore invisible to LoRA adapter merging.

A.4 MergeGuard under the Fixed-Rank Evaluation Constraint

MergeGuard explicitly excludes parameter-efficient fine-tuning from its threat model and operates on full-parameter task vectors. Its first stage redistributes task information through full-parameter optimization, and its second stage applies a masked, scaled subtraction of the redistributed task vector.

A LoRA-constrained analogue could optimize the released factors and impose a representable mask, but this changes the feasible geometry and does not constitute a faithful realization of MergeGuard's original full-parameter stages. We therefore exclude MergeGuard from the fixed-rank adapter-side comparison.

B Evaluated Merging Methods and Datasets

B.1 Merging Methods

LoRA background. We adapt a network whose weights are obtained by large-scale pretraining. For a single weight matrix, let $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ denote the base weight. Adapting this model to a downstream task by full fine-tuning replaces W with $W + \Delta W$, where the update $\Delta W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ has the same shape as W and therefore the same number of trainable parameters. As models grow, learning and storing such a full-size update for every downstream task becomes increasingly expensive. LoRA addresses this issue by constraining the update ΔW to be low-rank.

Successful merging. For the adapters and merging interface defined in Section 3, let $\mathrm{Perf}_j(\cdot)$ denote the performance metric of adapter ϕ_j, where higher values indicate better performance. Successful merging should preserve the capability of each constituent adapter,

$$\mathrm{Perf}_j(W_{\text{merge}}) \approx \mathrm{Perf}_j(W_{\phi_j}), \qquad j = 1, \ldots, n.$$

We use task arithmetic as the default merging method and evaluate six additional methods in Table 2. All methods are applied to the same unprotected and protected ViT-B/32 adapters without retraining LoRA-PBH for a particular merging method.

Task arithmetic (TA). Task arithmetic directly adds scaled task-specific parameter updates to the shared base model (Ilharco et al., 2023). In the notation of Section 3, it constructs $W_{\text{merge}} = W + \sum_j \lambda_j \Delta W_{\phi_j}$. Our default pairwise evaluation uses the same coefficient for the defender and free-rider adapters, $\lambda_{\text{def}} = \lambda_{\text{fr}} = c = 0.5$.

TIES-Merging. TIES-Merging reduces interference among task updates in three stages: it removes small-magnitude entries, elects an aggregate sign for each parameter, and merges only entries whose signs agree with the elected sign (Yadav et al., 2023). This procedure suppresses redundant updates and prevents conflicting directions from canceling one another during merging. In our evaluation, TIES retains the largest-magnitude 20% of each task update, elects the sign from the signed sum, and takes the disjoint mean over sign-consistent entries. The resulting update is scaled by $c = 0.5$.

DARE. DARE randomly drops entries from each task update and rescales the retained entries by the inverse keep probability, preserving the update in expectation while reducing interference (Yu et al., 2024). Because DARE is a sparsification preprocessing step rather than a complete aggregation rule, the retained updates are subsequently combined by a base merging method. We use DARE-TIES with a 90% drop rate, inverse-probability rescaling, and random seed 42, followed by TIES with 20% retained density. The resulting update is scaled by $c = 0.5$.

KnOTS. KnOTS addresses misalignment among independently trained LoRA adapters by jointly transforming their layer-wise low-rank updates into an SVD-aligned space, where an existing merging rule can be applied more effectively (Stoica et al., 2025). The merged representation is then mapped back to the original parameter space. We concatenate each pair of matrix updates column-wise, compute the full economy SVD, and retain all singular values larger than 10^{-5}. We then apply task arithmetic in the aligned space, reconstruct the merged update, and use $c = 0.5$; no fixed-rank truncation is applied.

DivMerge. DivMerge determines task-dependent aggregation weights by minimizing the Jensen–Shannon divergence between the outputs of the merged model and those of the constituent task models (Touayouch et al., 2026). The resulting weights balance the adapters according to their output behavior rather than assigning a fixed coefficient to every adapter. We adapt the paper's task-wise classification configuration to ViT and optimize one coefficient per adapter on 200 unlabeled validation inputs per task. The coefficients are initialized to 0.5 and optimized for four epochs with Adam (learning rate 10^{-2} and default moments), using four examples per task in each batch. We apply no additional projection or constraint to the learned coefficients.

LoRA-LEGO. LoRA-LEGO treats each LoRA rank component as a minimal semantic unit, clusters units collected from different adapters, and uses the cluster centroids to assemble a merged adapter of adjusted rank (Zhao et al., 2025). Its reweighting stages further calibrate the contributions of the selected units and the resulting merged adapter. Following the authors' default, we set the number of clusters and merged rank to $k = 2r = 16$ for our rank-8 source adapters. Each centroid is rescaled with the infinity norm to match the cluster-average semantic-unit norm, and the merged LoRA output is multiplied by $\sqrt{r/k}$.

Core Space Merging. Core Space Merging constructs common alignment bases for the source LoRA adapters, projects their low-rank updates into compact core matrices, and performs aggregation in this shared core space (Panariello et al., 2025). The merged core is then mapped back through the shared bases, retaining a low-rank representation without merging full dense matrices. We instantiate the method as TIES-Core with the paper's ViT-B/32 configuration: the reference bases are obtained by economy SVD of the concatenated source bases without additional truncation, and TIES in the core space uses scale $\alpha = 0.6$ and retains the largest-magnitude 10% of entries before sign election and disjoint-mean aggregation.

B.2 Image-Classification Datasets

The vision evaluation covers five datasets spanning fine-grained recognition, textures, land cover, traffic signs, and remote-sensing scenes.

Stanford Cars. Stanford Cars is a fine-grained recognition dataset containing 196 vehicle classes distinguished by make, model, and year (Krause et al., 2013).

DTD. The Describable Textures Dataset (DTD) contains natural images organized into 47 human-interpretable texture categories (Cimpoi et al., 2014).

EuroSAT. EuroSAT consists of Sentinel-2 satellite images from 10 land-use and land-cover classes, including residential, industrial, agricultural, and natural regions (Helber et al., 2019).

GTSRB. The German Traffic Sign Recognition Benchmark (GTSRB) contains 43 traffic sign classes with substantial variation in scale, illumination, and viewpoint (Stallkamp et al., 2012).

RESISC45. NWPU-RESISC45 is a remote-sensing scene-classification dataset comprising 45 scene categories, ranging from natural landscapes to dense urban infrastructure (Cheng et al., 2017).

B.3 Language-Understanding Tasks

The language-model evaluation covers five classification and multiple-choice reasoning tasks.

HellaSwag. HellaSwag evaluates commonsense inference by asking a model to select the most plausible continuation of a context from four candidates (Zellers et al., 2019).

ARC-Challenge. ARC-Challenge contains multiple-choice grade-school science questions whose challenging subset requires more than simple retrieval or surface-level matching (Clark et al., 2018).

BoolQ. BoolQ consists of naturally occurring yes/no questions paired with passages and evaluates whether the answer can be inferred from the provided context (Clark et al., 2019).

SST-2. SST-2 is a sentence-level binary sentiment-classification task that labels movie-review sentences as positive or negative (Socher et al., 2013).

RTE. Recognizing Textual Entailment (RTE) is a binary task that determines whether a hypothesis is entailed by a given premise, using the formulation included in GLUE (Wang et al., 2019).

B.4 Diffusion Adapters and Evaluation Pairs

SDXL LoRA adapters and sources. We use the public Stable Diffusion XL 1.0 base model from Hugging Face[2] and six public LoRA checkpoints. These adapters respectively encode a Muppet-style character concept, a yarn-art style, cylindrical snack-package imagery, a sticker-sheet style, unusual runway outfits, and an Old-Gods-inspired visual concept. The top row of Figure 2 pairs the SDXL MuppetShow LoRA as the defender[3] with the SDXL Yarn Art Style LoRA as the free-rider[4]. The middle row pairs the SDXL PringlesTube LoRA as the defender[5] with the SDXL StickerSheet LoRA as the free-rider[6]. The bottom row pairs the SDXL WeirdOutfit LoRA as the defender[7] with the SDXL Old Gods Rising LoRA as the free-rider[8].

Generation configuration. All images are generated with `stabilityai/stable-diffusion-xl-base-1.0` using the Diffusers `EulerDiscreteScheduler` with a scaled-linear beta schedule, epsilon prediction, leading timestep spacing, and no Karras sigmas. We use 16 denoising steps and a classifier-free guidance scale of 7.0. Each standalone or pre-merged LoRA adapter is applied at an inference scale of 0.5 using TA. The common negative prompt is: "blurry, low quality, deformed, disfigured, duplicate, malformed, text, caption, logo, watermark."

Pair-specific prompts and seeds. For MuppetShow defender $\times$ Yarn Art Style free-rider, the resolution is 640×640, and all conditions use seed 12345. The positive prompt is: "MuppetShow, a green frog puppet and a glamorous pig puppet performing a duet on a colorful theater stage, Yarn art style, handmade knitted puppets, visible yarn fibers and crochet stitches, warm stage lighting, full body, highly detailed."

For PringlesTube defender $\times$ StickerSheet free-rider, the resolution is 640×640, and all conditions use seed 41001. The positive prompt is: "A colorful collection of six different PringlesTube cans with expressive mascot faces, classic red, green, orange and purple flavors, arranged as a clean StickerSheet on a pale blue background, thick white die-cut borders, product illustration, front view, sharp details."

[2] https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0

[3] https://huggingface.co/Norod78/sdxl-muppetshow-lora, checkpoint `SDXL-MuppetShow-Lora-step00002500.safetensors`.

[4] https://huggingface.co/Norod78/SDXL-YarnArtStyle-LoRA, checkpoint `SDXL_Yarn_Art_Style.safetensors`.

[5] https://huggingface.co/Norod78/SDXL-PringlesTube-Lora, checkpoint `SDXL-PringlesTube-Lora.safetensors`.

[6] https://huggingface.co/Norod78/SDXL-StickerSheet-Lora, checkpoint `SDXL-StickerSheet-Lora.safetensors`.

[7] https://huggingface.co/Norod78/weird-fashion-show-outfits-sdxl-lora, checkpoint `sdxl-WeirdOutfit-Dreambooh.safetensors`.

[8] https://huggingface.co/Norod78/but-the-old-gods-are-rising-sdxl, checkpoint `Old_gods_rising_SDXL.safetensors`.

For WeirdOutfit defender × Old Gods Rising free-rider, the resolution is 640×768, and all conditions use seed 51001. The positive prompt is: "Full-body editorial photograph of a runway model walking through a ruined gothic cathedral, wearing an extravagant sculptural couture dress made of feathers, mirrored metal and glowing fabric, WeirdOutfit style, but the old gods are rising, psychedelic tentacles emerging from the shadows, dramatic fashion lighting, highly detailed." Within each pair, all five displayed conditions use the same positive prompt, negative prompt, resolution, and seed.

C Method Details

We use the notation from Section 6: ϕ_{def} denotes the defender adapter, $\tilde{\phi}_{\mathrm{def}}$ denotes the current protected adapter, and we write $\tilde{\phi} = \tilde{\phi}_{\mathrm{def}}$ when no ambiguity arises. Moreover, ΔW_ψ is the matrix update of an adapter ψ. All formulas are stated for a single LoRA-adapted weight matrix. For multiple adapted matrices, W, ΔW, and q denote collections, and each squared Frobenius norm is summed over the adapted matrices. All losses and barriers within one algorithm step are evaluated on the same defender minibatch. In particular, $\mathcal{L}_{\mathrm{s}}(\tilde{\phi})$ in Eq. (13) is the minibatch estimate of $\mathcal{L}_{\mathrm{def}}(W_{\tilde{\phi}})$.

Intermediate defense objectives. From Eq. (10), $W_{\mathrm{merge}} = W_{\tilde{\phi}} + q$. Substituting this representation into Eq. (7) shows that for a merge, the performance-based objective is to

$$\max_{\tilde{\phi}} \mathrm{Perf}_{\mathrm{def}}(W_{\tilde{\phi}}) - \mathrm{Perf}_{\mathrm{def}}(W_{\tilde{\phi}} + q),$$

subject to Eq. (5).

Because performance metrics such as accuracy are not differentiable, we maximize the barrier of the loss values in Eq. (11). For the perturbation set $\mathcal{Q}(\tilde{\phi})$ defined in Section 6.1, the protection objective is to maximize the *worst-case* barrier over this set:

$$\max_{\mathcal{D}} \quad \inf_{q \in \mathcal{Q}(\mathcal{D}(\phi_{\mathrm{def}}))} \mathcal{B}_{\mathcal{D}(\phi_{\mathrm{def}})}(q)$$

Simply solving this on its own will not preserve the standalone performance. To preserve the standalone performance while degrading merged performance, we ultimately solve the constrained maximization problem in Eq. (12).

C.1 Generalization from Sampled to Unseen Merge Perturbations

The discussion of the SoftMin in Section 6.3 only concerns the sampled perturbations constructed during optimization. However, after release, an attacker may induce merge perturbations outside this sampled set. This naturally raises the following question:

> *Does maximizing the sampled SoftMin also improve the preservation barrier against unseen merge perturbations?*

The proposition below gives a conditional lower bound for a fixed protected release and a fixed nonempty indexed probe collection $\hat{\mathcal{Q}} = (q_1, \ldots, q_M)$, allowing repeated probes as in Section 6.2. Minima over this collection range over its entries. It requires a finite directed coverage radius and the local Lipschitz condition stated below. Its key intuition is simple: if every unseen perturbation is sufficiently close to one sampled perturbation, and the preservation barrier varies smoothly between nearby perturbations, then the sampled SoftMin, minus the coverage penalty, provides a lower bound on the worst-case barrier over the entire merge space.

To formalize this intuition, we first quantify how well the sampled perturbations cover the unknown perturbation space.

For multiple LoRA matrices, we define the product Frobenius norm

$$\|q - p\|_F = \left(\sum_{\ell} \|q^{(\ell)} - p^{(\ell)}\|_F^2 \right)^{1/2}.$$

The directed coverage radius is

$$\varepsilon(\tilde{\phi}, \hat{\mathcal{Q}}) = \sup_{q \in \mathcal{Q}(\tilde{\phi})} \min_{p \in \hat{\mathcal{Q}}} \|q - p\|_F, \tag{27}$$

which measures the largest distance from any valid merge perturbation to its nearest sampled perturbation.

We further assume that the preservation barrier changes smoothly between nearby perturbations. Specifically, suppose that for every $q \in \mathcal{Q}(\tilde{\phi})$ and its nearest sampled perturbation $p \in \hat{\mathcal{Q}}$,

$$|\mathcal{B}_{\tilde{\phi}}(q) - \mathcal{B}_{\tilde{\phi}}(p)| \leq G\|q - p\|_F,$$

where G is a local Lipschitz constant.

Proposition 1 (Generalization from sampled to unseen merge perturbations)**.** *Let $\beta > 0$ and $\varepsilon(\tilde{\phi}, \hat{\mathcal{Q}}) < \infty$. Under the local Lipschitz assumption above,*

$$\inf_{q \in \mathcal{Q}(\tilde{\phi})} \mathcal{B}_{\tilde{\phi}}(q) \geq \mathcal{S}_\beta(\tilde{\phi}; \hat{\mathcal{Q}}) - G\varepsilon(\tilde{\phi}, \hat{\mathcal{Q}}). \tag{28}$$

Proposition 1 gives a conditional connection between optimization over a fixed finite set of sampled surrogate merges and the worst-case barrier over unseen merge perturbations. Under the stated local Lipschitz condition and finite directed coverage radius, $\mathcal{S}_\beta(\tilde{\phi}; \hat{\mathcal{Q}}) - G\varepsilon(\tilde{\phi}, \hat{\mathcal{Q}})$ lower-bounds the unknown worst-case barrier. The proposition does not establish that the training sampler automatically attains a small coverage radius, and it does not prove robustness against adaptive attackers.

Proof. For any $q \in \mathcal{Q}(\tilde{\phi})$, let $p \in \hat{\mathcal{Q}}$ be its closest probe. The Lipschitz condition and Eq. (27) give

$$\begin{aligned}
\mathcal{B}_{\tilde{\phi}}(q) &\geq \mathcal{B}_{\tilde{\phi}}(p) - G\varepsilon(\tilde{\phi}, \hat{\mathcal{Q}}) \\
&\geq \min_{u \in \hat{\mathcal{Q}}} \mathcal{B}_{\tilde{\phi}}(u) - G\varepsilon(\tilde{\phi}, \hat{\mathcal{Q}}) \\
&\geq \mathcal{S}_\beta(\tilde{\phi}; \hat{\mathcal{Q}}) - G\varepsilon(\tilde{\phi}, \hat{\mathcal{Q}}).
\end{aligned}$$

Taking the infimum over q proves Eq. (28). □

C.2 Algorithm

Here $\mathrm{sg}(\cdot)$ denotes the stop-gradient operator, and Λ denotes the sampling distribution of the coefficient shared by the defender and surrogate updates in the task-arithmetic probe.

Algorithm 1 LoRA-PBH with task-arithmetic probe construction: one hardening step.

Input: public base matrix W; defender adapter ϕ_{def}; current protected adapter $\tilde{\phi}_{\mathrm{def}}$; defender minibatch; retained surrogate bank Ψ; probe perturbations per step M; inverse temperature β; weight-space-anchor coefficient μ_w; barrier weight ν; merging-coefficient sampler Λ.

1: **for** $i = 1, \ldots, M$ **do**
2: sample merging coefficient $\lambda_i \sim \Lambda$ and retained surrogate adapter $\psi_i \sim \mathrm{Unif}(\Psi)$ independently with replacement
3: construct $q_i \leftarrow (\lambda_i - 1)\ \mathrm{sg}(\Delta W_{\tilde{\phi}_{\mathrm{def}}}) + \lambda_i\, \Delta W_{\psi_i}$
4: **end for**
5: **Pass 1 (no grad).** Compute $\mathcal{B}_i \leftarrow \mathcal{L}_{\mathrm{def}}(W_{\tilde{\phi}_{\mathrm{def}}} + q_i) - \mathcal{L}_{\mathrm{def}}(W_{\tilde{\phi}_{\mathrm{def}}})$ for each i.
6: Compute stop-gradient SoftMin weights $w_i \leftarrow \mathrm{sg}\Big(\exp(-\beta\mathcal{B}_i)/\sum_{j=1}^{M}\exp(-\beta\mathcal{B}_j)\Big)$.
7: **Pass 2 (with grad).** Minimize the gradient-equivalent surrogate

$$\begin{aligned}&(1+\nu)\mathcal{L}_{\mathrm{s}}(\tilde{\phi}_{\mathrm{def}})\\&\quad+\mu_w\mathcal{A}_w(\tilde{\phi}_{\mathrm{def}})\\&\quad-\nu\sum_{i=1}^{M} w_i\lambda_i\,\mathcal{L}_{\mathrm{def}}(W_{\tilde{\phi}_{\mathrm{def}}}+q_i).\end{aligned}$$

8: Clip gradients and take an AdamW step on the LoRA parameters of $\tilde{\phi}_{\mathrm{def}}$.

At hardening step t, the M sampled probe perturbations form the indexed collection $\hat{\mathcal{Q}}_t$ in Section 6.2. A new collection is sampled for every update, so training optimizes a sequence of realized probe collections rather than one fixed full-bank SoftMin. When $M = 1$, $\mathcal{S}_\beta = \mathcal{B}_1$, the sole SoftMin weight is one, and β has no effect on that single-probe stochastic barrier-hardening update. Proposition 1 instead applies to any one fixed probe collection and its coverage radius. All forwards run in evaluation mode, so stochastic layers such as dropout are disabled; only the protected LoRA parameters receive gradients in Pass 2.

For log-sum-exp SoftMin, $\nabla\mathcal{S}_\beta = \sum_{i=1}^{M} w_i \nabla\mathcal{B}_i$; evaluating w_i from the current barriers and treating them as constants in Pass 2 therefore gives the exact first-order SoftMin gradient. Since $\mathcal{B}_i = \mathcal{L}_{\mathrm{def}}(W_{\tilde{\phi}_{\mathrm{def}}} + q_i) - \mathcal{L}_{\mathrm{s}}(\tilde{\phi}_{\mathrm{def}})$, the standalone coefficient becomes $1 + \nu$. Finally, detaching $\Delta W_{\tilde{\phi}_{\mathrm{def}}}$ inside q_i removes the task-arithmetic chain-rule factor λ_i; the explicit λ_i in Pass 2 restores that factor and recovers the gradient of the corresponding no-stop-gradient objective.

C.3 Self-Generated Surrogate Adapters: Full Procedure

Step 1: Feature extraction. Run the frozen model with the defender adapter ϕ_{def} attached over the defender-side surrogate-construction split, and cache the modality-specific features $\{z_i\}_{i=1}^{N} \subset \mathbb{R}^d$ described in Appendix D.

Step 2: Surrogate-task construction. For each cluster count C in a finite set and for multiple random seeds, run k-means with C clusters on $\{z_i\}$ to obtain surrogate-label assignments $\tilde{y}_i \in \{0, \ldots, C-1\}$. Each (C, seed) pair defines one surrogate task. The surrogate labels need not match the true classes; their role is to induce different training objectives from the defender data.

Step 3: Surrogate-task LoRA training. For each surrogate task, warm-start from the defender adapter ϕ_{def} and attach a temporary C-way pseudo-task head, replacing the task head where applicable, then train for a small number of steps. This temporary head is discarded after training; filtering and hardening use the original defender-task prediction/evaluation interface. Denote the resulting LoRA-only candidate by ψ_j^{cand} and store its update $\Delta W_{\psi_j^{\mathrm{cand}}}$. The candidate pool is $\Psi_{\mathrm{cand}} = \{\psi_1^{\mathrm{cand}}, \ldots, \psi_R^{\mathrm{cand}}\}$, where R is the candidate-pool size. Because each candidate surrogate adapter is warm-started from ϕ_{def}, it is not treated as an independent attacker sample. It is used as a defender-conditioned proxy for constructing probe perturbations during hardening.

Step 4: Barrier filtering. Let $\mathcal{V}$ be the fixed collection of defender batches used for filtering and let Λ_{f} be the finite filtering-coefficient set. Using the original defender-task prediction/evaluation interface, define

$$\begin{aligned} s_j &:= \frac{1}{|\mathcal{V}||\Lambda_{\mathrm{f}}|} \sum_{b \in \mathcal{V}} \sum_{\lambda \in \Lambda_{\mathrm{f}}} \mathcal{B}^{(b)}_{\phi_{\mathrm{def}}}\left(q^{\mathrm{f}}_j(\lambda)\right), \\ q^{\mathrm{f}}_j(\lambda) &:= (\lambda - 1)\Delta W_{\phi_{\mathrm{def}}} \\ &\quad + \lambda \Delta W_{\psi^{\mathrm{cand}}_j}, \end{aligned} \tag{29}$$

where the superscript (b) denotes evaluation on batch b. Retain the $J \leq R$ candidates with the smallest scores and reindex them as $\Psi = \{\psi_1, \ldots, \psi_J\}$. This is the surrogate bank used by Algorithm 1. Each probe uses exactly one retained surrogate, and the M probes at a hardening step are sampled independently with replacement.

D Additional Experimental Details

D.1 Task-Adapter Training and Evaluation

Vision task adapters. We use OpenCLIP models initialized with OpenAI pretrained weights. LoRA modules are attached to linear layers in the visual tower whose names match `(qkv|proj|fc1|fc2|mlp|attn)`. The same LoRA configuration is used for every matched matrix. The base image encoder and zero-shot classification head remain frozen, and only the LoRA factors are trained. Each task adapter uses rank $r = 8$, $\alpha = 16$, LoRA dropout 0, batch size 128, AdamW learning rate 10^{-4}, weight decay 0.1, cosine learning-rate decay with 500 warmup steps, and cross-entropy loss without label smoothing. We train every adapter for 30 epochs, reserve 10% of the training split as an internal validation split for surrogate construction and barrier filtering, and report final accuracies on the held-out test split.

LLM adapters and sources. For each LLM backbone, we retain the original LoRA parameterization of the released task checkpoints. For Llama-2-7B, we use the adapters for HellaSwag[9], ARC-Challenge[10], BoolQ[11], SST-2[12], and RTE[13]. For Mistral-7B, we use the adapters for HellaSwag[14], ARC-Challenge[15], BoolQ[16], SST-2[17], and RTE[18]. For Llama-3.1-8B, we use the adapters for HellaSwag[19], ARC-Challenge[20], BoolQ[21], SST-2[22], and RTE[23].

LLM base checkpoints and loader-declared LoRA interfaces. The LLM pipeline uses `meta-llama/Llama-2-7b-hf`, `mistralai/Mistral-7B-v0.1`, and `meta-llama/Meta-Llama-3.1-8B` as the three base checkpoints. Its model registry records rank $r = 8$, $\alpha = 16$, and target modules `q_proj` and `v_proj` for Llama-2-7B and Mistral-7B, and rank $r = 6$, $\alpha = 12$, and target modules `q_proj`, `k_proj`, `v_proj`, and `o_proj` for Llama-3.1-8B. The external adapters are loaded directly from their released configurations, rather than reconstructed from these registry fields.

[9] https://huggingface.co/Styxxxx/llama2_7b_lora-hellaswag
[10] https://huggingface.co/Styxxxx/llama2_7b_lora-arc_challenge
[11] https://huggingface.co/Styxxxx/llama2_7b_lora-bool_q
[12] https://huggingface.co/Styxxxx/llama2_7b_lora-sst2
[13] https://huggingface.co/Styxxxx/llama2_7b_lora-rte
[14] https://huggingface.co/predibase/hellaswag_processed
[15] https://huggingface.co/erjui/mistral-7b-lora-arc
[16] https://huggingface.co/rana-shahroz/mistral-7b-lora-r8-boolq-epochs5-adapter
[17] https://huggingface.co/predibase/glue_sst2
[18] https://huggingface.co/rana-shahroz/mistral-7b-seed4232001-lora-r8-rte-epochs5-adapter
[19] https://huggingface.co/igzi/lora-hellaswag
[20] https://huggingface.co/igzi/lora-arc_challenge
[21] https://huggingface.co/te4bag/LoRA-llama-3.1-8B-boolq
[22] https://huggingface.co/igzi/lora-sst2
[23] https://huggingface.co/ASethi04/llama-3.1-8b-rte-lora

LLM evaluation. We evaluate with the LM Evaluation Harness (Gao et al., 2023), using 10-shot normalized accuracy for HellaSwag, 25-shot normalized accuracy for ARC-Challenge, and zero-shot accuracy for BoolQ, SST-2, and RTE.

For all five tasks, the defender loss is the same multiple-choice ranking loss. For each answer candidate, we average the causal language-model negative log-likelihood only over its answer-completion tokens and use the negative value as the choice score. After masking padded dummy choices, cross-entropy over the candidate scores is used.

Defender and free-rider protocol. For classification experiments, each task adapter is used in turn as the defender, while every adapter trained on another task serves as a possible free-rider. This gives 20 defender–free-rider combinations per backbone. LoRA-PBH protects each defender adapter independently before any test-time free-rider is introduced. The defense uses only defender-side data and defender-generated surrogate adapters; it does not access the evaluated free-rider adapters or their training data. The same protected defender adapter is subsequently evaluated against all four unseen free-riders.

Reported metrics. For image classification and language understanding, the performance metric $\mathrm{Perf}(\cdot)$ in Section 4.2 is instantiated as task accuracy. We additionally report *free-rider-task merged performance*, the accuracy of the same merged model on the free-rider task, to verify whether the merged model preserves the performance of its constituent adapters as a whole rather than merely shifting performance away from the defender task.

D.2 Surrogate and Hardening Configuration

Vision surrogate construction. For each defender task, we run the original adapter on the internal validation split and cluster its image features with k-means. We use $C \in \{4, 8\}$ clusters and eight random seeds for each C, producing 16 surrogate tasks and hence $R = 16$ candidate surrogate adapters. For every surrogate task, we initialize from the original adapter, replace the classification head with a C-way linear head, and train the LoRA factors and temporary head for 200 steps using AdamW with learning rate 2×10^{-4}, batch size 128, and cross-entropy loss. We score the resulting candidate surrogate adapters by the pre-hardening preservation barrier in Appendix C.3 and retain the $J = 8$ candidates with the smallest filtering scores.

LLM surrogate construction. For LLMs, we use the same cluster counts, candidate-pool size, retained-bank budget, training steps, and learning rate, replacing image features with the final non-padding prompt hidden state. Filtering the candidate surrogate adapters averages the pre-hardening barrier over four defender batches and coefficients $\Lambda_{\mathrm{f}}^{\mathrm{LLM}} = \{0.2, 0.3, \ldots, 1.0\}$ before retaining the eight lowest-scoring surrogate adapters.

More precisely, we use the original defender adapter to extract the final transformer-layer hidden state at the last non-padding token of the complete evaluation-aligned prompt. We apply Euclidean k-means for 20 iterations directly to these unnormalized features. Each surrogate adapter is warm-started from the original defender LoRA and jointly trained with a newly initialized C-way linear head using pseudo-label cross-entropy and AdamW with learning rate 2×10^{-4}; all other AdamW arguments use the implementation defaults.

Hardening. LoRA-PBH is initialized from the trained task adapter and updates only its LoRA factors; the base model and all non-LoRA parameters, including the task/evaluation head when present, remain frozen. We harden for 100 steps using batch size 128, AdamW learning rate 2×10^{-4}, weight decay 0, gradient clipping at norm 1, and seed 0. We set the barrier weight to $\nu = 0.5$ and the normalized weight-space anchor coefficient to $\mu_w = 0.05$. At each step, we sample one retained surrogate adapter and draw the shared TA coefficient continuously and uniformly from $[0.2, 1.0]$. Thus, the reported runs use $J = 8$ and a single-probe barrier update ($M = 1$). The inverse temperature for the multi-probe SoftMin implementation is $\beta = 20$; as established in Appendix C.2, it has no operational effect when $M = 1$. The LLM experiments use the same LoRA-PBH objective and hardening configuration without retuning.

Hardening time. On a single NVIDIA H100 GPU, the complete vision protection pipeline, including construction of the 16 pseudo-task adapters and 100 hardening steps with batch size 128, takes approximately 2, 7, and 26 minutes on average for ViT-B/32, ViT-B/16, and ViT-L/14, respectively.

D.3 Adaptive-Attacker Configuration

Adaptive objective. Formally, for a protected adapter $\tilde{\phi}$, let $\mathcal{Q}_{\mathrm{ad}}(\tilde{\phi})$ denote the set of merged-model perturbations induced after adaptive optimization of the released adapter. The attacker seeks the perturbation that minimizes the preservation barrier, $q^{\star}_{\mathrm{ad}}(\tilde{\phi}) \in \arg\min_{q \in \mathcal{Q}_{\mathrm{ad}}(\tilde{\phi})} \mathcal{B}_{\tilde{\phi}}(q)$.

Attack implementation. The evaluated feature-guided attack provides an implementable surrogate for the adaptive objective above without access to defender data. It applies a trainable counter-transformation $\mathcal{T}_\theta$ to a local protected-adapter copy, where θ consists of the local LoRA factors initialized from $\tilde{\phi}_{\mathrm{def}}$. It uses two-coefficient TA with $\eta = (c_{\mathrm{p}}, c_{\mathrm{atk}}) \in [0.2, 1.0]^2$, where c_{p} and c_{atk} multiply the transformed protected copy and external adapter, respectively. Define $W^{\mathrm{adv}}_{\theta,\eta} := \mathcal{M}_\eta(W, \mathcal{T}_\theta(\tilde{\phi}_{\mathrm{def}}), \phi_{\mathrm{atk}})$. Using only the free-rider attack-training set $\mathcal{S}^{\mathrm{tr}}_{\mathrm{atk}}$, we minimize

$$\begin{aligned}\mathcal{L}_{\mathrm{atk}}(\theta, \eta) := \mathbb{E}_{(x,y)\sim\mathcal{S}^{\mathrm{tr}}_{\mathrm{atk}}}\Big[& d_{\cos}\Big(h_{W^{\mathrm{adv}}_{\theta,\eta}}(x), h_{W_{\tilde{\phi}_{\mathrm{def}}}}(x)\Big) \\ & + \gamma_{\mathrm{CE}}\, \ell_{\mathrm{CE}}\Big(g_{\mathrm{atk}}\Big(h_{W^{\mathrm{adv}}_{\theta,\eta}}(x)\Big), y\Big)\Big].\end{aligned} \tag{30}$$

Here, $h_U(x)$ denotes the feature representation produced by model U, g_{atk} is the fixed free-rider classification head, $d_{\cos}$ is cosine distance, ℓ_{CE} is cross-entropy, and γ_{CE} weights the free-rider classification term. The public base model, free-rider adapter, protected standalone model used as the fixed feature reference, and free-rider head remain fixed.

We optimize for 100 steps with batch size 128. AdamW updates the LoRA factors at learning rate 2×10^{-4}; the two TA coefficients use learning rate 10^{-2} and remain in $[0.2, 1.0]$. The feature and cross-entropy weights are 1.0 and $\gamma_{\mathrm{CE}} = 0.1$, respectively, and gradients are clipped at norm 1.

With seed 0, we split the free-rider validation set into 90% attack-training and 10% checkpoint-selection subsets and evaluate checkpoints every 10 steps. Among checkpoints whose adaptive merged model retains at least 80% of the free-rider adapter's standalone validation accuracy, we select the checkpoint $(\hat{\theta}, \hat{\eta})$ with the smallest mean cosine feature distance on the checkpoint-selection subset. The selected checkpoint induces $q_{\mathrm{eval}} := W^{\mathrm{adv}}_{\hat{\theta},\hat{\eta}} - W_{\tilde{\phi}_{\mathrm{def}}} \in \mathcal{Q}_{\mathrm{ad}}(\tilde{\phi}_{\mathrm{def}})$. The defender test set is loaded only afterward for final evaluation. We use the same ViT-B/32 defender–free-rider combinations as in the non-adaptive setting, and every selected checkpoint satisfies this threshold.

Table 5: Merged performance on ViT-B/32 averaged over all defender–free-rider combinations under different settings.

Attack setting	Defender-task merged	Free-rider-task merged
Unprotected	74.0	74.0
Non-adaptive	3.0	11.7
Adaptive	34.4	59.7

D.4 Additional Merge-Setting Evaluations

Section 7.2 has already demonstrated that LoRA-PBH generalizes across different application domains, backbone architectures, and merging methods. We next evaluate two additional merge configurations that are particularly relevant in practice: varying the task-arithmetic (TA) merge coefficient and increasing the number of merged adapters. Figure 4(a) varies the merge coefficient, while Figure 4(b) evaluates multi-adapter merging.

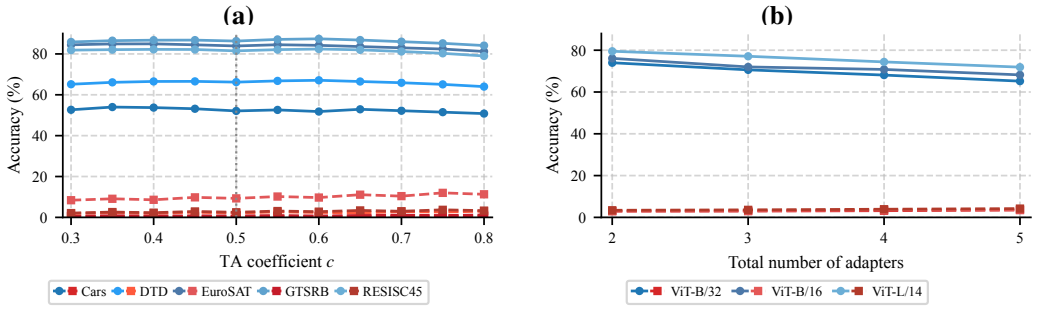


Figure 4: Defender-task merged performance under different merge configurations. (a) ViT-B/32 task-level results for $c \in [0.3, 0.8]$. (b) Backbone-level averages for merging two to five adapters at $c = 0.5$, including one defender. Solid and dashed curves denote unprotected and protected adapters, respectively.

Table 6: Average defender-task merged performance (%) under TA multi-adapter merging at $c = 0.5$. The count includes one defender adapter. Unprotected and Protected denote the defender-adapter condition.

	Defender-task merged					
	ViT-B/32		*ViT-B/16*		*ViT-L/14*	
# adapters	Unprotected	Protected	Unprotected	Protected	Unprotected	Protected
2	74.0	3.0	76.1	3.0	79.5	3.4
3	70.6	3.0	72.0	3.2	77.1	3.6
4	68.1	3.2	70.8	3.5	74.4	3.9
5	65.2	3.5	68.2	3.7	71.9	4.2

Sensitivity to the TA coefficient. We vary the TA coefficient to determine whether protection depends on the default value $c = 0.5$. On ViT-B/32, we evaluate merged models containing the unprotected and protected defender adapters for $c \in [0.3, 0.8]$ with step size 0.05. This deterministic evaluation sweep is distinct from hardening, where the coefficient is sampled continuously and uniformly from $[0.2, 1.0]$ at each step. The resulting defender-task merged-performance curves are shown in Figure 4(a).

Across all five defender tasks, protected merged performance remains consistently low throughout the entire coefficient range from $c = 0.3$ to 0.8, whereas the corresponding unprotected merged performance stays substantially higher. This behavior is observed for every defender task, indicating that LoRA-PBH is robust to the attacker's choice of merge strength and does not rely on a particular merging coefficient.

Multi-adapter merging. We further evaluate whether adding more free-rider adapters can restore merged performance on the defender task. For each defender task, we apply TA at $c = 0.5$ and vary the total number of merged adapters, including the defender adapter, from two to five. Figure 4(b) summarizes the curves, and Table 6 reports the corresponding defender-task merged performance for all three ViT backbones.

The same trend persists when increasing the number of merged adapters from two to five. Across all three ViT backbones, protected merged performance remains between 3.0% and 4.2%, compared with 65.2%–79.5% before protection. Even after introducing four additional free-rider adapters, unauthorized merging fails to recover the protected performance, demonstrating that LoRA-PBH naturally extends from pairwise merging to more realistic multi-adapter merging scenarios.

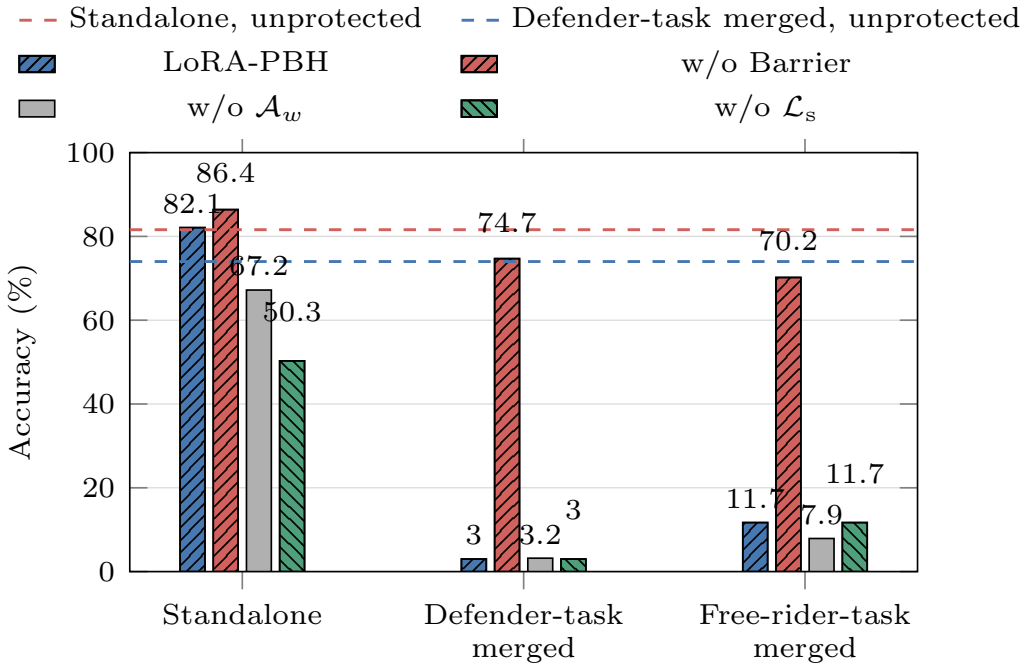


Figure 5: Ablation study on ViT-B/32. The horizontal reference lines indicate the average standalone performance (81.6%) and defender-task merged performance (74.0%) before protection.

D.5 Ablation Studies

To understand the contribution of each component of the LoRA-PBH objective, we perform an ablation study on ViT-B/32. Starting from the full objective in Eq. (15), the *w/o Barrier* variant removes the barrier loss $\mathcal{L}_{\text{bar}}$, the *w/o Weight Anchor* variant removes the normalized weight-space anchor $\mathcal{A}_w$, and the *w/o Standalone Loss* variant removes the standalone loss $\mathcal{L}_{\text{s}}$. Figure 5 compares all three variants with the full LoRA-PBH objective.

Removing the preservation barrier yields standalone performance of 86.4%, but defender-task merged performance increases to 74.7%, essentially recovering the unprotected level (74.0%). Merged performance on the free-rider task similarly increases to 70.2%. These results indicate that the standalone loss and the weight anchor alone cannot prevent performance preservation after unauthorized merging. The preservation barrier is therefore the key component responsible for suppressing merged performance.

In contrast, removing the normalized weight-space anchor preserves low defender-task merged performance (3.2%), but degrades standalone performance from 82.1% to 67.2%. Thus, $\mathcal{A}_w$ helps preserve standalone performance during barrier optimization.

Finally, removing $\mathcal{L}_{\text{s}}$ further reduces standalone performance to 50.3%, while defender-task and free-rider-task merged performance remain low at 3.0% and 11.7%. This confirms that $\mathcal{L}_{\text{s}}$ provides the task signal for standalone performance, whereas $\mathcal{A}_w$ stabilizes it during barrier optimization.

Update-shift diagnostic. The performance ablation alone does not identify the parameter-space mechanism. To test directly whether $\mathcal{A}_w$ limits induced-update drift, we compute at the final hard-

Table 7: Direct induced-update shift on ViT-B/32. Both statistics are computed from the final protected adapter relative to the same unprotected/original defender adapter and averaged once over the five defender adapters. For multiple LoRA matrices, squared Frobenius norms are summed across matrices.

Variant	Normalized update distance d_{upd}	Update-norm ratio r_{norm}
Full LoRA-PBH	3.61	1.38
w/o $\mathcal{A}_w$	8.02	3.15

ening checkpoint,

$$d_{\mathrm{upd}} = \frac{\sum_\ell \left\| \Delta W_{\tilde{\phi}}^{(\ell)} - \Delta W_{\phi_{\mathrm{def}}}^{(\ell)} \right\|_F^2}{\sum_\ell \left\| \Delta W_{\phi_{\mathrm{def}}}^{(\ell)} \right\|_F^2},$$
$$r_{\mathrm{norm}} = \frac{\sqrt{\sum_\ell \left\| \Delta W_{\tilde{\phi}}^{(\ell)} \right\|_F^2}}{\sqrt{\sum_\ell \left\| \Delta W_{\phi_{\mathrm{def}}}^{(\ell)} \right\|_F^2}}. \tag{31}$$

Here, d_{upd} is the normalized update distance that appears in $\mathcal{A}_w$, while r_{norm} separates update-norm growth from directional or structural displacement. Each statistic is computed once per defender at the final checkpoint and then averaged over the five ViT-B/32 defender adapters, so the same protected adapter is not counted again for each of its four free-rider evaluations.

Table 7 provides direct evidence for the stabilizing role of the weight anchor across the five ViT-B/32 defenders. Removing $\mathcal{A}_w$ increases d_{upd} from 3.61 to 8.02 ($2.22\times$) and r_{norm} from 1.38 to 3.15 ($2.28\times$). Thus, without the anchor, the average protected-to-original update-norm ratio is 3.15, compared with 1.38 under full LoRA-PBH. The simultaneous increase in both statistics shows that the additional displacement includes substantial update-norm growth rather than being attributable only to a change in update direction. Together with the aggregate performance ablation above, where removing $\mathcal{A}_w$ preserves low defender-task merged performance but degrades standalone performance, this diagnostic supports that $\mathcal{A}_w$ stabilizes barrier hardening by limiting induced-update drift, in part by constraining update-norm growth, rather than serving as the primary source of merge suppression. Because d_{upd} also depends on update direction, these two statistics do not attribute all additional displacement to norm growth.

Alternative methods for constructing q. In addition to the default TA construction, we use TIES-Merging, DARE, and DivMerge to construct the surrogate perturbations during optimization. For each operator $\mathcal{M}$, the resulting merged model defines $q = W_{\mathrm{merge}}^{(\mathcal{M})} - W_{\tilde{\phi}_{\mathrm{def}}}$. We retain the coefficient range $[0.2, 1.0]$, 16 candidate surrogate adapters, the same retained-bank and hardening budgets, and all other experimental settings.

Table 8: Average ViT-B/32 performance (%) when an alternative merging method is used to construct q during LoRA-PBH optimization. Merged Protected denotes defender-task merged performance. Results are averaged over all ordered defender–free-rider pairs and the same six additional evaluation merging methods reported in Table 2.

Method used to construct q	Standalone Protected	Merged Protected
TIES-Merging	80.7	6.3
DARE	80.2	5.3
DivMerge	79.3	3.7

Table 8 shows that all three alternatives maintain low defender-task merged performance, indicating that reliable merge suppression is not specific to TA-based samples. Their standalone protected

performance is lower than that obtained with TA, although merge suppression remains at a comparable level. One plausible explanation is that TA provides a smooth, coefficient-controlled additive perturbation path whose barrier gradient is easier to balance against $\mathcal{L}_{\mathrm{s}}$ and $\mathcal{A}_w$. TIES-Merging, DARE, and DivMerge introduce trimming, stochastic masking, or data-dependent reweighting, respectively, and therefore produce more heterogeneous sampled directions that may require more update capacity to harden while preserving standalone performance.

D.6 Additional Evaluation Results

Table 9: Cross-backbone ViT averages. Standalone performance averages five defender adapters; defender-task and free-rider-task merged performance average all defender–free-rider combinations. The standalone Δ is Protected minus Unprotected. Performance is in $\%$.

	Standalone			Merged		
Backbone	Unprotected	Protected	Δ	Defender-task Unprotected	Defender-task Protected	Free-rider-task Protected
ViT-B/32	81.6	82.1	$+0.5$	74.0	3.0	11.7
ViT-B/16	83.9	84.4	$+0.5$	76.1	3.0	12.7
ViT-L/14	87.0	87.6	$+0.6$	79.5	3.4	12.3

Table 10: ViT-B/32 free-rider-task merged performance ($\%$) under TA at $c = 0.5$, averaged over four defenders for each free-rider task.

Free-rider task	Merged Unprotected	Merged Protected
Cars	52.1	37.6
DTD	66.2	4.5
EuroSAT	83.9	7.4
GTSRB	86.3	3.7
RESISC45	81.5	5.3
Avg.	74.0	11.7

Table 11: LLM free-rider-task merged performance ($\%$) under TA at $c = 0.5$, averaged across all defender–free-rider combinations per backbone. Δ is Protected minus Unprotected.

Backbone	Free-rider-task merged		
	Unprotected	Protected	Δ
Llama-2-7B	72.8	64.4	-8.4
Mistral-7B	80.8	72.8	-8.0
Llama-3.1-8B	76.6	69.2	-7.4
Avg.	76.7	68.8	$\mathbf{-7.9}$