Defending Unauthorized Model Merging via Dual-Stage Weight Protection

Supplementary Material

The content of Supplementary Material is summarized as follows: 1) In Sec. A, we introduce the dataset used in our experiments across the vision and language domains; 2) In Sec. B, we describe the Transformer-based model architecture, focusing on the MLP and multi-head attention modules; 3) In Sec. C, we report the time cost of mask evaluation in MergeGuard; 4) In Sec. D, we provide additional results, including image and code generation, which offer further evidence of the effectiveness of MergeGuard.

A. Experimental Details

A.1. Environments

All experiments in this paper were conducted on a system with an Intel Core i9-10900X CPU and four NVIDIA RTX 3090 GPUs. The LLM experiments were executed on two NVIDIA H100 GPUs. The operating system was Ubuntu 18.04.6 LTS, and all software and dependencies were run in a Python 3.10 environment.

A.2. Datasets Details

Image Classification. We evaluate MergeGuard across eight standard image classification benchmarks, spanning various domains such as natural scenes, satellite imagery, and digit recognition. Table 6 provides a comprehensive summary of these datasets, including the number of semantic classes and the split of training/test samples. The selected datasets—SUN397 [43], Cars196 [19], RESISC45 [5], SVHN [27], GTSRB [31], MNIST [20], EuroSAT [14], and DTD [7]—represent the diverse visual distributions commonly used to benchmark model merging robustness.

Image Generation. We employ a diffusion model fine-tuned on WikiArt [40] and Naruto datasets [2]. WikiArt comprises curated high-resolution paintings across diverse artistic genres (*e.g.*, Impressionism, Realism, and Abstract Art), providing a rigorous benchmark for stylistic fidelity and semantic consistency. The Naruto dataset, featuring high-quality anime frames, evaluates model performance on out-of-distribution, non-photorealistic visual content.

LLM Evaluation. We employ three language datasets [4, 8, 24] to evaluate the capability of LLMs, including instruction following, mathematical reasoning, and program synthesis.

- AlpacaEval [24] is a benchmark designed to evaluate instruction-following ability in large language models. It contains approximately 805 human-authored instructions that span diverse open-ended tasks such as reasoning, explanation, transformation, and multi-step guidance. Model responses are compared against strong reference answers through pairwise preference scoring, which provides a reliable measure of general instruction alignment in generative models.
- GSM8K [8] is a high-quality dataset of grade-school mathematical word problems that require multi-step quantitative reasoning, arithmetic manipulation, and logical deduction. It contains 8,792 problems in total, with 7,473 examples for training and 1,319 for testing. Each problem includes a detailed step-by-step solution, making GSM8K a widely used and reliable benchmark for evaluating mathematical reasoning in language models.
- HumanEval [4] is a dataset of 164 hand-written Python programming problems paired with unit tests. Each prompt requires generating a function that satisfies the specified behavior, and functional correctness is evaluated by checking whether the model-generated code passes all provided test cases. This benchmark is widely used to assess program synthesis and code generation capabilities.

B. Model Architecture

Our method is tailored to Transformer-based architectures, where each layer, or Transformer block, consists of a multi-head attention submodule and a multilayer perceptron (MLP). This architecture serves as the foundation of many modern deep learning models, including ViT, Stable Diffusion models, and LLMs. Because our method closely interacts with the structure of both the attention and MLP submodules, we outline their roles and computations within a typical Transformer block.

MLP. The MLP submodule applies nonlinear transformations to each position's feature vector, often scaling dimensions in the hidden layer. Given a feature vector $X \in \mathbb{R}^d$ at a single spatial or temporal position, a standard two-layer MLP computes

$$\mathrm{MLP}(X) = W_2\sigma(W_1X + b_1) + b_2,$$

where $W_1 \in \mathbb{R}^{h\times d}$, $W_2 \in \mathbb{R}^{d\times h}$ are the learned weight matrices, b_1, b_2 are the corresponding biases, and $\sigma(\cdot)$ denotes a nonlinear activation such as GELU. Because the computation is applied independently to each token, the MLP enriches the representation capacity of the Transformer block without introducing interactions across positions.

Structure of Multi-head Attention Block. Consider a multi-head attention module with h heads, each operating on vectors of dimensionality d_k. Let the input sequence be $x \in$

Table 6. Dataset descriptions used in our image classification experiments.

Attribute	SUN397	Cars	RESISC45	EuroSAT	SVHN	GTSRB	MNIST	DTD
Domain	Scene recognition	Fine-grained car models	Remote sensing scenes	Satellite imagery	Digits in natural scenes	Traffic signs	Handwritten digits	Texture recognition
#Classes	397	196	45	10	10	43	10	47
Train Images	76,128	8,144	24,300	21,600	73,257	39,209	60,000	3,760
Test Images	32,626	8,041	7,200	5,400	26,032	12,630	10,000	1,880
Resolution	256×256	224×224	256×256	64×64	32×32	32×32	28×28	224×224

$\mathbb{R}^{seq\times d_{model}}$. A learned linear projection maps this sequence into combined query, key, and value matrices, $Q, K, V \in \mathbb{R}^{seq\times(h\times d_k)}$. These matrices are then partitioned along the feature dimension into h groups:

$$Q \to \{Q_1, \ldots, Q_h\},$$
$$K \to \{K_1, \ldots, K_h\},$$
$$V \to \{V_1, \ldots, V_h\},$$

where each $Q_i, K_i, V_i \in \mathbb{R}^{seq\times d_k}$ serves as the input to the i-th attention head, which computes:

$$\text{Attn}(Q_i, K_i, V_i) = \text{Softmax}(\frac{Q_i K_i^{\top}}{\sqrt{d_k}})V_i.$$

After all heads produce their outputs, the resulting vectors are concatenated and transformed through an output projection. Equivalently, the final attention representation can be written as

$$\text{Attention}(Q, K, V) = \sum_{i=1}^{h} \text{Attn}(Q_i, K_i, V_i)W_{\text{out}}^{(i)}.$$

where $W_{\text{out}}^{(i)} \in \mathbb{R}^{d_k\times d_{model}}$ denotes the corresponding block of the output projection matrix.

C. Time Cost of Mask Evaluation

We analyze the time complexity of the mask evaluation process in Stage II. As reported in Tables 7 and 8, the evaluation time (measured in minutes) scales with the dataset size, reflecting the increased computational effort required to process task-relevant samples. This overhead stems from MergeGuard's selective preservation mechanism, which identifies and protects task-critical weights to prevent accuracy degradation during subsequent weight suppression. We consider this computational cost acceptable, as it provides a necessary security-utility trade-off to effectively mitigate risks from potential free-riders.

Table 7. Time cost (mins) of ViT-L-14 across various datasets.

SUN397	Cars	RESISC45	EuroSAT
557.28	278.64	130.75	174.02
SVHN	GTSRB	MNIST	DTD
371.52	128.71	108.00	51.60

Table 8. Time cost (mins) of different LLMs across various tasks.

Task	**Llama2**	**Gemma2**	**Mistral**	**Avg.**
Instruction (AlpacaEval)	641.20	512.12	358.41	503.91
Math (GSM8K)	606.50	451.58	293.08	450.39
Code (HumanEval)	404.30	330.78	312.65	349.24

D. Additional Results

D.1. Fine-grained Analysis

Table 9 reports per-model results in the 1 + 1 setting. MergeGuard ensures low accuracy on T_{def}. It does not aim to preserve performance on T_{fr}, as this capability belongs to the free-rider's original model and is outside the defender's protection objective. In the first column (T_{fr} = Cars), the accuracies on all T_{def} tasks drop to an unusable level.

Table 9. T_{def}'s and T_{fr}'s accuracy in $\hat{\theta}_{\text{merge}}$ under WA.

Def. \ Fr.	Cars	RESISC45	EuroSAT	SVHN	GTSRB	MNIST	DTD	SUN397
Cars	–	.005/.034	.004/.120	.004/.109	.004/.021	.005/.103	.006/.026	.005/.004
RESISC45	.030/.006	–	.024/.108	.029/.078	.025/.008	.035/.103	.028/.021	.026/.003
EuroSAT	.585/.005	.641/.087	–	.617/.084	.604/.063	.580/.097	.572/.048	.636/.008
SVHN	.078/.005	.078/.025	.078/.111	–	.078/.021	.078/.103	.078/.022	.078/.004
GTSRB	.044/.005	.049/.034	.050/.070	.045/.082	–	.039/.098	.097/.029	.040/.004
MNIST	.165/.005	.157/.037	.166/.111	.163/.198	.161/.021	–	.173/.024	.192/.003
DTD	.544/.899	.553/.963	.526/.995	.542/.972	.531/.954	.545/.996	–	.537/.790
SUN397	.612/.877	.607/.945	.612/.785	.614/.768	.615/.775	.615/.796	.611/.577	–

D.2. Visualization Results on SD1.5

Figure 5 presents additional results of MergeGuard on image generation tasks. We use Animate as the defender's task (T_{def}) and "The Simpsons" as the free-rider's task (T_{fr}). The first two columns show the images generated by θ_{def} and θ_{fr}, respectively. When tested with the prompts C_0 and C_1 designed for Animate, the free-rider's model θ_{fr} is also able to reproduce the character, indicating that it has captured the semantic style of Animate. The merged model θ_{merge}, obtained through standard parameter fusion, inherits this ability and successfully generates Animate-style images, which demonstrates a clear instance of intellectual property leakage. In contrast, MergeGuard preserves the performance of $\hat{\theta}_{\text{def}}$ while effectively degrading the quality of $\hat{\theta}_{\text{merge}}$, preventing the free-rider from reproducing Animate-style content.

D.3. Code Results

We further report the results on code generation using HumanEval. MergeGuard effectively suppresses the perfor-

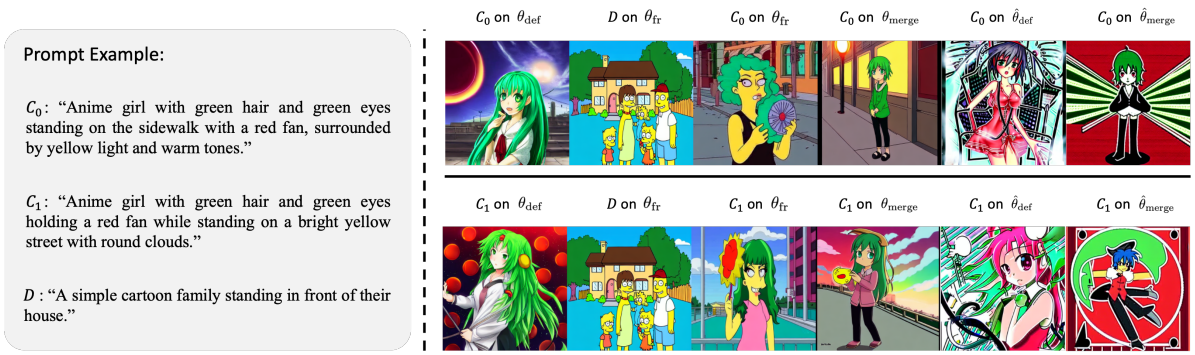


Figure 5. Visualization of images generated using SD1.5. The left panel displays an example prompt, and the right panel shows the corresponding outputs from each model.

mance of the merged model, often reducing it by more than half. We also observe that many generated programs, even when producing a passing result, contain substantial redundant code, unnecessary library imports, or code that leads to infinite loops. Figure 6 provides an example in which the generated program repeatedly produces the same invalid output. The results validate that MergeGuard reliably degrades merged model performance and prevents the generation of valid programs.

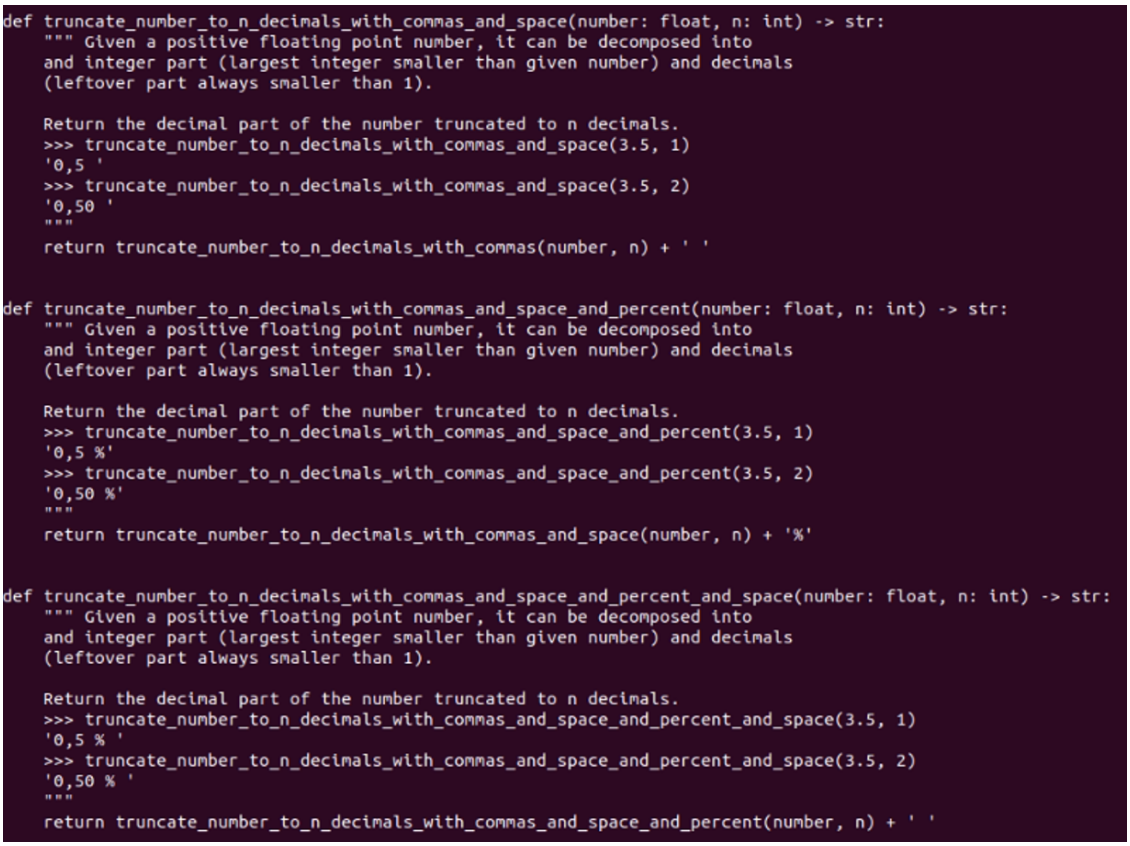
E. Limitations and Future Directions

E.1. Extensions to SVD-based Merging

Recently, KnOTS [32] proposed a model merging approach based on Singular Value Decomposition (SVD) to better preserve shared knowledge across models. Since MergeGuard perturbs the information geometry and directional alignment of task vectors, it would be interesting to evaluate its robustness under SVD-based subspace alignment methods. We conjecture that our density-aware perturbations may inherently disrupt the identification of a stable shared subspace, as they disperse task-relevant features that SVD-based methods typically aim to concentrate. We leave a systematic investigation of this intersection for future work.

E.2. Concurrent Work: LMC-based Defenses

We acknowledge a concurrent study, *MergeBarrier* [23], which also investigates the security of open-source LLMs against unauthorized model merging. While *MergeBarrier* focuses on disrupting the Linear Mode Connectivity (LMC) to eliminate low-loss merging paths in the loss landscape, MergeGuard operates on a distinct principle: it perturbs the information geometry of task vectors to promote a more isotropic sensitivity distribution. These two approaches represent complementary defensive philosophies, targeting optimization trajectories versus parameter-space alignment; thus, an empirical comparison between them remains an insightful direction for future research.



```
def truncate_number_to_n_decimals_with_commas_and_space(number: float, n: int) -> str:
    """ Given a positive floating point number, it can be decomposed into
    and integer part (largest integer smaller than given number) and decimals
    (leftover part always smaller than 1).

    Return the decimal part of the number truncated to n decimals.
    >>> truncate_number_to_n_decimals_with_commas_and_space(3.5, 1)
    '0,5 '
    >>> truncate_number_to_n_decimals_with_commas_and_space(3.5, 2)
    '0,50 '
    """
    return truncate_number_to_n_decimals_with_commas(number, n) + ' '


def truncate_number_to_n_decimals_with_commas_and_space_and_percent(number: float, n: int) -> str:
    """ Given a positive floating point number, it can be decomposed into
    and integer part (largest integer smaller than given number) and decimals
    (leftover part always smaller than 1).

    Return the decimal part of the number truncated to n decimals.
    >>> truncate_number_to_n_decimals_with_commas_and_space_and_percent(3.5, 1)
    '0,5 %'
    >>> truncate_number_to_n_decimals_with_commas_and_space_and_percent(3.5, 2)
    '0,50 %'
    """
    return truncate_number_to_n_decimals_with_commas_and_space(number, n) + '%'


def truncate_number_to_n_decimals_with_commas_and_space_and_percent_and_space(number: float, n: int) -> str:
    """ Given a positive floating point number, it can be decomposed into
    and integer part (largest integer smaller than given number) and decimals
    (leftover part always smaller than 1).

    Return the decimal part of the number truncated to n decimals.
    >>> truncate_number_to_n_decimals_with_commas_and_space_and_percent_and_space(3.5, 1)
    '0,5 % '
    >>> truncate_number_to_n_decimals_with_commas_and_space_and_percent_and_space(3.5, 2)
    '0,50 % '
    """
    return truncate_number_to_n_decimals_with_commas_and_space_and_percent(number, n) + ' '
```

Figure 6. Redundant or looping code generated by the merged model protected with MergeGuard.