Towards Scalable and Robust Model Versioning

Wenxin Ding, Arjun Nitin Bhagoji, Ben Y. Zhao, Haitao Zheng
Department of Computer Science, University of Chicago
{*wenxind, abhagoji, ravenben, htzheng*}*@cs.uchicago.edu*

***Abstract*—As the deployment of deep learning models continues to expand across industries, the threat of malicious incursions aimed at gaining access to these deployed models is on the rise. Should an attacker gain access to a deployed model, whether through server breaches, insider attacks, or model inversion techniques, they can then construct white-box adversarial attacks to manipulate the model's classification outcomes, thereby posing significant risks to organizations that rely on these models for critical tasks. Model owners need mechanisms to protect themselves against such losses without the necessity of acquiring fresh training data - a process that typically demands substantial investments in time and capital.**

In this paper, we explore the feasibility of generating multiple versions of a model that possess different attack properties, without acquiring new training data or changing model architecture. The model owner can deploy one version at a time and replace a leaked version immediately with a new version. The newly deployed model version can resist adversarial attacks generated leveraging white-box access to one or all previously leaked versions. We show theoretically that this can be accomplished by incorporating parameterized *hidden distributions* into the model training data, forcing the model to learn task-irrelevant features uniquely defined by the chosen data. Additionally, optimal choices of hidden distributions can produce a sequence of model versions capable of resisting compound transferability attacks over time. Leveraging our analytical insights, we design and implement a practical model versioning method for DNN classifiers, which leads to significant robustness improvements over existing methods. We believe our work presents a promising direction for safeguarding DNN services beyond their initial deployment.

I. Introduction

As deep learning models become increasingly prevalent across various industries, the risk of malicious attacks attempting to breach access to these deployed models is growing. When an attacker obtains access to a deployed model, via many methods including server breaches, insider attacks, or model inversion attacks, they can craft white-box adversarial attacks to manipulate the model's classification results. As the classification results lose their reliability, the deployed ML service becomes obsolete and needs to be replaced or removed. For example, in the medical field, image classification faces real-world attacks that aid insurance fraud. Some physicians engage in overprescribing medications and misdiagnosing conditions, prompting insurance companies to employ image classification for diagnosis verification. Adversarial attacks against these models are well-documented and raise significant concerns [1], [2]. Another example is classifiers for image content moderation on discussion boards and online platforms, which are frequently attacked by bad actors seeking to bypass content moderation with prohibited content [3], [4].

Unfortunately, replacing a breached model is a challenging task. This is because, building powerful deep learning models often involves acquiring and curating high quality training datasets, a process that incurs significant investment in time and capital, and can be difficult or impractical to repeat. For example, a model identifying diseases like rare skin lesions may take years to collect the training data, involving curation by specialists and de-identification to comply with privacy regulations [5]. Similarly, training data for content moderation models is often extremely sensitive and difficult to procure, e.g. images of extreme violence or abuse of minors. Obtaining and labeling these challenging images require manual inspection and careful supervision.

These observations motivate us to investigate how model owners can protect themselves against model losses *without* the necessity of acquiring new training data. Ideally, they would like to train multiple versions of the target model from the same training set, deploy one version at a time and replace a leaked version with a new one. At the time of its deployment, each new version i of the model should resist adversarial attacks as if it were the *sole* model version that had been deployed. That is, when attempting to attack model i, an attacker would gain minimal or no advantage by obtaining white-box access to any or all previous model versions $\{j \mid j < i\}$. The longer the model sequence supports this *sequential* (or *compound*) robustness, the stronger the protection against repeated model leakages. We refer to this problem as *scalable and robust model versioning*.

Unfortunately, solving this problem is difficult because of two significant challenges. First is the well known phenomenon of *attack transferability*: adversarial attacks generated on one model often succeed on similar models trained on the same task, even when they use different architectures [6], [7]. To date, few if any techniques provide the ability to generate multiple (> 3) models on the same task with low attack transferability between them, while maintaining normal classification accuracy. Second, our problem must address a novel but natural evolution of the white-box attack, *a sequential, compound transferability attack*. As each leaked model is retired, an attacker with no white-box access to the new version i, can still utilize their white-box access to prior versions (1 to $i-1$) to orchestrate a strong attack against version i. This new requirement and the need to sequentially deploy/replace models make our problem distinct from existing works (*e.g.,* constructing robust ensembles [8], [9], [10]), creating a new, open challenge.

To tackle these challenges, our work introduces a principled

approach for generating a sequence of robust model versions from a *single* training dataset using a *single* model architecture. Our method automatically curates and incorporates *parameterized hidden data* into the model training data, forcing the model to learn task-irrelevant features that are distinctly defined by the selected hidden data. More specifically, given an original task, we curate synthetic data that are drawn from some hidden distributions irrelevant to the task, and augment the task's original training data with the new hidden data. Thus, the training data of a class now includes both its original training data and new data drawn from a chosen hidden distribution. Using the combined data, we train a model version from scratch. And by varying the choices of hidden distributions, we can produce different model versions.

Our hypothesis is that the above process, if well-designed, can produce a diverse set of model versions. These model versions would not only effectively accomplish the primary task but also exhibit distinct attack properties, because they are trained on varying hidden distributions that naturally introduce diverse non-robust features into each model. Leveraging this variability, the model trainer can carefully select, organize and continuously deploy a *sequence* of model versions to withstand compound transferability attacks over time.

We validate this hypothesis by first performing theoretical analysis in a simplified setting. Our formal proof demonstrates that optimizing the selection of hidden distributions significantly reduces the transferability of compound attacks against subsequent model versions. Additionally, effective hidden distributions are characterized by a single point in the feature space, facilitating the parameterization process for efficient optimization. Together, these analytical findings demonstrate the importance and viability of optimizing the choices of hidden distributions when constructing the model sequence. This stands in contrast to an earlier work [11] that employed randomly selected hidden data without any optimization.

Building on our analytical results, we develop a practical, greedy search-based algorithm for constructing hidden distribution-based model versions for deep neural network (DNN) classifiers. We implement and evaluate our method using three image classification tasks that encompass different image categories (objects, medical images, and faces) and varying number of classes (7 to 1283). Our design significantly outperforms alternative methods for model versioning, including [11] and those designed to produce "orthogonal" models.

Summary of Contributions. To the best of our knowledge, our work is the first to provide a principled investigation of scalable and robust model versioning – a crucial task for safeguarding DNN services *beyond* their initial deployment. Our work makes three key contributions:

- We formally define the process of hidden distribution-based training as a solution for model versioning (§III and §IV);
- We analytically demonstrate the critical impact of hidden distributions on model versioning and develop a practical algorithm for systematically selecting hidden distributions to construct robust model versions (§V and §VI).
- We evaluate our design by building a sequence of model versions for three image classification tasks. These models achieve significantly higher robustness against attacks compared to existing methods (§VII).

Finally, we discuss the limitations of our work and potential future directions. We hope that our work can inspire further research efforts in this critical yet underexplored area.

II. Background and Related Work

To provide context, we briefly describe transferability of adversarial example attacks, existing methods to restrict transferability between models, and those to produce model variants.

Transferability of Adversarial Examples. It is well-known that adversarial examples generated for one model can produce misclassification on other similar models [12], [13], a phenomenon known as attack *transferability*. It is widely used by attacks where attackers have no access to the internals of the target model. One can also increase attack transferability by modifying attack methodology [7], [14], gaining limited query access to the target model [15] and leveraging learned model features [16], [17]. When exploring conditions that produce high attack transferability, Demontis et al. showed that the alignment of input gradients between models and the reduction of variability of the loss surface are critical for high transferability [6]. Others found that transferability between models correlates strongly with their semantic layer similarity [18] or feature similarity [19].

Reducing (Pairwise) Attack Transferability. One can apply adversarial training (*e.g.,* [20]) to reduce a model's vulnerability to attacks, which may help reduce attack transferability from other models. However, doing so faces high training cost and reduced task accuracy. Recent works (*e.g.,* DVERGE [8] and TRS [9]) proposed to train an ensemble of diverse models with low pairwise attack transferability, to resist adversarial attacks against the entire ensemble. These ensemble methods, like adversarial training, require iterative optimization to simultaneously train the full ensemble at once. The resulting optimization is computationally expensive, especially for ensembles with more than 3 models. For example, DVERGE [8] reported a high pairwise attack transferability of 79% when training an ensemble of size 5.

Our work considers a very different problem. Rather than training a set of models that operate as an ensemble to classify inputs and resist attacks (where attackers know either all or none of the models), we seek to produce a sequence of model versions and deploy them one by one in a sequential order, each triggered by a model leakage event. Each model version i operates by itself to classify inputs and faces a new type of black-box attacks constructed using all prior versions. That is, while an attacker has no access to version i, it can leverage white-box access to all prior versions (1 to $i-1$) to build a powerful "group-based" attack against version i. Previous studies have not considered this novel form of attack, and mitigating it cannot be achieved solely by reducing pairwise attack transferability.

Model Versioning. In practice, deployed models often evolve over time to adapt to changes in data or to incorporate new training methods [21], [22], [23]. However, none of these methods explicitly consider how to update a model after it is leaked to attackers. The only known work on this topic is [11], which randomly selects hidden distributions to produce model variants. However, [11] does not consider reducing attack transferability, instead focusing on creating a separate input filter to identify potential adversarial examples at run-time. Our work is inspired by [11], but differs in two key aspects. First, we study whether and how hidden distributions can be optimized to minimize attack transferability against subsequent model versions. We are the first to establish an analytical framework to formally illustrate the critical impact and viable path for optimizing the selection of hidden distributions. Notably, our findings stand in contrast to [11]'s random selection decision. Second, we develop a principled method for selecting hidden distributions and organizing a sequence of model versions for DNN classifiers. Experiments in §VII show that our method significantly outperforms [11] both with and without the input filter. Therefore, our study makes a tangible step forward in tackling the challenging problem of model versioning.

Detecting Model/Server Breaches. One assumption made by our work is that the model owner/deployer can detect that the current model version has been breached, so that our work can focus on the problem of finding replacement models. This assumption is not unrealistic. Many years of research in the computer security community has focused on intrusion detection and on detecting when a server has been breached (and when specific files are accessed), *e.g.,* intrusion detection systems, APT detection/analysis, and OS-level secure access logs. In addition, classification systems rarely stand alone, and often lead to downstream errors and negative outcomes. Any attacker that uses the breached model to trigger negative outcomes downstream can be detected, with the downstream errors used to trace back to the compromised model, *e.g.,* post-attack forensic systems [24].

III. Problem and Threat Model

In this section, we describe the problem of model versioning and its threat model. We focus our discussion on the adversary's capabilities once they have breached the deployed model and the objectives of the model owner in responding to and recovering from such a breach. We start from the simple scenario of single (or one-time) model breach and then move to the broader context of multiple (or repeated) model breaches. With these considerations in mind, we also examine existing solutions and their limitations.

For the rest of the paper, we consider the standard multi-class classification model $\mathcal{M} : \mathcal{X} \subseteq \mathbb{R}^n \rightarrow \mathcal{L}$. The classification task is the main task where training data on the main task is denoted as D_{train} and test data on the main task is D_{test}.

A. One-time Breach

We first consider the simple case where a deployed model $\mathcal{M}_1$ is breached by an attacker. Here our work assumes that the model owner/deployer has detected the breach. To recover from such loss and continue the model service, the model owner replaces $\mathcal{M}_1$ with a new version $\mathcal{M}_2$.

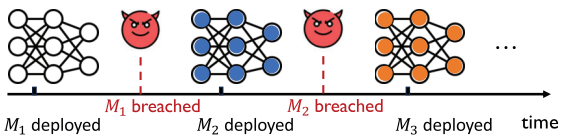

Fig. 1: *Model versioning: after the deployed model $\mathcal{M}_i$ is breached by the attacker, the model owner replaces it with a new version $\mathcal{M}_{i+1}$ to ensure uninterrupted service.*

Adversary Capabilities. In this case, the parameters of $\mathcal{M}_1$ are, at some point, leaked or inferred by an attacker (see Figure 1). While several threats to data privacy and security could arise from such a leak, we focus on the particularly pernicious one of *white-box adversarial examples*. Here we assume the attacker is capable of constructing targeted adversarial examples [1] with the knowledge of $\mathcal{M}_1$'s architecture and weights. Specifically, for a desired class $l_t \in \mathcal{L}$, the attacker can generate $\tilde{x} = x + \delta_x$ such that

$$\mathcal{M}_1(\tilde{x}) = l_t \neq \mathcal{M}_1(x)$$

with $d(x, \tilde{x}) < \epsilon$, where l_t is not the correct class of x and $d(\cdot, \cdot)$ is a distance function (usually a p-norm). Finally, since the attacker has no knowledge of the replacement model $\mathcal{M}_2$, they will generate white-box adversarial examples from $\mathcal{M}_1$ and leverage transferability to attack $\mathcal{M}_2$.

Model Owner's Goal. To recover from the breach, the model owner deploys a new model $\mathcal{M}_2$ to replace $\mathcal{M}_1$, where

- $\mathcal{M}_2$ maintains high performance on the main task assessed by the test data D_{test};
- $\mathcal{M}_2$ is robust against white-box adversarial attacks computed based on $\mathcal{M}_1$.

Robustness Measure. Here the threat to the model owner is the leakage of $\mathcal{M}_1$, which the attacker can leverage to craft white-box adversarial examples. Thus, we measure the robustness of the model replacement by the ***directional attack transferability*** from $\mathcal{M}_1$ to $\mathcal{M}_2$:

$$\mathcal{AT}(\mathcal{M}_1 \rightarrow \mathcal{M}_2).$$

This metric is directional, *i.e.,* the sequential order of the model deployment matters.

B. Multiple Breaches

In practice, model breaches can happen multiple times – after replacing $\mathcal{M}_1$ with $\mathcal{M}_2$, the deployed model $\mathcal{M}_2$ can be breached, at some point, forcing the model owner to replace it with $\mathcal{M}_3$, etc. (see Figure 1). And more importantly, after breaching the deployed model multiple times, the attacker has obtained more knowledge from the leaked models, and can launch stronger adversarial attacks. Therefore, the model

[1] We focus on targeted adversarial examples for ease of exposition. All results hold for untargeted adversarial examples as well.

owner requires a stronger notion of robustness than the one-time breach case, and a scalable method to extend the sequence of model deployment beyond the simple case of $\mathcal{M}_1$ and $\mathcal{M}_2$.

In the following discussion, we consider a sequence of models $\mathcal{M}_1, ..., \mathcal{M}_i$ that has been deployed and breached, with $\mathcal{M}_i$ ($i \geq 2$) being the most recent model that is breached. Now the model owner needs to replace $\mathcal{M}_i$ with $\mathcal{M}_{i+1}$.

Adversary Capabilities. We consider a powerful attacker that has white-box access to all the breached models. With no knowledge of the replacement model $\mathcal{M}_{i+1}$, the attacker leverages transferability to attack $\mathcal{M}_{i+1}$. The attacker is capable of constructing targeted adversarial examples using knowledge of $\mathcal{M}_1, ..., \mathcal{M}_i$, *i.e.,* generating $\tilde{x} = x + \delta_x$ such that, for at least one j in $\{1, ..., i\}$,

$$\mathcal{M}_j(\tilde{x}) = l_t \neq \mathcal{M}_j(x), \quad j \in \{1, ..., i\}$$

where $d(x, \tilde{x}) < \epsilon$. We also consider "highly cautious" attackers who only apply $\tilde{x}$ to attack $\mathcal{M}_{i+1}$ if $\tilde{x}$ succeeds on *all* prior versions ($\mathcal{M}_1, ..., \mathcal{M}_i$).

Model Owner's Goal. Besides having high performance and robust against any model that is previously breached, the model owner also needs

- a scalable approach to find the replacement model $\mathcal{M}_{i+1}$;
- $\mathcal{M}_{i+1}$ is robust against the strong attacker that has knowledge of all previous models $\mathcal{M}_1, ..., \mathcal{M}_i$.

Robustness Measure. We define a new notion of robustness to characterize the strong attacker that has white-box access to all previous model versions. This is measured by the attack transferability under attacks generated using an ensemble of $\mathcal{M}_1, ..., \mathcal{M}_i$, namely the ***compound attack transferability***:

$$\mathcal{AT}(\{\mathcal{M}_j\}_{j=1}^{i} \rightarrow \mathcal{M}_{i+1}).$$

Correspondingly, the model owner needs a model versioning technique that produces a sequence of models $\mathcal{M}_1, ..., \mathcal{M}_i, \mathcal{M}_{i+1}$ to minimize the compound attack transferability at each model version.

C. Design Challenges and Initial Solutions

We identify two unique challenges facing the design of scalable and robust model versioning.

- **Versioning uncertainty** – The model owner cannot foresee the exact number of model versions required beforehand, nor can they anticipate the timing of a potential breach of the deployed model. Yet the model owner must promptly identify a replacement model once a breach is detected.
- **Continuous expansion of attacker knowledge** – With each occurrence of model leakage, the attacker gains more information about the models, making it hard to maintain low compound attack transferability while simultaneously ensuring high performance on the primary task. On the other hand, the model owner's lack of access to new training data hinders their ability to gather additional knowledge for constructing new models and defending against attacks.

Initial Solutions. Drawing upon existing literature, we identify several initial solutions to perform model versioning. We also explore their limitations, which motivate our work.

- **Varying model initialization** – One might consider generating model variants by varying the model initialization before training. However, since these models are trained using the same dataset, they tend to converge with a high likelihood and share high attack transferabilities. Later, we validate this projection both theoretically and empirically.
- **Varying training batch order** – Similarly, one can produce model versions by injecting "randomness" to the order of training data. Yet since the training data remains unchanged, these models also tend to converge with a high probability.
- **Varying model architecture** – One can employ distinct model architectures across model versions. Yet this method faces two critical limitations. First, the available model architectures for a given task are often limited, especially for high-performance models tailored to complex tasks. Second, the mere use of different architectures does not assure a reduction in transferability in a principled manner.
- **Generating model ensemble** – Ensemble-based model optimization a(e.g., TRS [9]) trains a fixed ensemble of models before deployment, aiming to improve ensemble robustness by reducing pairwise attack transferability among models in the ensemble. This approach is ill-suited for our specific problem due to the two distinct challenges outlined earlier: version uncertainty and continuous expansion of attack knowledge. The ensemble methods require knowledge of the exact number of model versions in advance, and once the ensemble is trained, there is no method for generating additional model versions. Furthermore, ensemble methods tend to be computationally expensive and primarily focus on reducing pairwise attack transferability among models. We confirm these via empirical experiments later on.
- **Detecting attack inputs** – As discussed in §II, recent work [11] develops a method to generate model variants by adding randomly generated images to model training data. Rather than reducing attack transferability, [11] focuses on creating a separate input filter to detect transferability-based adversarial examples at run-time. Consequently, as more model versions are leaked, the attack transferability remains high (*e.g.,* 90%) while the input filter loses its effectiveness. Our experiments also confirm this observation.

The above discussion shows that none of the existing solutions effectively address the challenges facing scalable and robust model versioning. In the following section, we introduce our proposed solution that significantly improves over existing solutions by utilizing optimized hidden training to continuously generate robust model versions over time.

IV. Proposed Solution: Optimized Hidden Training

We propose a new approach to model versioning, utilizing the novel concept of "optimized hidden training" to continuously produce model versions that exhibit resilience against compound transferability attacks. Next, we present the

intuition behind our solution, followed by the formal definition and optimization process for configuring hidden training.

A. Design Intuition

Augmenting Model Training Using "Hidden Data." We propose to create distinct model versions by introducing carefully planned variability into the model training data. At a broad level, our solution aligns with the concept of "training data augmentation." However, what sets our approach apart is the automatic self-curation of additional training data for each model version, all without the necessity of acquiring new training data. In the following, we refer to this supplemental training data, which is unique to each model version, as "hidden data" because it is constructed internally with a focus on maintaining secrecy.

Curating Hidden Data from a Single Feature Point. Another distinctive contribution of our work is the novel concept of curating hidden data from a *single* feature point, which represents a parameterized, hidden distribution that is irrelevant to the classification task. By varying the choice of this feature point, we seek to naturally introduce variability to the loss surface and the non-robust features learned by the models. Later in §V, we analytically validate this design decision.

Specifically, we first establish the task feature space by training an original model $\mathcal{M}_{ori}$ solely on the task training data D_{train}. We denote the task feature space $\mathcal{F}_{ori}$. Next, we select a solitary feature point h_i within $\mathcal{F}_{ori}$ for the i^{th} model version, and then create the associated data, which possesses a feature corresponding to h_i concerning $\mathcal{M}_{ori}$. Given that each model version i has its own hidden data aligned with its unique h_i, we can apply hidden training to generate multiple model versions that excel at the same task while exhibiting distinct attack characteristics.

The question that naturally arises is why we opt for using a single feature point to construct the hidden data. We argue that this approach offers the following two significant advantages.

- **Adding unique "distortions" to decision boundary** – Incorporating hidden data into the training process is expected to have an impact on the model's decision boundary within the feature space. Our strategy of concentrating all efforts on a single feature point is designed to efficiently "adjust" the decision boundary in the direction defined by that particular feature point. In contrast, if we were to generate hidden data from multiple feature points (similar to the conventional augmentation method), their effects on the decision boundary might counterbalance each other. This would not only lead to more subtle changes in the decision boundary but also diminish the uniqueness of those changes. In Figure 2, we present several illustrative examples where the single-point-based method introduces version-specific modifications to the decision boundaries, unlike the multi-point-based method.
- **Maintaining original task performance** – Including non-task data during training may increase the training complexity. When adding multiple feature points to a class, it is likely that the added training data disturbs the model performance on this class. Therefore, adding just a single feature point helps maintain high performance of the model without introducing high complexity to model training, making the training process as efficient as the original model.

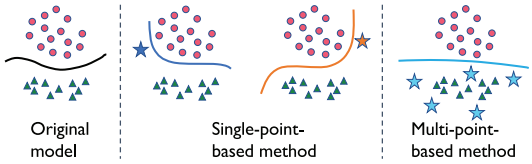

Fig. 2: *Impact of hidden training on models. By generating hidden data from a single feature point, one can introduce unique modifications to model decision boundaries. But when generating hidden data from multiple feature points, both the extent and uniqueness of the modification decrease.*

B. Formal Definition

We now formally define the formulation of hidden data and the process of hidden training to create new model versions.

Definition IV.1. (**Parameterized Hidden Data**) Let h be a point in $\mathcal{F}_{ori}$, where $\mathcal{F}_{ori}$ represents the feature space of the model trained solely on the task training data D_{train}. For $x \in \mathbb{R}^n$, we abuse the notation to let $\mathcal{F}_{ori}(x)$ denote the corresponding feature value of x in the feature space $\mathcal{F}_{ori}$. Let $\mathcal{X}_h$ be data in the input space $\mathbb{R}^n$ such that $\forall x \in \mathcal{X}_h$, $d(\mathcal{F}_{ori}(x), h) < \epsilon_h$, where $d(\cdot,\cdot)$ is a distance metric for $\mathcal{F}_{ori}$. To protect a specific class l_t against adversarial examples, we label all $x \in \mathcal{X}_h$ as l_t, producing the following *hidden data* for this class:

$$D_{\text{hidden}} = \{(x, l_t) \mid x \in \mathcal{X}_h\}.$$

Definition IV.2. (**Hidden Training**) Given the hidden dataset D_{hidden}, the hidden training process is to train a new model from scratch utilizing the merged training data, $D_{\text{train}} \cup D_{\text{hidden}}$. This newly trained model is denoted as $\mathcal{M}_{D_{\text{train}} \cup D_{\text{hidden}}}$.

In practice, one can implement the hidden training process via three consecutive steps:

1. Using Algorithm 1 to select a set of feature points $\{h\}$ from the feature space $\mathcal{F}_{ori}$, one for each class to be protected. To maintain high performance on the original task, they should avoid overlapping with the feature areas of D_{train}.

2. Using Algorithm 2 to construct the hidden data D_{hidden} based on $\{h\}$ to ensure that the feature values of D_{hidden} closely match each of h. Here the data curation employs a technique reminiscent of computing adversarial examples, wherein an input data is perturbed to adjust its feature value towards the target feature point h.

3. Train a new model from scratch using $D_{\text{train}} \cup D_{\text{hidden}}$.

Following this process, model owners can generate multiple model versions without acquiring new training data. By choosing N different sets of points in the task feature space, *i.e.,* $\{h\}_1, ..., \{h\}_N$, they can produce N different sets of hidden training data $D_{\text{hidden}\,1}, ..., D_{\text{hidden}\,N}$, and correspondingly, N different variants of the original model:

$\mathcal{M}_{D_{\text{train}} \cup D_{\text{hidden}\,1}}, ..., \mathcal{M}_{D_{\text{train}} \cup D_{\text{hidden}\,N}}$. For ease of notation, we hereby refer to those models as $\mathcal{M}_1$, ..., $\mathcal{M}_N$. This model versioning process is *efficient* and *scalable*, and there is no need to fix N a priori. Generating additional model versions beyond N is achieved by selecting new hidden feature points and repeating the hidden training process.

To our knowledge, we are the first to present a formal definition of the robust model versioning problem where h is the key parameter. None of the previous work, including [11], seek to understand impact of the choice of h on (compound) attack transferability against subsequent models.

C. Optimizing Hidden Training for Robustness

Using hidden training, model owners can generate a sequence of model variants capable of performing the original classification task while resisting transferability-based attacks. For a sequence of N model versions, the level of robustness is contingent on the selection of $h_1, ..., h_N$ used to curate hidden training data for each model version. In this work, we consider two methods for selecting hidden features: *random selection* and *greedy optimization*. In §V and §VI, we explore, both analytically and experimentally, their effectiveness in producing a robust sequence of model versions.

- **Random selection** – The simplest method is to randomly select a new feature point when generating a new model version [11]. One might assume that the randomness in $h_1, ..., h_N$ would naturally introduce diversity among the resulting model versions. However, we demonstrate through both analytical and empirical results that the randomness in feature point selection does not necessarily diminish attack transferability. This becomes particularly true as the value of N increases since the attacker accumulates more knowledge about the models with each successful model breach.
- **Greedy optimization** – To resist compound transferability attacks, the model owner can leverage their own access and understanding of previous (and leaked) models to strategically choose the hidden feature points for the subsequent replacement model. This selection is greedy as the model owner does not know how many additional model versions they need to generate a priori. To recover from the loss of model version i, we pre-train the next version $i+1$, based on all previously breached versions $\mathcal{M}_1, ..., \mathcal{M}_i$. We defer the in-depth discussion of the algorithm to §VI, as it relies on the insights derived from the analytical study of hidden training to be presented in §V.

V. Analytical Study of Hidden Training

In this section, we present an analytical case study on model versioning using parameterized hidden training. Our goal is to demonstrate the importance of carefully selecting hidden features when producing the sequence of robust model versions. Our analysis focuses on binary classification tasks utilizing linear Support Vector Machine (SVM) models. We examine how the choice of h impacts the model's decision boundary and explore different configurations of h to create a sequence of model versions that resists direct and compound transferability attacks over time. We also discuss ways to generalize our analysis to more complex settings.

A. Preliminaries

We consider a binary classification task with two classes over input space $\mathcal{X} \subseteq \mathbb{R}^2$. Let $\mathcal{X}_+$ and $\mathcal{X}_-$ denote the task data of class $+$ and class $-$, respectively. Let $D_{\text{train}+}$ and $D_{\text{train}-}$ denote the task training data of the two classes, respectively. The detailed configuration of these datasets are shown in Table I. The task here is to train linear SVM model versions to classify points from $\mathcal{X}$ to $\{+, -\}$, using the task training data $D_{\text{train}+}$, $D_{\text{train}-}$, and the chosen hidden data $\mathcal{X}_h$.

Input Space = Feature Space. We note that in the linear SVM setting, the input space and the feature space are identical, *i.e.,* $\{h\} = \mathcal{X}_h$. Thus, to streamline our discussion, we directly use the chosen hidden feature $\{h\}$ to represent $\mathcal{X}_h$. However, this statement does not hold in general, specifically for DNN models where the feature space is considerably different from the input space.

Hidden Training in SVM. An SVM model $\mathcal{M} : \mathbb{R}^2 \rightarrow \{+, -\}$ is a function that takes data from $\mathbb{R}^2$ as input and outputs the class label. We assume that the class to be protected, *i.e.,* the class that will be targeted by the attacker, is class $+$. Hidden training involves choosing hidden data h and the target class $+$, and adding it to the training data set.

Our analysis of hidden training starts from the following theorem and the definition of attackable region.

Theorem V.1. ***(Hidden Training Determines SVM).*** *When h is in $\{(x, y) \in \mathbb{R}^2 \mid |x| < c - 1, |y| \leq y_{lim}\}$, h determines the decision boundary of the trained SVM model.*

As defined by Table I, c is the x-axis center of the training data's feature cluster for $+$ class and y_{lim} bounds the y-axis of the feature space. The proof of Theorem V.1 can be found in Appendix A.

This theorem shows that, in the two-dimensional space, the optimal linear SVM classifier is the one that bisects the shortest connection between the convex hulls of the training data in the classes [25]. When h is not in $D_{\text{train}+}$, h determines the convex hull of $D_{\text{train}+} \cup D_{\text{hidden}}$. A properly chosen h can also determine the SVM classifier when it changes the shortest connection between the convex hulls. Thus, Theorem V.1 characterizes the region in $\mathbb{R}^2$ where the SVM is determined by the hidden data added to class $+$.

Attackable Region. Given a model version $\mathcal{M}_i$, we define its attackable region to identify the subspace of $\mathbb{R}^2$ where any adversarial attacks targeting class $+$ can appear. This region contains all possible attacks generated by the strongest adversary, one who operates without any limitations on perturbation budgets or computational resources. We use $\mathcal{AR}_i$ to denote the attackable region of $\mathcal{M}_i$.

Definition V.2. (**Attackable Region**) For an SVM model $\mathcal{M}_i$, we define its attackable region for the target class $+$ as

$$\mathcal{AR}_i = \{(x, y) \in \mathcal{X}_- \mid \mathcal{M}_i(x, y) = +\}.$$

Task data $(\mathcal{X}_+, \mathcal{X}_-)$	Data in $\mathcal{X}_+$ and in $\mathcal{X}_-$ are symmetrically distributed about the y-axis. For an input $(x, y) \in \mathcal{X}$, we have $(x, y) \in \mathcal{X}_+$ if $x \geq \delta$ or $-\delta < x < 0$ where $\delta > 0$. Otherwise, $(x, y) \in \mathcal{X}_-$. Moreover, we bound the space such that for all $(x, y) \in \mathcal{X}$, $\lvert y \rvert \leq y_{\text{lim}}$. Figure 3 illustrates the setup. In this setting, no SVM model achieves perfect accuracy on $\mathcal{X}$. However, there exists SVM models that can linearly separate our training data, defined as below.
Task training data $(D_{\text{train}+}, D_{\text{train}-})$	Data in $D_{\text{train}+}$ is uniformly distributed in a unit circle centered at $(c, 0)$ and $D_{\text{train}-}$ in a unit circle centered at $(-c, 0)$ where $c \gg \delta$. We use D_{train} to denote the clean training data where $D_{\text{train}} = D_{\text{train}+} \cup D_{\text{train}-}$. Therefore, $\mathcal{X}$ is not linearly separable but D_{train} is.

TABLE I: *Configuration of task data and training data.*

The attackable region of model $\mathcal{M}_i$ contains all input data that should be categorized as class $-$ but wrongly classified as class $+$ by the model. That is, any possible attack directed towards class $+$ that an attacker can construct using only $\mathcal{M}_i$ falls within $\mathcal{AR}_i$. Beyond $\mathcal{AR}_i$, the data is either correctly classified by the model or bears a ground truth label of $+$.

B. Impact of Hidden Data Choice: One-time Breach

Next, we study how the choice of hidden data h affects the robustness of model versions, starting from the case of one-time breach. For this simple case, the robustness is measured by the directional attack transferability from $\mathcal{M}_1$ to $\mathcal{M}_2$. Our analysis focuses on modeling the directional attack transferability using the area of the attackable region. For two models $\mathcal{M}_1$ and $\mathcal{M}_2$ trained with different hidden data, all transferable attacks exist in the intersection of $\mathcal{AR}_1$ and $\mathcal{AR}_2$.

Definition V.3. (**Directional Attack Transferability**) Let $\mathcal{S} : \mathbb{R}^2 \to \mathbb{R}$ be a function that computes area of a subspace in $\mathbb{R}^2$. The directional attack transferability from $\mathcal{M}_1$ to $\mathcal{M}_2$, denoted as $\mathcal{AT}(\mathcal{M}_1 \to \mathcal{M}_2)$, is computed as:

$$\mathcal{AT}(\mathcal{M}_1 \to \mathcal{M}_2) = \frac{\mathcal{S}(\mathcal{AR}_1 \cap \mathcal{AR}_2)}{\mathcal{S}(\mathcal{AR}_1)}$$

When the two models share no attackable region (*i.e.,* $\mathcal{AR}_1 \cap \mathcal{AR}_2 = \emptyset$), the directional attack transferability becomes zero. The following theorem defines a condition on the choice of h_1 and h_2 to meet such condition.

Theorem V.4. ***(Nullifying Directional Attack Transferability)*** *For two model versions $\mathcal{M}_1$ and $\mathcal{M}_2$ produced from hidden training, we have $\mathcal{AT}(\mathcal{M}_1 \to \mathcal{M}_2) = 0$ when decision boundaries of $\mathcal{M}_1$ and $\mathcal{M}_2$*
(1) have opposite signs of slope, and;
(2) intersect at (x_I, y_I) with $x_I > \delta$.

The proof of Theorem V.4 is in Appendix B. In the proof, we also illustrate how to find qualified h_1 and h_2 to achieve zero transferability. We first exclude any linear separator that is not achievable by adding hidden data $\{h, +\}$ to D_{train}. Next, given an achievable linear separator, we reconstruct the coordinates of h by finding the connection between the two convex hulls bisecting by the separator. After identifying an achievable linear separator with hidden data h_1 that intersects the x-axis

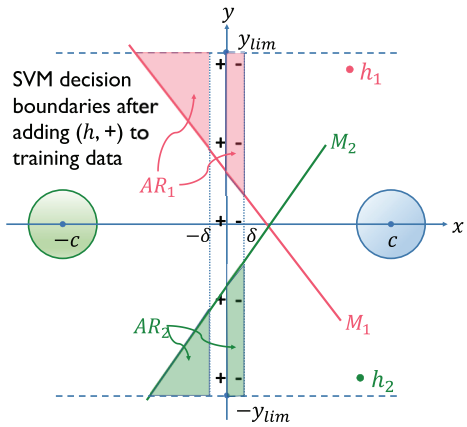

Fig. 3: *Illustration of SVM decision boundaries after hidden training using $(h_1, +)$ and $(h_2, +)$, respectively. The two resulting models $\mathcal{M}_1$ and $\mathcal{M}_2$ share zero attack transferability because their attackable regions do not overlap.*

at a point $> \delta$, we select h_2 as the coordinate symmetrical to h_1 along the x-axis, and use it to train $\mathcal{M}_2$. Figure 3 illustrates a specific scenario where the two models have no overlap in their attackable regions, resulting in zero attack transferability.

Additional Observations. Theorem V.4 leads to three additional findings, which later inform our approach in designing practical algorithms for selecting hidden features (§VI).

- **Random h values can rarely produce SVMs with low attack transferability** – Theorem V.4 demonstrates that h_1 and h_2 must be located on opposite sides of the x-axis to minimize directional attack transferability. Yet if h_1 and h_2 are randomly selected, there is a 50% chance that both fall on the same side of the x-axis, resulting in a high attack transferability between the two model versions.
- **Varying training parameters does not lower attack transferability** – In the SVM setting, adjusting model training parameters such as initialization and training batch order has no impact on the trained model. Therefore, in practice, such variations are unlikely to result in model versions with reduced attack transferability.
- **Separation between hidden data and task data** – Effective hidden data is often situated at a considerable distance from the training data in the feature space. Thus, we look for candidates of hidden features in outliers within each class.

C. Impact of Hidden Data Choice: Repeated Breaches

We now consider a sequence of N ($N > 2$) model versions $(\mathcal{M}_1, ..., \mathcal{M}_N)$. Within this sequence, after launching $\mathcal{M}_i$, an attacker, who has white-box access to $\mathcal{M}_1, ..., \mathcal{M}_{i-1}$ but not $\mathcal{M}_i$, can attack $\mathcal{M}_i$ using a combined knowledge of all previous model versions, *i.e.,* launching a compound transferability attack. We first establish a theoretical model for the compound attack transferability in the SVM case, utilizing the attackable region concept. All potential compound attacks are found in the union of $\mathcal{AR}_1, \ldots, \mathcal{AR}_{i-1}$, and the successful ones against $\mathcal{M}_i$ are situated in the intersection of $\mathcal{AR}_i$ and the aforementioned union.

Definition V.5. (**Compound Attack Transferability**) The compound attack transferability from $\mathcal{M}_1, ..., \mathcal{M}_{i-1}$ to $\mathcal{M}_i$, denoted as $\mathcal{AT}(\{\mathcal{M}_j\}_{j=1}^{i-1} \rightarrow \mathcal{M}_i)$, is computed as:

$$\mathcal{AT}(\{\mathcal{M}_j\}_{j=1}^{i-1} \rightarrow \mathcal{M}_i) = \frac{\mathcal{S}(\mathcal{AR}_i \cap (\cup_{j=1}^{i-1} \mathcal{AR}_j))}{\mathcal{S}(\cup_{j=1}^{i-1} \mathcal{AR}_j)}$$

Definition V.5 allows us to compute the attack transferability against any model version within a sequence of model versions. In this model, we characterize the strongest attacker that can utilize all attackable regions of previous leaked versions. That is, any attack generated from an ensemble of the previous versions is included by the union of attackable regions.

Next, we show that one can employ greedy search to construct a sequence of model versions, where the compound attack transferability within the sequence is upper bounded.

Theorem V.6. ***(Greedy Search Can Upper Bound Compound Transferability)*** *Using greedy search, we can construct a sequence of model versions $\mathcal{M}_1, ..., \mathcal{M}_N$ such that,*

$$\max_{i, i \leq N} \mathcal{AT}(\{\mathcal{M}_j\}_{j=1}^{i-1} \rightarrow \mathcal{M}_i) \leq \alpha_N$$

where α_N increases with N.

The detailed proof is in Appendix C. Here we briefly sketch the greedy search process. First, we construct $\mathcal{M}_1$ and $\mathcal{M}_2$, leveraging Theorem V.4 to identify h_1 and h_2 such that $\mathcal{AT}(\mathcal{M}_1 \rightarrow \mathcal{M}_2) = 0$, and their combined attackable region $\mathcal{AR}_1 \cup \mathcal{AR}_2$ is sufficiently large to minimize overlapping portion with subsequent model versions. In our design, $\cup_{j=1}^{i} \mathcal{AR}_j = \mathcal{AR}_1 \cup \mathcal{AR}_2$ for any $i \geq 2$ and the decision boundaries of $\mathcal{M}_1$ and $\mathcal{M}_2$ are approximately orthogonal. Next, we select h_3 so that the decision boundary of $\mathcal{M}_3$ is in parallel to that of $\mathcal{M}_1$ but shifted to create a much smaller attackable region. Since $\mathcal{AR}_3 \cap (\mathcal{AR}_1 \cup \mathcal{AR}_2)$ is small, the compound attack transferability $\mathcal{AT}(\{\mathcal{M}_1, \mathcal{M}_2\} \rightarrow \mathcal{M}_3)$ is low. Similarly, we select h_4 so that $\mathcal{M}_4$ has a decision boundary parallel to $\mathcal{M}_2$, while $\mathcal{AR}_4$ is much smaller than $\mathcal{AR}_2$. Also, the decision boundaries of $\mathcal{M}_3$ and $\mathcal{M}_4$ are approximately orthogonal. Following this alternating strategy, we can progressively deploy the subsequent model versions, yet each time we gradually *increase* the attackable region. As a result, the decision boundaries of $\mathcal{M}_i$ and $\mathcal{M}_{i+2}$ are parallel while those of $\mathcal{M}_i$ and $\mathcal{M}_{i+1}$ are approximately orthogonal.

Our strategy applies greedy optimization to find the best model version for the current version i, and does not assume the knowledge of N when choosing $\mathcal{M}_i$. The maximum compound transferability $\max_{i, i \leq N} \mathcal{AT}(\{\mathcal{M}_j\}_{j=1}^{i-1} \rightarrow \mathcal{M}_i)$ is zero at $N = 2$ and gradually increases with N, because the intersection of attackable regions with the previous versions increases with N. As an example, Table II shows the upper bound α_N value as a function of N, under a specific configuration of data parameters (*i.e.,* those defined by Table I). Here α_N increases gracefully with N, demonstrating the robustness of model versioning using hidden training.

N	2	4	6	8
α_N	0	0.17	0.37	0.4

TABLE II: *A realization of the model sequence and the corresponding upper bound on the compound attack transferability. The upper bound increases with length of model sequence N.*

D. Generalizing the Analysis

While our theoretical analysis targets two-dimension binary classification settings utilizing linear SVMs, it offers insights that can be applied to construct robust model versions for more complex classification tasks. Next we discuss directions in which our analysis can be expanded to those settings.

High Dimensional Settings. Extending our proof to binary classification in higher dimensions is relatively straightforward. In a d-dimensional space, the corresponding hidden features lie in a $(d-2)$-dimensional space, *e.g.,* in 3-D settings, the effective hidden features are confined to a line. We can apply similar methods used by the proofs of Theorems V.1 and V.4 to find the optimal hidden features.

Multi-class Settings. Extending our analysis to multi-class SVM is much more challenging. There exist two main methods to reason about multi-class SVMs, One versus One (OVO) and One versus Rest (OVR). For OVO, our proof naturally extends when considering protecting one class (l_1) against another class (l_2), by deriving the corresponding attackable regions. If the goal is to protect l_1 against all other classes (l_2, l_3, etc.), the attackable region will be the union of all the attackable regions against l_1. However, since this region is hard to quantify analytically, it is hard to find a closed form derivation of the optimal h. For OVR, our proof directly applies if we can assume that all data belonging to the rest of the classes lies on one side of the ($y = 0$) line, *i.e.,* we consolidate them into a single class. If the other classes are not linearly separable from l_1, then we need to employ soft-SVMs, which do not have a closed form solution.

Complex Feature Extractors. Our analysis can be extended to models that employ complex feature extractors such as DNNs and Kernel SVMs. For this we formulate the theoretical problem where the linear SVMs (used in our analysis) operate on the feature space instead of the input space. Thus, our analysis directly models the hidden features (h) that can reduce attack transferability. Here we need to make two assumptions. First, there exists an inversion process to create data in the input space that realizes the chosen hidden features. In fact, our empirical algorithm for DNN models does use such an inversion process in Algorithm 2. Second, the kernel or feature mapping of the input space must satisfy the properties described in Table I. The latter may not hold in practice exactly, but our empirical results show that the same insights do apply.

VI. Generating DNN Model Versions

Our analysis in §V establishes the theoretical foundation for the task of model versioning, but also suggests design principles for developing practical versioning algorithms for

DNN models. We now present these principles and our detailed algorithm design.

A. Design Principles

Our theoretical analysis produces three key guidelines for selecting hidden features.

- Theorem V.1 shows that a single h (per protected class) is not only adequate but also serves as a preferred optimization factor for perturbing feature space and altering attack landscape. We follow this format to choose h as the anchor for generating hidden data in the input domain.
- Theorem V.4 shows that effective hidden data exists outside the convex hull of the training data and is relatively distant from the training data. Thus, in our practical algorithm, we set the requirements that h does not overlap with features of the task training data and must keep a minimum distance to the center of training data in the feature space.
- Both Theorem V.4 and V.6 show that optimizing the choices of h can effectively reduce attack transferability compared to random selection. Furthermore, the optimal locations of hidden features do not solely depend on their distance to the original feature clusters but exhibit a complex geometric relationship. This leads us to design a greedy-search based optimization method for locating the right h values.

B. Detailed Algorithm Design

We now present the detailed algorithm for DNN model versioning. Previously, Definition IV.2 already outlines the three sequential steps for creating a model version after selecting a feature point h (per protected class). Thus, our discussion below focuses on how to select the set of feature points in $\mathcal{F}_{ori}$, one set for each model version. To streamline our presentation, we assume a single class is being protected, *i.e.,* one h value per model version. The complete process for protecting one or more classes is listed in Appendix D.

Choosing h. Conceptually, one might expect that opting for h_i values displaying greater distance from $h_1, ..., h_{i-1}$ in the feature space would produce a more distinct model version $\mathcal{M}_i$. Similarly, choosing h_i far from the feature clusters of the target class would also improve the model robustness. However, our empirical experiments show that there is no meaningful correlation between such distances[2], for either the attack transferability or the normal classification accuracy. These findings align with the third design principle in §VI-A.

Driven by these findings, we develop a greedy search approach for selecting hidden features from a predefined candidate pool. To build this candidate pool, we first identify a few "edge" points within each class's feature cluster and rescale the feature vectors of these edge points to move them farther away from the class cluster. This ensures a sufficient gap between the hidden data and any task-specific data, following the second design principle in §VI-A. Next, we train models using these feature point candidates, producing a pool of candidate models. Finally, we choose the sequence of model versions, one model at a time, from this model pool. Each time, we first generate instances of compound transferability attacks based on Projected Gradient Descent (PGD), leveraging white-box access to all previous model versions. Then, we select, from the pool, the model candidate that has the lowest attack success rate as the subsequent model version in the sequence.

Online vs. Offline Model Generation. Our algorithm for generating model versions adopts a greedy approach, *i.e.,* generating models one by one. Yet it presents the model owner with two options: (i) waiting to train a replacement model only after detecting a breach in the current version, and (ii) pretraining multiple model versions in advance. The second option takes more computation power, but offers the key advantage of immediate model replacement. This is because once a model is known to be breached, leaving it active is clearly undesirable. Similarly, taking the system offline while training a new model version is also undesirable. Swapping in a pretrained model version minimizes security risk and downtime, and is particularly important for large models that take a long time to train. Finally, for our algorithm, pretraining (*e.g.,* N) models **does not** restrict the model owner from generating additional model versions beyond N.

Curating Hidden Data. As discussed in Definition IV.2, we curate hidden data by choosing a set of initial images and perturbing them using the technique proposed by [26] so that their features in the task feature space $\mathcal{F}_{ori}$ closely match h. The choice of initial images is flexible. For our implementation, we use a well-trained GAN model [27], [28] to generate initial images. Appendix F shows samples of hidden data (*i.e.,* perturbed GAN images) produced by our experiments. We emphasize here that the key property of the hidden data is not what it looks like in the input space, but rather how close its representations are to the chosen feature point in the feature space.

VII. Empirical Study

In this section, we evaluate our hidden training method on DNN models, using three image classification tasks. We also compare our method to alternative methods to version models.

A. Experiment Setup

Datasets and Architectures. We consider three image classification tasks.

- `CIFAR10` [29] is widely used to evaluate adversarial attacks and defenses. The task is to classify 10 objects with $50,000$ training images and $10,000$ testing images. The default model architecture is ResNet-18 [30], and the number of classes is 10. We also experiment with VGG-16 [31].
- `SkinCancer` [5] consists of $10,000$ dermatoscopic images collected over 20 years. The task is to recognize 7 types of skin cancer with $8,912$ training images and $1,103$ testing images. The model architecture is Densenet-121 [32].

[2]We used three different distance metrics: averaged pairwise ℓ_2 distances, averaged pairwise cosine distances and the Earth Mover's distance (also known as 1-Wasserstein). The results are consistent across all three metrics.

	Original	Hidden Training
CIFAR10	92.1%	$91.4 \pm 0.2\%$
SkinCancer	88.7%	$90.5 \pm 0.5\%$
YouTubeFace	98.9%	$98.6 \pm 0.4\%$

TABLE III: *Performance of original and hidden trained models on clean inputs.*

- YouTubeFace [33] contains face images of $1,283$ people ($587,137$ training images and $6,4150$ testing images). The task is to perform face recognition. The model architecture is ResNet-50 [30] and the number of classes is 1283.

Attack Configurations. We consider three well-known white-box adversarial attacks: Projected Gradient Descent (PGD) [34], Carlini & Wagner attack (CW) [35], and Elastic-net Attack (EAD) [36]. We use them to build both directional and compound transferability attacks. By default, we show results of PGD since it leads to the highest attack transferability. For PGD attacks, the L_∞ perturbation budget is 0.03 for CIFAR10, 0.05 for SkinCancer, and 0.25 for YouTubeFace.

In our experiments, the attacker only submits an attack instance against the target model if it has succeeded on prior model(s). Thus, the attack success rate equals the attack transferability defined in §III. Specifically, to launch *directional* transferability attacks against $\mathcal{M}_i$, the attacker has white-box access to only one prior model version $\mathcal{M}_j$, runs a standard white-box attack to create 1000 attack instances that succeed on $\mathcal{M}_j$, and applies them to $\mathcal{M}_i$. For *compound* transferability attacks against $\mathcal{M}_i$, we use the ensemble attack from Tramèr *et. al.* [37] that leverages white-box knowledge of all previous model versions to create 1000 attack instances, ensuring that they succeed on at least one previous model version. We also consider a "cautious" attacker who only launches an attack instance against $\mathcal{M}_i$ if it succeeds on *all* prior model versions. For all cases, we report the attack transferability (and thus the attack success rate) as the fraction of attack instances launched against the target model $\mathcal{M}_i$ that actually succeed on $\mathcal{M}_i$.

Training Configuration. In hidden training, by default, we use hidden data that is 20% of the original training data of the target class. Later in §VII-D we show that varying this ratio between 10% and 30% leads to the same conclusion. We set the model pool size to 50. Additional details of training and attack configurations are in Table VII in Appendix E.

B. Performance of Hidden Training

We evaluate the effectiveness of hidden training by *normal classification accuracy*, *scalability*, and *robustness to attacks*.

Normal Classification Accuracy. One of the objectives of hidden training is to develop models whose normal model accuracy is on par with the original model (without any hidden training). To verify this, for each classification task, we generate 50 model versions using hidden training and an original model without hidden training. In Table III, we report the normal classification accuracy of all 50 model versions for each task, in terms of mean and standard deviation, along with that of the original model. We see that the models trained using hidden training achieve a normal accuracy comparable to that of the original model.

Interestingly, for SkinCancer, hidden training achieves a higher mean normal accuracy than the original model. This is likely because SkinCancer's limited training sample size and large sparsity of the training images. In this case, the hidden data injected during training essentially functions as a type of data augmentation, which enhances model generalizability.

$\mathcal{AT} = \mathcal{AT}(\mathcal{M}_1 \to \mathcal{M}_i)$	$\mathcal{AT} < 0.2$	$\mathcal{AT} < 0.3$	$\mathcal{AT} < 0.4$
# of qualified model versions	25	45	49

TABLE IV: *Richness of replacement model pool.*

Richness of Replacement Models. When a deployed model is breached by attackers, the model owner quickly recovers by deploying another model version that shares low attack transferability from the breached one(s). It is important to ensure that the set of qualified replacement models is rich enough so that the adversary cannot easily enumerate through the space to predict/construct the replacement model version.

We evaluate the richness of replacement models as follows. Given the 50 model versions generated by hidden training, we pick a model whose average attack transferability to the other 49 models is low, and set it as the breached model to be replaced. We then examine the rest 49 models and study their directional attack transferability *from* the breached model. Table IV presents the results for CIFAR10, where for 25 of the 49 models, the directional attack transferability from the breached model is less than 0.2.

Robustness against Attacks. Next, we evaluate the robustness of the model sequence created by hidden training. For each task and the pool of 50 model versions generated by hidden training, we apply the greedy search method proposed in §VI to build a sequence of $N = 8$ models.

(1) Directional Transferability Attacks – Figure 4-6 plot, for the three classification tasks, the heatmap views of the directional attack transferability from a source model j to a target model i for the sequence of 8 models. As previously mentioned, in our experiments, the attack transferability equals the attack success rate.

We see that the model versions maintain a low directional transferability, even though our method aims at reducing the compound attack transferability. The mean directional transferability is low: 19% for CIFAR10, 13% for SkinCancer and 2% for YouTubeFace. It is worth noting that, for all three tasks, there is no obvious monotonicity in the directional transferability as the model sequence grows. This means that the attacker is unable to "optimize" the choice of model j ($j < i$) to be used to attack model i.

(2) Compound Transferability Attacks – Next we evaluate how the sequence of 8 models resists the much stronger compound transferability attacks, implemented using the ensemble attack proposed in [37].

We start from the case where the attacker launches an attack instance against the target model if it succeeds on at least one prior model. Figure 7 plots, for each of the three

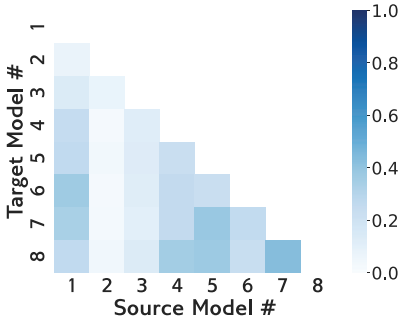

Fig. 4: *Directional attack transferability within the model sequence (*CIFAR10*, PGD). Mean=0.19.*

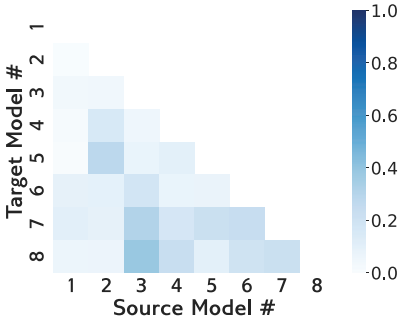

Fig. 5: *Directional attack transferability within the model sequence (*SkinCancer*, PGD). Mean=0.13.*

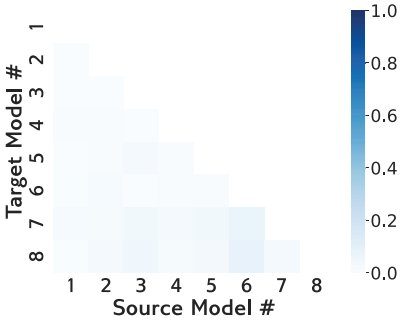

Fig. 6: *Directional attack transferability within the model sequence (*YouTubeFace*, PGD). Mean=0.02.*

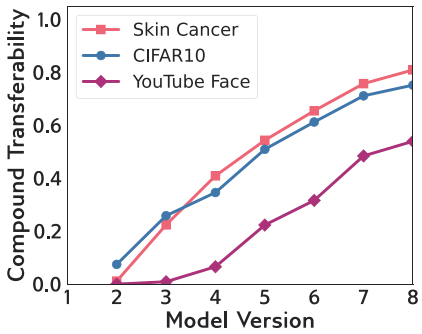

Fig. 7: *Success rate of PGD-based compound transferability attacks against a sequence of N=8 model versions.*

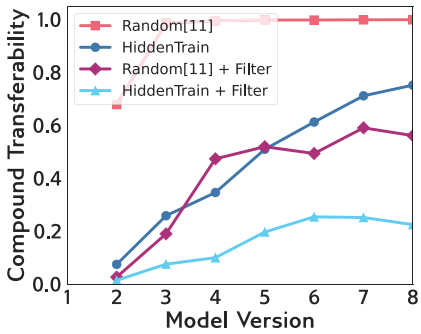

Fig. 8: *Success rate of PGD-based compound transferability attacks, ours (HiddenTrain) vs. [11] (*CIFAR10*).*

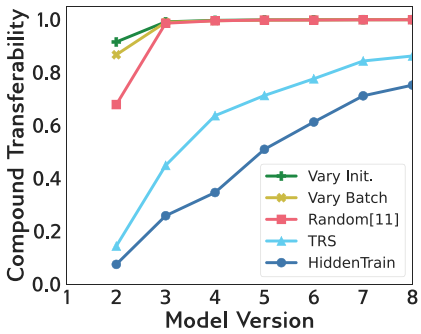

Fig. 9: *Success rate of PGD-based compound transferability attacks under different versioning methods (*CIFAR10*).*

classification tasks, the attack success rate for each model version in the sequence, which grows gracefully with the model version count. This trend aligns with our analytical findings in §V. Moreover, we find that 90% of attack instances generated using the ensemble attack already succeed on *all* previous model versions, demonstrating the powerfulness of the ensemble attack and the effectiveness of our method.

Next we consider "cautious" attackers who only deploy attack instances that succeed on all prior versions. Table VIII (Appendix F) shows that they only produce a minor increase in the compound attack transferability. Such increase can be effectively suppressed by combining hidden training with run-time input filtering proposed by [11] (discussed next).

(3) Hidden Training + Run-time Detection – As a training-time defense, hidden training can be combined with run-time attack detection systems to improve resilience against compound transferability attacks. To illustrate this, we implement, for each model version, the input filter proposed by [11] to identify whether an input is an attack generated from any of the previous model versions and block any recognized as such. Under our scenario, these detected attack inputs now carry zero transferability. Figure 8 plots the attack success rate (in terms of transferability) with and without the filter, for CIFAR10. The combined defense effectively suppresses the attack, *e.g.,* for model version 8, the attack success rate reduces drastically from 75.2% to 22.5%!

To evaluate the "contribution" of hidden training to this combined defense, we plot in the same figure the results of the model versioning proposed by [11] with and without the input filter. We note that [11] does not pick hidden features but randomly selects a set of GAN-generated images as the additional data to train model versions. The large gain over [11] demonstrates the effectiveness of our hidden data selection process in building a more robust model sequence. The increased diversity of models in the sequence also contributes to enhancing the effectiveness of real-time attack detection.

(4) Comparison to Alternatives – We compare hidden training with baseline techniques presented in §III-C: varying model initialization, varying training batch order, computing model ensembles (TRS [9]), and the random selection method [11]. Here we implement TRS [9] assuming that the model owner needs to train 8 models. Due to the extreme high training cost of TRS (see §VII-C), we were only able to train the TRS models for CIFAR10. Results in Figure 9 show the success rate of compound transferability attacks. Here we can clearly observe the advantage of hidden training in generating a robust model sequence.

C. Computation Overhead

We examine computation overhead for hidden training and model versioning. Compared to training the original model, generating a sequence of $N = 8$ models requires (1) producing

hidden data required to train a pool of 50 models, (2) training 50 models, and (3) running the greedy algorithm to select a sequence of 8 models from the model pool. We find that the computation overhead is dominated by model training.

	Original Model	Hidden Training		TRS
		(1 Model)	(50 Models)	(8 Models)
CIFAR10	34.6 s	34.8 s	29 min	$>$ 100 min
SkinCancer	193.5 s	334 s	4.6 hr	$>$ 30 hr
YouTubeFace	62.1 s	112.5 s	1.5 hr	$>$ 16 hr

TABLE V: *Time spent to train a single epoch.*

Table V lists the time required to train a single epoch for the three classification tasks, using a Titan RTX GPU. We also provide the training time for the original model and for TRS [9], assuming that the model owner sets up TRS to train 8 models as an ensemble. These results show that our method consumes significantly less time compared to TRS, even when creating a pool of 50 models. For both SkinCancer and YouTubeFace, we encountered convergence issues when attempting to produce 8 models using TRS, as each training epoch exceeded 16 hours in duration.

	Generate Hidden Data	Optimizing Sequence
CIFAR10	70.8 s	$<$ 6 min
SkinCancer	824.8 s	$<$ 15 min
YouTubeFace	173.6 s	$<$ 15 min

TABLE VI: *Computation overhead beyond model training.*

Table VI lists the additional overhead required beyond model training, including time taken to generate and perturb 2000 images for a given feature point h, and average time required to identify the next subsequent model. For latter, the main overhead is to generate adversarial examples using test data and estimate the compound attack transferability. This table shows that the time for generating hidden training data per model is comparable to 2 epochs of hidden training, indicating that the overall overhead is still dominated by model training. The optimization time to produce a sequence of N models is less than 15 minutes$\times(N-1)$, significantly less than the time required to train $N = 8$ TRS models.

D. Ablation Study

Attack Configuration. So far we report results assuming the attacker launch PGD based attacks, which are known to carry strong transferability across models. We also evaluate hidden training on two other white-box adversarial attacks: CW [35] and EAD [36]. Figure 15 and 16 (Appendix F) show the compound attack transferability of multiple model versioning methods, for both attacks. Again, hidden training is the most effective at producing robust model sequences.

Hidden Data Portion. We also implement hidden training by varying the portion of hidden data (relatively to the task training data) from 10% to 30%, with 20% being the default configuration in our experiments. Results in Figure 17 (Appendix F) indicate that varying the proportion of hidden data does not affect the effectiveness of hidden training in maintaining a low compound attack transferability. Furthermore, all versions of the hidden-trained models consistently achieve high normal classification accuracy comparable to that of the original model (results omitted for brevity).

Impact of Model Architecture. In addition to ResNet-18, we also conduct hidden training using VGG-16 [31]. Figure 18 (Appendix F), shows that hidden training yields the lowest compound attack transferability, demonstrating its applicability across multiple model architectures.

Protected Classes. We vary the number of protected classes (see Appendix F) and obtain consistent results: model versioning via hidden training can protect multiple classes against compound transferability attacks.

VIII. Conclusion and Limitations

As classifiers are increasingly deployed in industrial settings, data breaches will inevitably impact stored ML models, causing white-box model breaches. This motivates us to address the pressing problem of robust model versioning to maintain reliable ML services despite repeated model leakages. We introduce model versioning via hidden training, and demonstrate both *theoretically* and *empirically* that, when properly configured, they can produce model versions robust against multiple transferability-based attacks while achieving high task accuracy. Compared to alternative methods such as random seeding and model ensembles, our method achieves higher robustness (against compound transferability attacks), high scalability and low cost for training model versions.

Limitations and Future Directions. As the first work in this area, our work faces several limitations that warrant additional research efforts. First, our theoretical analysis considers linear SVM models. Further work is necessary to extend it to DNNs. Second, we propose a greedy method to progressively construct DNN model versions from a pool of candidates, demonstrating the feasibility and benefits of hidden training. Yet the overhead for generating such pool of models can be heavy, especially for large models and for protecting many/all of the model classes simultaneously. Thus, we are unable to evaluate our design on large datasets like ImageNet. Additional efforts are needed to produce stronger and more efficient optimization methods. Finally, our experiments consider a loss-based, compound attack to produce adversarial examples, defined by a prior work [37]. Follow-up research should study the feasibility of stronger attacks and refine the model versioning design to resist such attacks.

Acknowledgements

We thank the anonymous reviewers for their insightful feedback. We thank Dr. Avrim Blum for his feedback on an earlier version of the work, and Shawn Shan for his help on configuring the experiments. This work is supported in part by NSF grants CNS2241303 and CNS1949650, the DARPA GARD program, and the C3.ai DTI program. Wenxin Ding is supported by an Eckhardt Fellowship at the University of Chicago. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any funding agencies.

REFERENCES

[1] S. G. Finlayson, J. D. Bowers, J. Ito, J. L. Zittrain, A. L. Beam, and I. S. Kohane, "Adversarial attacks on medical machine learning," *Science*, no. 6433, 2019.

[2] X. Ma, Y. Niu, L. Gu, Y. Wang, Y. Zhao, J. Bailey, and F. Lu, "Understanding adversarial attacks on deep learning based medical image analysis systems," *Pattern Recognition*, vol. 110, 2021.

[3] M. B. Rahman, H. A. Mustafa, and M. D. Hossain, "Towards evaluating robustness of violence detection in videos using cross-domain transferability," *Journal of Information Security and Applications*, 2023.

[4] A. N. Bhagoji, W. He, B. Li, and D. Song, "Practical black-box attacks on deep neural networks using efficient query mechanisms," in *Proc. of ECCV*, 2018.

[5] P. Tschandl, C. Rosendahl, and H. Kittler, "The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions," *Scientific data*, 2018.

[6] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli, "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks," in *Proc. of USENIX Security*, 2019.

[7] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," in *Proc. of ICLR*, 2017.

[8] H. Yang, J. Zhang, H. Dong, N. Inkawhich, A. Gardner, A. Touchet, W. Wilkes, H. Berry, and H. Li, "Dverge: diversifying vulnerabilities for enhanced robust generation of ensembles," *Proc. of NeurIPS*, vol. 33, 2020.

[9] Z. Yang, L. Li, X. Xu, S. Zuo, Q. Chen, P. Zhou, B. Rubinstein, C. Zhang, and B. Li, "Trs: Transferability reduced ensemble via promoting gradient diversity and model smoothness," *Proc. of NeurIPS*, 2021.

[10] H. Dbouk and N. Shanbhag, "Adversarial vulnerability of randomized ensembles," in *Proc. of ICML*, 2022.

[11] S. Shan, W. Ding, E. Wenger, H. Zheng, and B. Y. Zhao, "Post-breach recovery: Protection against white-box adversarial examples for leaked dnn models," in *Proc. of CCS*, 2022.

[12] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. of ICLR*, 2014.

[13] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proc. of ICLR*, 2014.

[14] W. Wu, Y. Su, M. R. Lyu, and I. King, "Improving the transferability of adversarial samples with adversarial transformations," in *Proc. of CVPR*, 2021.

[15] F. Suya, J. Chi, D. Evans, and Y. Tian, "Hybrid batch attacks: Finding black-box adversarial examples with limited queries," in *Proc. of USENIX Security*, 2020.

[16] N. Inkawhich, K. J. Liang, L. Carin, and Y. Chen, "Transferable perturbations of deep feature distributions," in *Proc. of ICLR*, 2020.

[17] J. Springer, M. Mitchell, and G. Kenyon, "A little robustness goes a long way: Leveraging robust features for targeted transfer attacks," *Proc. of NeurIPS*, 2021.

[18] C. Cianfarani, A. N. Bhagoji, V. Sehwag, B. Zhao, H. Zheng, and P. Mittal, "Understanding robust learning through the lens of representation similarities," *Proc. of NeurIPS*, vol. 35, 2022.

[19] C. Wiedeman and G. Wang, "Disrupting adversarial transferability in deep neural networks," *Patterns*, 2022.

[20] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[21] GoogleCloud, "Mlops: Continuous delivery and automation pipelines in machine learning," https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning, 2023.

[22] Stable Diffusion, "Hugging face stable diffusion," https://huggingface.co/CompVis/stable-diffusion, accessed: 2023-24-01.

[23] T. Xu, G. Goossen, H. K. Cevahir, S. Khodeir, Y. Jin, F. Li, S. Shan, S. Patel, D. Freeman, and P. Pearce, "Deep entity classification: Abusive account detection for online social networks," in *Proc. of USENIX Security*, 2021.

[24] S. Shan, A. N. Bhagoji, H. Zheng, and B. Y. Zhao, "Poison forensics: Traceback of data poisoning attacks in neural networks," in *Proc. of USENIX Security*, 2022.

[25] B. Schölkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[26] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," *arXiv preprint arXiv:1804.00792*, 2018.

[27] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Proc. of NeurIPS*, 2014.

[28] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *Proc. of ICLR*, 2017.

[29] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of CVPR*, 2016.

[31] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[32] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. of CVPR*, 2017.

[33] "https://www.cs.tau.ac.il/~wolf/ytfaces/," YouTube Faces DB.

[34] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Artificial intelligence safety and security*. Chapman and Hall/CRC, 2018.

[35] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. of IEEE S&P*, 2017.

[36] P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C.-J. Hsieh, "Ead: elastic-net attacks to deep neural networks via adversarial examples," in *Proc. of AAAI*, 2018.

[37] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *arXiv preprint arXiv:1705.07204*, 2017.

In this appendix, we list the detailed proofs for the three theorems presented §V. We then present the hidden training algorithm in Appendix D, and additional empirical results in Appendix F.

APPENDIX A
PROOF OF THEOREM V.1

In this proof, we explicitly compute the linear decision boundary of an SVM model in the presence of an added hidden data $h = (v, w)$. Without loss of generality, we assume $1 - c < v < c - 1$ and $|w| \leq z$. We use Figure 10 to illustrate the notations and the process of finding the SVM decision boundary described below. As stated earlier, in the SVM setting considered by our analysis, the input space and the feature space are identical.

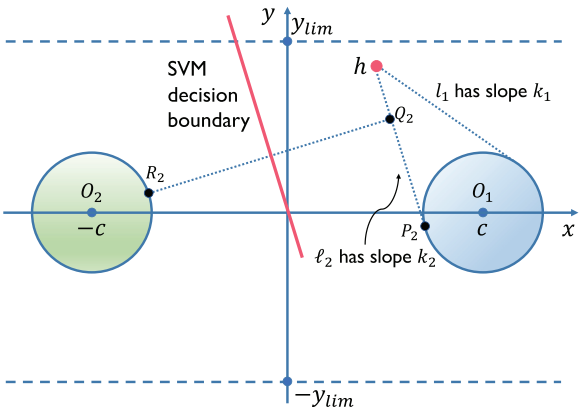

Fig. 10: Illustration of an SVM decision boundary.

We use $\mathcal{R}(p, q)$ to denote a unit circle centered at (p, q), *i.e.,* $\mathcal{R}(p, q) = \{(x, y) \in \mathbb{R}^2 | (x - p)^2 + (y - q)^2 \leq 1\}$. As in the problem setup, we have two classes, $+$ and $-$. Training data for class $+$, denoted $\mathcal{D}_+$, is uniformly distributed in $\mathcal{R}(c, 0)$. Training data $\mathcal{D}_-$ is uniformly distributed in $\mathcal{R}(-c, 0)$. Let O_1 denote a center $(c, 0)$ and O_2 denote the other center $(-c, 0)$, and we have $c > 1$.

In a two-dimensional space, the optimal classifier is the one bisecting the shortest connection between the convex hulls of $\mathcal{D}_+$ and $\mathcal{D}_-$ [25]. Therefore, we first find the convex hull with new training data h in class $+$ and the other convex hull $\mathcal{R}(-c, 0)$ is not affected by h. We know that the convex hull of $\mathcal{D}_+ \cup \{h\}$ is enclosed by $\mathcal{R}(c, 0)$ and the line segments tangent to $\mathcal{R}(c, 0)$ passing through h.

To compute the lines tangent to $\mathcal{R}(c, 0)$, we assume the line is of the form $y = kx + b$. We then solve for k and b such that Equation 1 only has one solution for x. Note that since $1 - c < v < c - 1$, no tangent line is of the form $x = b$.

$$\begin{cases} (x - c)^2 + y^2 = 1 \\ y = kx + b. \end{cases} \tag{1}$$

This yields

$$(x - c)^2 + (kx + b)^2 = 1 \tag{2}$$

and the line being tangent indicates that the quadratic function in Equation 2 has only one solution for x. Therefore, we solve for k using

$$\begin{cases} (2kb - 2u)^2 - 4(1 + k^2)(u^2 + b^2 - 1) = 0 \\ w = kv + b. \end{cases}$$

The tangent lines are denoted ℓ_1 and ℓ_2, have slopes $k_1 = \frac{-w(c-v)+\sqrt{(c-v)^2+w^2-1}}{(c-v)^2-1}$ and $k_2 = \frac{-w(c-v)-\sqrt{(c-v)^2+w^2-1}}{(c-v)^2-1}$, and have intercepts $w - k_1 v$ and $w - k_2 v$ respectively. Note that $k_1 > k_2$. Let P_1 and P_2 denote the tangent point on $\mathcal{R}(c, 0)$ by line ℓ_1 and ℓ_2 respectively.

If $w = 0$, by symmetry of the convex hulls, the classifier is $x = \frac{-c+v+1}{2}$.

If $w > 0$, the shortest connection can be found by finding the shortest distance between O_2 and the line segment $P_2 h$. Any point in the convex hull other than $P_2 h$ has greater distance to $\mathcal{R}(-c, 0)$.

The orthogonal line to ℓ_2 has slope $-\frac{1}{k_2}$. Let the line pass through O_2, we can then compute for its intercept. This line is of the form $y = -\frac{1}{k_2}(x - c)$. Then we set the above computed line to intersect with ℓ_2. We denote the intersection point by

Q_2. Q_2 has coordinates $(\frac{k_2^2v-k_2w-c}{k_2^2+1}, \frac{-k_2c-k_2v+w}{k_2^2+1})$. If $\frac{k_2^2v-k_2w-c}{k_2^2+1} > v$ then the Q_2 is on the line segment P_2h. In this case O_2Q_2 is the shortest distance between O_2 and the convex hull. Otherwise, the shortest distance is hO_2.

If Q_2 is on the line segment P_2h, we find the point on $\mathcal{R}(-c,0)$ that is intersected by O_2Q_2, denoted by R_2. We solve the equation

$$\begin{cases} (x+c)^2+y^2=1 \\ y=-\frac{1}{k_2}(x-c) \end{cases}$$

to get the coordinates of R_2 to be $(\frac{-k_2}{\sqrt{k_2^2+1}}-c, \frac{1}{\sqrt{k_2^2+1}})$. For the line bisecting Q_2R_2, it has the same slope as ℓ_2. To find the intercept, we let the line pass through the mid-point of Q_2R_2.

Otherwise, we let R_2 be the point on $\mathcal{R}(-c,0)$ that is intersected by O_2h. We solve the equation

$$\begin{cases} (x+c)^2+y^2=1 \\ y=\frac{w}{c+v}(x+c) \end{cases}$$

to get the coordinates of R_2 to be $(\frac{c+v}{\sqrt{(c+v)^2+w^2}}-c, \frac{w}{\sqrt{(c+v)^2+w^2}})$. We then bisect the line segment R_2h to get the decision boundary.

Therefore,

- If $\frac{k_2^2v-k_2w-c}{k_2^2+1} > v$, the shortest distance between the convex hulls is between $(\frac{k_2^2v-k_2w-c}{k_2^2+1}, \frac{-k_2c-k_2v+w}{k_2^2+1})$ and $(\frac{-k_2}{\sqrt{k_2^2+1}}-c, \frac{1}{\sqrt{k_2^2+1}})$. Bisecting the line, we can see that the decision boundary has slope $K(h)=k_2$ and intercept $B(h)=\frac{-k_2c+k_2v-w-\sqrt{k_2^2+1}}{2}$.
- Otherwise, the shortest distance between the convex hulls is between (v,w) and $(\frac{c+v}{\sqrt{(c+v)^2+w^2}}-c, \frac{w}{\sqrt{(c+v)^2+w^2}})$. Bisecting the line, we can see that the decision boundary has slope $\frac{-c-v}{w}$ and intercept $\frac{-c^2+v^2+w^2+\sqrt{(c+v)^2+w^2}}{2w}$.

When $w<0$, by symmetry, we repeat the same analysis as in the case when $w>0$. We omit some details in this case.

If $w<0$, the shortest connection can be found by finding the shortest distance between O_2 and the line segment with slope k_1. The orthogonal line to ℓ_1 passing through O_2 intersects ℓ_1 at point Q_1 with coordinate $(\frac{k_1^2v-k_1w-c}{k_1^2+1}, \frac{-k_1c-k_1v+w}{k_1^2+1})$. If $\frac{k_1^2v-k_1w-c}{k_1^2+1} > v$ then the intersection is on the tangent line segment and O_2Q_1 is the shortest distance between O_2 and the convex hull. Otherwise, the shortest distance is O_2P.

- If $\frac{k_1^2v-k_1w-c}{k_1^2+1} > v$, the shortest distance between the convex hulls is between $(\frac{k_1^2v-k_1w-c}{k_1^2+1}, \frac{-k_1c-k_1v+w}{k_1^2+1})$ and $(\frac{k_1}{\sqrt{k_1^2+1}}-c, -\frac{1}{\sqrt{k_1^2+1}})$. Bisecting the line, we can see that the decision boundary has slope $K(h)=k_1$ and intercept $B(h)=\frac{k_1c-k_1v+w-\sqrt{k_1^2+1}}{2}$.
- Otherwise, the shortest distance between the convex hulls is between (v,w) and $(\frac{c+v}{\sqrt{(c+v)^2+w^2}}-c, \frac{w}{\sqrt{(c+v)^2+w^2}})$. Bisecting the line, we can see that the decision boundary has slope $K(h)=\frac{-c-v}{w}$ and intercept $B(h)=\frac{-c^2+v^2+w^2+\sqrt{(c+v)^2+w^2}}{2w}$.

APPENDIX B
PROOF OF THEOREM V.4

First, we will show that if two SVM decision boundaries have opposite signs of slope and intersect at (x_I, y_I) with $x_I \geq \delta$, then their attackable region has empty intersection. This yields zero directional attack transferability. We then explain how to find the hidden data h that satisfies the desired property.

Without loss of generality, let $\mathcal{M}_1$ have the decision boundary $y=k_1x+b_1$ with $k_1<0$ and $\mathcal{M}_2$ have the decision boundary $y=k_2x+b_2$ with $k_2>0$. Then for any $(x,y)\in\mathcal{AR}_1$, we have $y \geq k_1\delta+b_1$. Similarly, for any $(x,y)\in\mathcal{AR}_1$, we have $y \leq k_2\delta+b_2$.

We can also compute the intersection of the two decision boundaries, where $x_I=\frac{b_2-b_1}{k_1-k_2}$. Since $x_I \geq \delta$, we have

$$\begin{aligned} &\frac{b_2-b_1}{k_1-k_2} \geq \delta \\ \implies & b_2-b_1 \leq \delta\cdot(k_1-k_2) \text{ since } k_1<0<k_2 \\ \implies & k_2\delta+b_2 \leq k_1\delta+b_1. \end{aligned}$$

Therefore, $\mathcal{AR}_1 \cap \mathcal{AR}_2 = \emptyset$, which implies $\mathcal{AT}_{1\to2}=\mathcal{AT}_{2\to1}=0$.

We now explain how to find h given a desired decision boundary $y=kx+b$. Without loss of generality, we assume $k>0$.

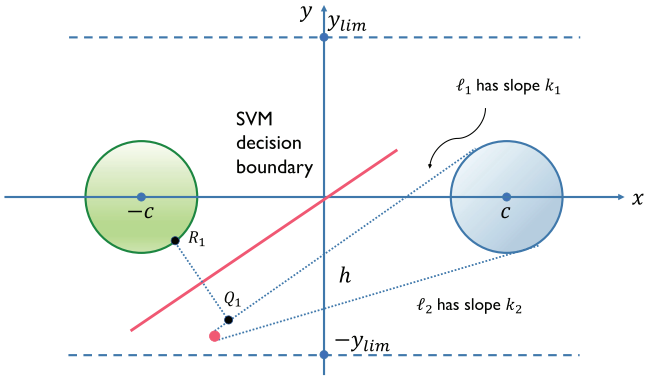

Fig. 11: Illustration of a SVM decision boundary.

By [25], we know that the decision boundary has to bisect the convex hull of the two classes. Moreover, the convex hull of class $-$ is the unit ball at $(-c, 0)$. Therefore, we can find the function for the line that the decision boundary bisects, which is R_1Q_1 in Figure 11: $y = -\frac{1}{k}(x + c)$. Using this line, we can further find point R_1 by solving the equations

$$\begin{cases} (x+c)^2 + y^2 = 1 \\ y = -\frac{1}{k}(x+c). \end{cases}$$

Solving the equations yields the coordinates of $R_1 : (\frac{k}{k^2+1} - c, -\frac{1}{\sqrt{k^2+1}})$. The mid point of R_1Q_1, which is also the intersection of the decision boundary with R_1Q_1, has coordinates that satisfy

$$\begin{cases} y = kx + b \\ y = -\frac{1}{k}(x+c). \end{cases}$$

Therefore, the mid point is $(\frac{-bk-c}{k^2+1}, \frac{b-ck}{k^2+1})$.

Using the coordinates of R_1 and mid point, we can find the coordinates of Q_1 because the mid point can also be found by $\frac{R_1+Q_1}{2}$. Thus, Q_1 has coordinates $(\frac{2(-bk-c)}{k^2+1} - \frac{k}{\sqrt{k^2+1}} + c, \frac{2(b-ck)}{k^2+1} + \frac{1}{\sqrt{k^2+1}})$.

If Q_1 is a valid choice as h, then we can choose $h = Q_1$ and the resulting decision boundary will be the desired $y = kx + b$. The constraints are:

1) $1 - c < \frac{2(-bk-c)}{k^2+1} - \frac{k}{\sqrt{k^2+1}} + c < c - 1$
2) $|\frac{2(b-ck)}{k^2+1} + \frac{1}{\sqrt{k^2+1}}| \leq y_{\text{lim}}$
3) $k_1 \cdot -\frac{1}{k} \geq -1$ where k_1 is the larger slope of the line passing through Q_1 and tangent to the unit ball of class $+$

The first two constraints enforce the h to be within the feasible region where we can add hidden data. The third constraint ensures that with $h = Q_1$, the decision boundary is indeed the desired one. If $k_1 \cdot -\frac{1}{k} < -1$, then R_1Q_1 is not the shortest distance between the convex hulls. Then the decision boundary would not bisect R_1Q_1. Therefore, we can check the feasibility of the decision boundary using the constraints.

With the algorithm for checking feasibility, we can iteratively search through the space to find a feasible linear separator.

APPENDIX C
PROOF OF THEOREM V.6

In Appendix B, we discuss how we validate feasibility of an SVM decision boundary. In this section, we explain how we construct a sequence of SVM models such that we can upper bound the maximum compound transferability of the sequence.

When $N = 2$: We build $\mathcal{M}_1$ by selecting some $k > 0$ such that $y = k(x - \delta)$ is a feasible decision boundary, using the verification method discussed in Appendix B. Since $y = k(x - \delta)$ is a feasible decision boundary, then by symmetry, $y = -k(x - \delta)$ is also feasible. Therefore, we can build $\mathcal{M}_1$ with decision boundary $y = k(x - \delta)$ and $\mathcal{M}_2$ with decision boundary $y = -k(x - \delta)$. By Appendix B, we have $\mathcal{AR}_1 \cap \mathcal{AR}_2 =$ so $\mathcal{AT}(\mathcal{M}_1 \to \mathcal{M}_2) = \mathcal{AT}(\mathcal{M}_2 \to \mathcal{M}_1) = 0$.

When $N > 2$: In this general case, we also start by finding a feasible decision boundary $y = k(x - \delta)$. Ideally, a larger k results in a smaller attackable region. However, since we need to choose a sequence of models with large separations, the chosen k cannot be too large. After selecting a feasible k value, we seek to find the largest $b > 0$ such that the decision

boundary $y = kx - b$ remains feasible. We denote this maximum value of b as $b_{\max}$. Here we observe that as k increases, the value of b_{max} satisfying the above requirement decreases.

Next we set $n = \lceil \frac{N}{2} \rceil - 1$ and $b = \frac{b_{\max}}{n}$. We set $\mathcal{M}_1$ with decision boundary $y = k(x - \delta)$ and $\mathcal{M}_2$ with $y = -k(x - \delta)$. For $\mathcal{M}_i$ ($i > 2$), if i is odd, we set its decision boundary to $y = k(x - \delta) - b \cdot \frac{i-1}{2}$; if i is even, we set its decision boundary to $y = -k(x - \delta) + b \cdot \frac{i-2}{2}$.

As i increases, $\mathcal{AR}_i$ decreases under our construction. Moreover, $\cup_{j=1}^{i-1} \mathcal{AR}_j = \mathcal{AR}_1 \cup \mathcal{AR}_2$ for all $i > 2$. Therefore, the compound transferability of the sequence is bounded by α_N where $\alpha_N = \mathcal{AT}(\{\mathcal{M}_j\}_{j=1}^{2} \rightarrow \mathcal{M}_3)$. This ends our proof.

Specific Realization Shown in Table 1. As an illustrative example, we assume $c = 100$, $y_{lim} = 30$ and $\delta = 0.1$. We choose $k = 7$ and select $b_{\max} = 12$. Note that actual $b_{\max}$ is slightly greater than 12, but we choose 12 to simplify the computation.

For an SVM with decision boundary $y = kx - b$ satisfying $k > 0$, $b > 0$ and $\frac{b - y_{lim}}{k} < -\delta$, we have $\mathcal{S}(\mathcal{AR}) = \frac{(y_{lim} - k \cdot \delta - b)^2}{2k} + \delta \cdot (y_{lim} - b + \frac{k\delta}{2})$. This is computed by summing up the area of a triangle and a trapezoid, as shown in the shadowed area in Figure 3. Given the above configuration, we have $\mathcal{S}(\mathcal{AR}_1) = \mathcal{S}(\mathcal{AR}_2) = 61.39$, with $\mathcal{AR}_1 \cap \mathcal{AR}_2 = \emptyset$.

Next, we compute α_N for each given N value. When $3 \leq N \leq 4$, we have $b = 12$ and $\alpha_N = \frac{\mathcal{S}(\mathcal{AR}_3)}{2\mathcal{S}(\mathcal{AR}_1)} = 0.17$. When $5 \leq N \leq 6$, we have $b = 6$ and $\alpha_N = 0.32$. When $7 \leq N \leq 8$, we have $b = 4$ and $\alpha_N = 0.37$. And finally when $9 \leq N \leq 10$, we have $b = 3$ and $\alpha_N = 0.4$.

APPENDIX D
HIDDEN TRAINING ALGORITHM

In the following, we provide a detailed discussion on our hidden training algorithm, in the context of producing a sequence of models. This is done in four steps.

Step 1: Selecting a candidate set of feature points (Algorithm 1). Our analytical study in Appendix B and Appendix C shows that effective hidden feature points often lie at the edge of the classes. Therefore, we first examine the original feature space and choose the feature vectors that are located at a distance from the center of each original class. Here we face two constraints: (1) features in the original feature space that are too far from the original classes may not be realizable in the input space, *i.e.,* it is difficult to generate the corresponding input data, and (2) the chosen feature points should not overlap with feature vectors of the task classes. These two considerations lead us to apply a feature-multiplier m to "drag" existing feature vectors away from the original classes, and bound m by $1.1 < m < 1.5$. As such, Algorithm 1 outputs a list of chosen feature points.

Step 2: Generating hidden data in the input space (Algorithm 2). Next, we generate a set of hidden data (in the input space) from each chosen feature point h and use them to train a model version parameterized by h. For this, we apply the algorithm from [26], which perturbs a given input x to move its feature vector to the target feature vector in the original feature space. Here we use a pre-trained GAN model [28] to generate a set of 2000 images as the initial inputs, and perturb each input to reach the chosen h. For each h, we apply up to 100 perturbation iterations (without any perturbation budget) to produce the corresponding hidden data. We then split the hidden data into training and testing sets, where the hidden data used for training is 20% of the original training data. The testing hidden data is used to validate the model performance.

Later in Figure 12, 13 and 14, we provide sample images of the hidden data generated for the three classification tasks used in our experiments. We would like to emphasize here that the key property of the hidden data is not what it looks like in the input space, but rather how close its representations are to the chosen feature point in feature space. For other application domains, the visual representation is a moot point.

Step 3: Training candidate model versions. Now given the hidden data produced in Step 2, we can run the usual model training by adding these data to the training set. This step creates a set of candidate model versions, referred to as $\mathcal{S}_{\mathcal{M}}$.

Step 4: Configuring the model sequence (Algorithm 3). Given the model pool $\mathcal{S}_{\mathcal{M}}$, we apply a greedy search method to choose the sequence of models $\mathcal{M}_1, \mathcal{M}_2, ...\mathcal{M}_i, ...$ The goal is to find $\mathcal{M} \in \mathcal{S}_{\mathcal{M}}$ that has the lowest compound attack transferability from $\mathcal{M}_1, ..., \mathcal{M}_{i-1}$. Ideally, we would like to search the whole feature space to find the feature point h_i that minimizes compound transferability. However, such global optimization is very difficult because the impact of h_i on the trained model version cannot be explicitly formulated. Instead, we apply a practical, greedy search method to gradually choose from the pool the next version i. The selection is driven by computing the compound transferability attacks and launching them on the candidate models to estimate the compound transferability. The candidate model with the lowest transferability is then selected. Therefore, the larger the pool size, the better the optimization. For our implementation, the pool size is 50 to achieve a low computation overhead.

Algorithm 1 Finding Feature Points

Input: Original model $\mathcal{M}_{ori}$, test data D_{test}, class labels $\mathcal{L}$, distance metric d, distance threshold ϵ_d, multiplier m
Output: a list of N feature points
Feature points $\mathcal{H} \leftarrow \{\}$
$\mathcal{F}_{ori} \leftarrow$ feature extractor from $\mathcal{M}_{ori}$
for $\ell \in \mathcal{L}$ **do**
  $X_\ell = \{(x, c) \in D_{\text{test}} \mid c = \ell\}$
  $F_\ell = \mathcal{F}_{ori}(X_c)$
  $h_\ell = mean(F_\ell)$
  for $h \in F_\ell$ **do**
    if $d(m * h, h_\ell) > \epsilon_d$ **then**
      Add $m * h$ to $\mathcal{H}$
    end if
  end for
end for
Output $\mathcal{H}$

Algorithm 2 Hidden Training

Input: Original model $\mathcal{M}_{ori}$, training data D_{train}, set of target classes $\mathcal{L}_T$, list of GAN images $\mathcal{S}_\mathcal{G}$, list of feature points $\mathcal{S}_\mathcal{H}$, perturbation algorithm from [26] $Perturb(.)$
Output: a list of hidden trained model versions $\mathcal{S}_\mathcal{M}$
Model list $\mathcal{S}_\mathcal{M} \leftarrow \{\}$
Number of protected classes $n_t \leftarrow len(\mathcal{L}_T)$
for i in $(0, len(\mathcal{S}_\mathcal{H}), n_t)$ **do**
  $D_{\text{hidden}_i} \leftarrow \{\}$
  for j, l_t in $enumerate(\mathcal{L}_T)$ **do**
    $g \leftarrow \mathcal{S}_\mathcal{G}[i * n_t + j]$
    $h \leftarrow \mathcal{S}_\mathcal{H}[i * n_t + j]$
    $X_h \leftarrow Perturb(\mathcal{M}_{ori}, h, g)$
    $D_{\text{hidden}_i} = D_{\text{hidden}_i} \cup \{(x, l_t) \mid x \in X_h\}$
  end for
  Train $\mathcal{M}_i$ with $D_{\text{train}} \cup D_{\text{hidden}_i}$
  Add $\mathcal{M}_i$ to $\mathcal{S}_\mathcal{M}$
end for
Output $\mathcal{S}_\mathcal{M}$

Algorithm 3 Forming Model Sequence by Greedy Search

Input: List of hidden trained models $\mathcal{S}_\mathcal{M}$, list of breached models $\mathcal{B}_\mathcal{M}$, adversarial example generating algorithm Adv, transferability computation algorithm $Trans(.)$, training data D_{test}, set of target classes L_T
Output: a model to be deployed
for l_t in $enumerate(L_T)$ **do**
  $X_{adv} = X_{adv} \cup Adv(\mathcal{B}_\mathcal{M}, D_{\text{test}}, l_t)$
end for
$\mathcal{M} = \arg\min_{\mathcal{M}_i \in \mathcal{S}_\mathcal{M}, \mathcal{M}_i \notin \mathcal{B}_\mathcal{M}} Trans(\mathcal{M}_i, X_{adv})$
Output $\mathcal{M}$

APPENDIX E
TRAINING AND ATTACK CONFIGURATIONS

In this section, we add additional information on the training and attack configurations. The configurations are listed in Table VII. We show the training configuration of the models in the original results and the attack configurations of PGD attacks.

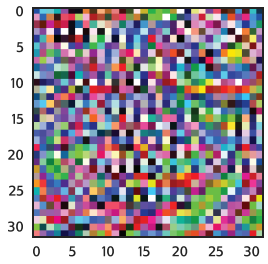

Fig. 12: *Sample hidden data used for* `CIFAR10`.

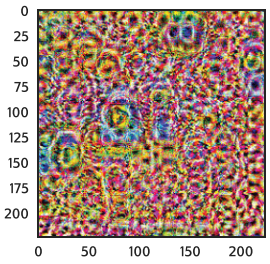

Fig. 13: *Sample hidden data used for* `SkinCancer`.

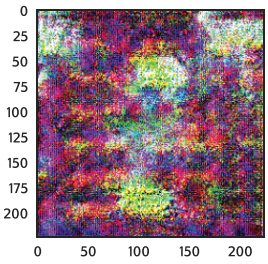

Fig. 14: *Sample hidden data used for* `YouTubeFace`.

	Training					Attack		
	Architecture	Epochs	Batch Size	Optimizer	Learning Rate	Perturbation Budget	Iterations	α
`CIFAR10`	ResNet-18	20	512	SGD	0.5	0.03	30	0.01
`SkinCancer`	Densenet-121	10	64	Adam	$1e-3$	0.05	30	0.01
`YouTubeFace`	ResNet-50	20	32	Adam	$1e-4$	0.25	100	0.05

TABLE VII: *Training and attack configurations.*

APPENDIX F
ADDITIONAL RESULTS ON ROBUSTNESS

In this section, we provide additional results for ablation study on the impact of model architecture and the number of protected classes.

Impact of Model Architecture We train VGG-16 models for the `CIFAR10` dataset and study the compound attack transferability for a sequence of 8 models. Figure 18 plots the success rate of compound transferability attacks, for both hidden training and four alternative methods. Again hidden training outperforms its alternatives. This result is consistent with prior results using ResNet18 models, demonstrating the applicability of hidden training across multiple model architectures.

Protected Classes. We also vary the number of protected classes when applying hidden training to create the model sequence. In Table IX, we present the compound attack transferability experienced by the 8th model in the sequence. Maintaining robustness at this model version is the hardest among the eight model versions, as the attacker now possesses white-box access to all preceding seven models. Additionally, we also include the results when combining hidden training with run-time attack detection and filtering, along with the results of using random hidden feature selection [11].

The results in Table IX show that protecting either 1 or 3 classes yields similar robustness performance. Furthermore, combining hidden training with run-time attack detection is highly effective in sustaining robustness even when the requirement is to protect a larger number of classes. As expected, hidden training largely outperforms random selection [11], with or without run-time filtering. Together, these findings demonstrate the ability of hidden training to safeguard multiple classes simultaneously.

Attack Aggressiveness. We also consider an attacker that choose to submit an attack instance to model $\mathcal{M}_i$ only if it succeeds on all prior models. We consider such attacks as cautious attacks. In our original experiments, we consider attacks that succeed on at least one previous models. When we examine the attack instances used by our original experiments, we find that more than 90% of the instances already succeed on all previous models.

Table VIII shows the result of comparing compound transferability with attacks selected using the above described two criteria. We see that the compound transferability of cautious attacks is slightly higher, especially at later model versions. However, the change is minor, especially when compared to [11], whose transferability is more than 90% beyond version 2.

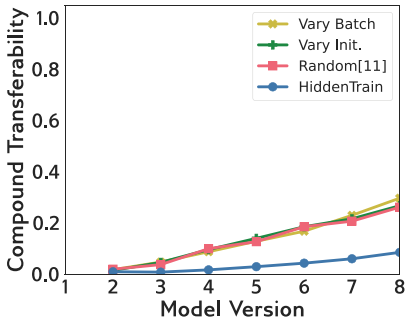

Fig. 15: *Success rate of CW-based compound transferability attacks under different versioning methods (*CIFAR10*)*

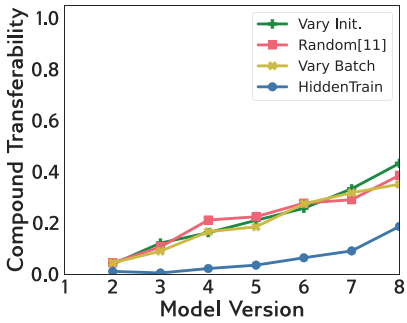

Fig. 16: *Success rate of EAD-based compound transferability attacks under different versioning methods (*CIFAR10*).*

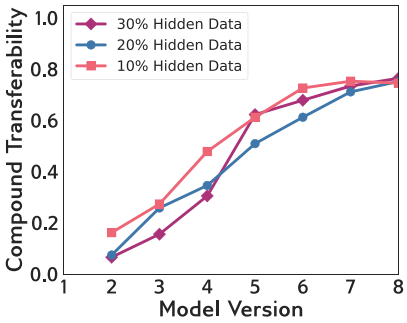

Fig. 17: *Success rate of PGD-based compound transferability attacks under different the portion of hidden data (*CIFAR10*).*

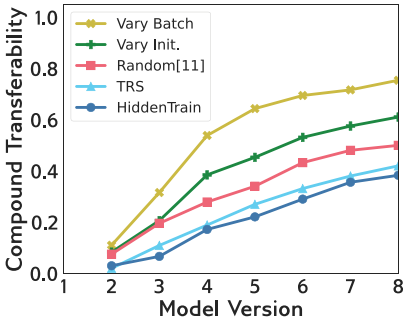

Fig. 18: *Success rate of PGD-based compound transferability attacks under different versioning methods (*CIFAR10*, VGG16 model architecture).*

Attack Instance Selection	2	3	4	5	6	7	8
Without Filter							
Succeed on at least 1 prior model	0.07	0.26	0.35	0.51	0.61	0.71	0.75
Succeed on all prior models	0.07	0.27	0.36	0.54	0.66	0.77	0.84
With Filter							
Succeed on at least 1 prior model *with* Filter	0.01	0.07	0.10	0.20	0.25	0.25	0.22
Succeed on all prior models *with* Filter	0.02	0.06	0.10	0.21	0.25	0.27	0.28

TABLE VIII: *Success rate of PGD-based compound transferability attacks from a "cautious" attacker, with and without run-time attack detection and filtering (*CIFAR10*).*

	Protect 1 class	Protect 3 classes
Random selection [11] *without* Filter	99.93%	99.38%
Hidden Training *without* Filter	75.22%	85.75%
Random selection [11] *with* Filter	56.14%	49.14%
Hidden Training *with* Filter	**22.5%**	**26.95%**

TABLE IX: *Success rate of PGD-based compound transferability attacks against the 8^{th} model version in the sequence, under different number of protected classes (* CIFAR10*).*