SSLGuard: A Watermarking Scheme for Self-supervised Learning Pre-trained Encoders

Tianshuo Cong[1,2] Xinlei He[2] Yang Zhang[2]

[1]*Tsinghua University* [2]*CISPA Helmholtz Center for Information Security*

Abstract

Self-supervised learning is an emerging machine learning (ML) paradigm. Compared to supervised learning which leverages high-quality labeled datasets to achieve good performance, self-supervised learning relies on unlabeled datasets to pre-train powerful encoders which can then be treated as feature extractors for various downstream tasks. The huge amount of data and computational resources consumption makes the encoders themselves become valuable intellectual property of the model owner. Recent research has shown that the ML model's copyright is threatened by *model stealing attacks*, which aim to train a surrogate model to mimic the behavior of a given model. We empirically show that pre-trained encoders are highly vulnerable to model stealing attacks. However, most of the current efforts of copyright protection algorithms such as watermarking concentrate on classifiers. Meanwhile, the intrinsic challenges of pre-trained encoder's copyright protection remain largely unstudied. We fill the gap by proposing *SSLGuard*, the first watermarking algorithm for pre-trained encoders. Given a clean pre-trained encoder, *SSLGuard* injects a watermark into it and outputs a watermarked version. The shadow training technique is also applied to preserve the watermark under potential model stealing attacks. Our extensive evaluation shows that *SSLGuard* is effective in watermark injection and verification, and is robust against model stealing and other watermark removal attacks such as input noising, output perturbing, overwriting, model pruning, and fine-tuning.[1]

1 Introduction

Deep learning (DL), in particular supervised deep learning models, has gained tremendous success during the past decade, and the development of supervised learning relies on a large amount of high-quality labeled data. However, high-quality data is often difficult to collect and the cost of labeling is expensive. *Self-supervised learning (SSL)* is proposed to resolve restrictions from lacking labeled data by generating "labels" from the unlabeled dataset (called *pre-training dataset*), and use the derived "labels" to pre-train an *encoder*. With huge amounts of unlabeled data and advanced model architectures, one can train a powerful encoder to learn informative representations (also referred to as embeddings) from the input data, which can be further leveraged as a feature extractor to train a *downstream classifier*. Such encoders pre-trained by SSL show great promise in various downstream tasks. For instance, on the ImageNet dataset [43], Chen et al. [13] show that, by using SimCLR pre-trained with ImageNet (unlabeled), the downstream classifier can achieve 85.8% top-5 accuracy with only 1% labels, which outperforms a supervised AlexNet but uses 100× fewer labels. He et al. [22] show that self-supervised learning can surpass supervised learning under seven downstream tasks including segmentation and detection.

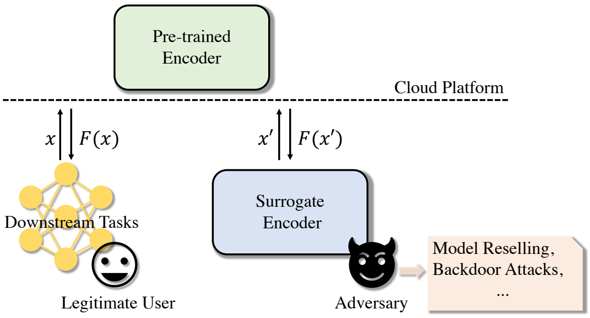


Figure 1: An illustration of deploying self-supervised learning pre-trained encoders as a service. The legitimate user aims to train downstream classifiers while the adversary tries to generate a surrogate encoder.

Compared to the supervised learning-based classifier which only suits a specific classification task, the SSL pre-trained encoder can achieve remarkable performance on different downstream tasks. However, the data collection and training process of SSL are also expensive as it benefits from larger datasets and more powerful computing devices. For instance, the performance of MoCo [22], one popular image encoder, pre-trained with the Instagram-1B dataset ($\sim$ 1 billion images) outperforms that of the encoder pre-trained with ImageNet-1M dataset (1.28 million images). SimCLR requires 32 TPU v3 cores to train a ResNet-50 encoder due to the large batch size setting (i.e., 4096) [13]. The cost to train a powerful encoder by SSL is usually prohibitive for individuals. Therefore, the high-performance encoders are usually pre-trained by leading AI companies with sufficient computing resources, and shared via cloud platforms for commercial

[1] Our code is available at https://github.com/tianshuocong/SSLGuard

usage. Nowadays, Encoder-as-a-Service (EaaS) [1, 2] is becoming popular. For instance, Clarifai [2] provides the image embedding model to generate representations of images for different downstream tasks and OpenAI provides access to GPT-3 [6] which can be considered as a powerful encoder for a variety of natural language processing (NLP) downstream tasks, such as code generation, style transfer, etc.

Once deployed on the cloud platform, the pre-trained encoders are not only accessible to legitimate users but also threatened by potential adversaries. As illustrated in Figure 1, for the legitimate user, the pre-trained encoder is used to train a downstream classifier. On the other hand, an adversary may perform *model stealing attacks* [30, 39, 46, 50] which aim to learn a surrogate encoder that has similar functionality as the pre-trained encoder published online. Such attacks may not only compromise the intellectual property of the service provider but also serve as a stepping stone for further attacks such as membership inference attacks [34,45,47] (i.e., mount membership inference attacks offline by using surrogate encoders), backdoor attacks [28] (i.e., publish another backdoored encoder), and adversarial examples [40]. The security and privacy of SSL encoders are threatened by these attacks, which call for effective defenses.

As one major technique to protect the copyright of a given model, model watermarking [26, 32] inserts a secret watermark pattern into the model. The ownership of the model can then be claimed if similar or exactly the same pattern is successfully extracted from the model. Recent studies on model watermarking mainly focus on the classifier that targeted a specific task [5, 26, 58]. However, compared to watermarking classifiers, watermarking SSL pre-trained encoders may face several intrinsic challenges. First, model watermarking against the classifier usually needs to specify a target class before being executed, while the SSL pre-trained encoder does not have such information. Second, downstream tasks for SSL pre-trained encoders are flexible, which challenges the traditional model watermarking scheme that is only suitable for one specific downstream task. Therefore, a new watermarking scheme should be designed to overcome those challenges to protect the copyright of SSL encoders. To the best of our knowledge, this has been left largely unstudied.

Our Work. In this paper, we first quantify the copyright breaching threat against SSL pre-trained encoders through the lens of model stealing attacks. Then, we introduce *SSLGuard*, the first watermarking algorithm for the SSL pre-trained encoders to protect their copyrights. Note that in this work, we consider image encoders only.

For model stealing attacks, we first assume that the adversary only has black-box access to the SSL pre-trained encoder (i.e., *victim* encoder). The adversary's goal is to build a surrogate encoder to "copy" the functionality of the victim encoder. We then characterize the adversary's background knowledge into two dimensions, i.e., the surrogate dataset's distribution and the surrogate encoder's architecture. Regarding the surrogate dataset which is used to train the surrogate encoder, we consider the adversary may or may not know the victim encoder's pre-training dataset. Regarding the surrogate encoder's architecture, we first assume that it shares the same architecture as the victim encoder. Then, we relax this assumption and find that the effectiveness of model stealing attacks can even increase by leveraging a larger model architecture. We empirically show that the model stealing attacks against victim encoders achieve remarkable performance. For instance, given a ResNet-50 encoder pre-trained on ImageNet by SimCLR, the ResNet-101 surrogate encoder can achieve 0.944 accuracy on STL-10 and 0.805 accuracy on CIFAR-10, while the accuracy for the victim encoder is 0.948 on STL-10 and 0.855 on CIFAR-10, respectively. This is because the rich information hidden in the embeddings can better help the surrogate encoder mimic the behavior of the victim encoder. We also show that the cost of stealing an encoder is much smaller than pre-training it from scratch, e.g., pre-training the BYOL ResNet-50 encoder costs $5,713.92 while stealing it with ResNet-101 only costs $72.49 (see Table 3 for the detailed comparison). Such observation emphasizes the underlying threat of jeopardizing the model owner's intellectual property and the emergence of copyright protection.

To protect the copyright of SSL pre-trained encoders, we propose *SSLGuard*, a robust *black-box* watermarking algorithm for SSL pre-trained encoders. Concretely, given a secret vector, the goal of *SSLGuard* is to inject a watermark based on the secret vector into a clean SSL pre-trained encoder. The output of *SSLGuard* contains a watermarked encoder and a key-tuple. To be specific, the key-tuple consists of a secret vector, a verification dataset, and a decoder. *SSLGuard* fine-tunes a clean encoder to a watermarked encoder. The watermarked encoder can preserve the utility of the clean encoder and map samples in the verification dataset to secret representations. We further introduce a decoder to transform these secret representations into the secret vector which may lie in another space. For other encoders, the decoder only transforms the representations generated from the verification dataset into random vectors. Recent research has shown that if a watermarked model is stolen, its corresponding watermark usually vanishes [37]. To remedy this situation, *SSLGuard* adopts a shadow dataset and a shadow encoder to locally simulate the process of model stealing attacks. In the training process, *SSLGuard* optimizes a trigger that can be recognized by both the watermarked encoder and the shadow encoder. We later show in Section 5 that such a design can strongly preserve the watermark even in the surrogate encoder stolen by the adversary.

Empirical evaluations over 7 datasets (i.e., ImageNet, CIFAR-10, CIFAR-100, STL-10, GTSRB, MNIST, and FashionMNIST) and 3 encoder pre-training algorithms (i.e., SimCLR, MoCo v2, and BYOL) show that *SSLGuard* can successfully inject/extract the watermark to/from the SSL pre-trained encoder without sacrificing its performance and is robust to model stealing attacks. Moreover, we consider various types of watermark removal attacks including input preprocessing (noising), output perturbing (noising and truncation), and model modification (overwriting, pruning, and fine-tuning) to "clean" the model. We empirically show that *SSLGuard* is still effective in such a scenario.

In summary, we make the following contributions:

- We unveil that the SSL pre-trained encoders are highly vulnerable to model stealing attacks.

- We propose *SSLGuard*, the first watermarking algorithm against SSL pre-trained encoders, which is able to protect the intellectual property of published encoders.

- Extensive evaluations show that *SSLGuard* is effective in injecting and extracting watermarks and robust against model stealing and other watermark removal attacks such as input noising, output perturbing, overwriting, model pruning, and fine-tuning.

2 Background

2.1 Self-supervised Learning

Self-supervised learning is a rising AI paradigm that aims to train an encoder by a large scale of unlabeled data. A high-performance pre-trained encoder can be shared into the public platform as an upstream service. In downstream tasks, customers can use the embeddings output from the pre-trained encoder to train their classifiers with limited labeled data [13] or even no data [41]. One of the most remarkable self-supervised learning paradigms is contrastive learning [13, 15, 21, 22, 41]. In general, encoders are pre-trained through contrastive losses which calculate the similarities of embeddings in a latent space. In this paper, we consider three representative contrastive learning algorithms, i.e., SimCLR [13], MoCo v2 [15], and BYOL [41].

SimCLR [13]. SimCLR is a simple framework for contrastive learning. It consists of four components, including *Data augmentation*, *Base encoder* $f(\cdot)$, *Projection head* $g(\cdot)$ and *Contrastive loss function*.

The data augmentation module is used to transform a data sample x randomly into two augmented views. Specifically, the augmentations include random cropping, random color distortions, and random Gaussian blur. If two augmented views are generated from the same data sample x, we treat them as a positive pair, otherwise, they are considered as a negative pair. Positive pairs of x are denoted as $\tilde{x_i}$ and $\tilde{x_j}$.

Base encoder $f(\cdot)$ extracts feature vectors $h_i = f(\tilde{x_i})$ from augmented inputs. Projection head $g(\cdot)$ is a small neural network that maps feature vectors to a latent space where contrastive loss is applied. SimCLR uses a multilayer perceptron (MLP) as the projection head $g(\cdot)$ to obtain the output $z_i = g(h_i)$.

For a set of samples $\{\tilde{x_k}\}$ including both positive and negative pairs, contrastive loss aims to maximize the similarity between the feature vectors of positive pairs and minimize those of negative pairs. Given N samples in each mini-batch, we could get $2N$ augmented samples. Formally, the loss function for a positive pair $\tilde{x_i}$ and $\tilde{x_j}$ can be formulated as:

$$l(i,j) = -\log \frac{\exp(\mathrm{sim}(z_i,z_j)/\tau)}{\sum_{k=1,k\neq i}^{2N} \exp(\mathrm{sim}(z_i,z_k)/\tau)},$$

where $\mathrm{sim}(\cdot,\cdot)$ denotes the cosine similarity function and τ denotes a temperature parameter. SimCLR jointly trains the base encoder and projection head by minimizing the final loss function:

$$\mathcal{L}_{SimCLR} = \frac{1}{2N} \sum_{k=1}^{N} [l(2k-1,2k) + l(2k,2k-1)],$$

where $2k-1$ and $2k$ are the indexes for each positive pair.

Once the model is trained, SimCLR discards the projection head and keeps the base encoder $f(\cdot)$ only, which serves as the pre-trained encoder.

MoCo v2 [15]. Momentum Contrast (MoCo) [22] is a famous contrastive learning algorithm, and MoCo v2 is the modified version (using projection head and more data augmentations).

MoCo points out that contrastive learning can be regarded as a dictionary lookup task. The "keys" in the dictionary are the embeddings output from the encoder. A "query" matches a key if they are encoded from the same image. MoCo aims to train an encoder that outputs similar embeddings for a query and its matching key, and dissimilar embeddings for others. The dictionary is desirable to be large and consistent, which contains rich negative images and helps to learn good representations. MoCo aims to build such a dictionary with a queue and momentum encoder.

MoCo contains two parts: *query encoder* $f_q(x;\theta_q)$ and *key encoder* $f_k(x;\theta_k)$. Given a query sample x^q, MoCo gets an encoded query $q = f_q(x^q)$. For other samples x^k, MoCo builds a dictionary whose keys are $\{k_0, k_1, ...\}$, $k_i = f_k(x_i^k)$, $i = 0,1,....$ The dictionary is a dynamic queue that keeps the current mini-batch encoded embeddings and discards ones in the oldest mini-batch. The benefit of using a queue is decoupling the dictionary size from the mini-batch size, so the dictionary size can be set as a hyper-parameter. Assume k_+ is the key that q matches, the loss function will be defined as:

$$\mathcal{L}_{MoCo} = -\log \frac{\exp(q \cdot k_+/\tau)}{\sum_{i=0}^{K} \exp(q \cdot k_i/\tau)}.$$

τ is a temperature hyper-parameter. MoCo trains f_q by minimizing contrastive loss and updates θ_q by gradient descent. However, it is difficult to update θ_k by back-propagation because of the queue, so f_k is updated by moving-averaged as:

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q,$$

where $m \in [0,1)$ denotes a momentum coefficient. Finally, we keep the f_q as the final pre-trained encoder.

BYOL [21]. Bootstrap Your Own Latent (BYOL) is a novel self-supervised learning algorithm. Different from previous methods, BYOL does not rely on the negative pairs, and it has a more robust selection of image augmentations.

BYOL's architecture consists of two neural networks: *online networks* and *target networks*. The online networks, with parameters θ, consist of an encoder f_θ, a projector g_θ and a predictor q_θ. The target networks are made up of an encoder f_ξ and a projector g_ξ. The two networks bootstrap the representations and learn from each other.

Given an input sample x, BYOL produces two augmented views $v \leftarrow t(x)$ and $v' \leftarrow t'(x)$ by using image augmentations

t and t', respectively. The online networks output a projection $z_\theta \leftarrow g_\theta(f_\theta(v))$ and target networks output a target projection $z'_\xi \leftarrow g_\xi(f_\xi(v'))$. The online networks' goal is to make prediction $q_\theta(z_\theta)$ similar to z'_ξ. Formally, the similarity can be defined as the following:

$$L_{\theta,\xi} = 2 - 2 \cdot \frac{\langle q_\theta(z_\theta), z'_\xi \rangle}{\|q_\theta(z_\theta)\|_2 \cdot \|z'_\xi\|_2}.$$

Conversely, BYOL feeds v' to online networks and v to target networks separately and gets $\widetilde{L_{\theta,\xi}}$. The final loss function can be formulated as:

$$\mathcal{L}_{BYOL} = L_{\theta,\xi} + \widetilde{L_{\theta,\xi}}.$$

BYOL updates the weights of the online and target networks by:

$$\theta \leftarrow \text{optimizer}(\theta, \triangledown_\theta L^{BYOL}_{\theta,\xi}, \eta),$$

$$\xi \leftarrow \tau\xi + (1-\tau)\theta,$$

where η is a learning rate of the online networks. The target networks' weight ξ is updated in a weighted average way, and $\tau \in [0,1]$ denotes the decay rate of the target encoder. Once the model is trained, we treat the online networks' encoder f_θ as the pre-trained encoder.

2.2 Model Stealing Attacks

Model stealing attacks [10,11,17,25,30,39,46,50,53] aim to steal the parameters or the functionality of the victim model. To achieve this goal, given a victim model $f(x;\theta)$, the adversary can issue a bunch of queries to the victim model and obtain the corresponding responses. Then the queries and responses serve as the inputs and "labels" to train the surrogate model, denoted as $f'(x;\theta')$. Formally, given a query dataset $\mathcal{D}$, the adversary can train $f'(x;\theta')$ by

$$\mathcal{L}_{steal} = \mathbb{E}_{x\sim\mathcal{D}}[\text{sim}(f(x;\theta), f'(x;\theta'))]. \tag{1}$$

where $\text{sim}(\cdot,\cdot)$ is a similarity function.

Note that if the victim model is a classifier, the response can be the prediction probability of each class. If the victim model is an encoder, the response can be the embeddings. A successful model stealing attack may not only breach the intellectual property of the victim model but also serve as a springboard for further attacks such as membership inference attacks [23, 24, 33, 34, 45, 47, 48], backdoor attacks [14, 28, 44, 56] and adversarial examples [9, 20, 31, 36, 40]. Previous work has demonstrated that neural networks are vulnerable to model stealing attacks. In this paper, we concentrate on model stealing attacks on SSL pre-trained encoders, which have not been studied yet.

2.3 DNNs Watermarking

Considering the cost of training deep neural networks (DNNs), DNNs watermarking algorithms have received wide attention as it is an effective method to protect the copyright of the DNNs. Watermarking is a traditional concept for media such as audio and video, and it has been extended to protect the intellectual property of deep learning models recently [5,26,38,42,51]. Concretely, the watermarking procedure can be divided into two steps, i.e., injection and verification. In the injection step, the model owner injects a watermark and a pre-defined behavior into the model in the training process. The watermark is usually secret, such as a trigger that is only known to the model owner [32]. In the verification step, the ownership of a suspect model can be claimed if the watermarked encoder has the pre-defined behavior when the input samples contain the trigger.

So far, the watermarking algorithms mainly focus on the classifiers in a specific task. However, how to design a watermarking algorithm for SSL pre-trained encoders that can fit various downstream tasks remains largely unexplored.

3 Threat model

In this paper, we consider two parties: the *defender* and the *adversary*. The defender is the owner of the victim encoder, whose goal is to protect the copyright of the victim encoder when publishing it as an online service. The adversary (also referred to as attacker), on the contrary, aims to steal the victim encoder, i.e., by model stealing attacks or directly obtaining the model (insider threat), and bypass the copyright protection method for the victim encoder.

Adversary's Motivation. Adversary's motivation lies in two areas: Firstly, EaaS is being popular and high-performance SSL encoders are often pre-trained by top AI companies [1, 2]. Pre-training an encoder requires collecting a huge amount of data, expert knowledge for designing architectures/algorithms, and many failure trials, which are expensive. This makes the model architectures or training algorithms be regarded as trade secrets and will not be publicly available, which makes it less possible for the adversary to directly train a comparable performance SSL encoder from scratch. Secondly, the cost of stealing an SSL encoder is quite less than training an SSL encoder from scratch. For instance, pre-training a ResNet-50 by BYOL needs $5,713.92 while generating a surrogate encoder with similar performance only needs $72.49 (see Table 3 for more details). Once the adversary steals the victim encoder successfully, they can resell it or deploy it on the cloud platform to be a commercial competitor.

Adversary's Background Knowledge. For the adversary, we first assume that they only have the black-box access to the victim encoder, which is the most challenging setting for the adversary [25,30,39,46]. In this setting, the adversary can only query the victim encoder with data samples and obtain their corresponding responses, i.e., the embeddings. Then, data samples and the corresponding responses are used to train the surrogate encoders. We categorize the adversary's background knowledge into two dimensions, i.e., the pre-training dataset and the victim encoder's architecture. Concretely, we assume that the adversary has a query dataset to perform the attack. Note that the query dataset does not need to be in the same distribution as the victim encoder's pre-training dataset. Regarding the victim encoder's architecture, we first assume that the adversary can obtain it since such information is usually publicly accessible. Then we empirically show that this assumption can be relaxed, and the attack is even more effective when the surrogate encoder leverages a deeper model architecture.

Adaptive Adversary. We then consider an adaptive adversary who knows that the victim encoder has already been watermarked. This means they can leverage watermark removal techniques including input preprocessing (noising), output perturbing (noising and truncation), and model modification (overwriting, pruning, and fine-tuning) on the victim encoder to bypass the watermark verification.

4 Design of Watermarking Algorithm

In this section, we present *SSLGuard*, a watermarking scheme to preserve the copyright of the SSL pre-trained encoders. *SSLGuard* should have the following properties:

- **Fidelity:** To minimize the impact of *SSLGuard* on the legitimate user, the influence of *SSLGuard* on clean pre-trained encoders should be negligible, which means *SSLGuard* should keep the utility of downstream tasks.
- **Effectiveness:** *SSLGuard* should judge whether a suspect model is a watermarked (or a clean) model with high precision. In other words, *SSLGuard* should extract watermarks from watermarked encoders effectively.
- **Undetectability:** The watermark cannot be extracted by a *no-matching* secret key-tuple. Undetectability ensures that ownership of the SSL pre-trained encoder could not be misrepresented.
- **Efficiency:** *SSLGuard* should inject and extract watermark efficiently. For instance, the time cost for the watermark injecting and extracting process should be less than pre-training an SSL model.
- **Robustness:** *SSLGuard* should be robust against model stealing attacks and other watermark removal attacks such as input noising, output perturbing, overwriting, model pruning, and fine-tuning.

In the following subsections, we will introduce the design methods for *SSLGuard*. Table 1 summarizes the notations used in this paper.

4.1 Overview

As shown by Cai et al. [7], in space $\mathbb{R}^n$, given two random vectors which are independently chosen with the uniform distribution on the unit sphere, the empirical distribution of angles θ between these two random vectors converges to a distribution with the following probability density function:

$$f(\theta) = \frac{1}{\sqrt{\pi}} \cdot \frac{\Gamma(\frac{n}{2})}{\Gamma(\frac{n-1}{2})} \cdot (\sin\theta)^{n-2}, \theta \in [0, \pi].$$

The distribution $f(\theta)$ will be very close to normal distribution if $n \geq 5$. The equation above implies that two random vectors in high-dimensional space (such as $\mathbb{R}^{256}$) are almost *orthogonal*. The inspiration for *SSLGuard* is based on the above mathematical fact: Given a vector that has the same dimension as embeddings, if the vector is randomly initialized, the average cosine similarity between these embeddings and the vector should be concentrated around zero. However, if the average cosine similarity is much bigger than 0 or even close to 1, this can be considered as a signal that those embeddings are strongly related to this vector. Therefore, the defender can generate a verification dataset $\mathcal{D}_v$ and a secret vector $sk \in \mathbb{R}^m$. Then, the defender can fine-tune a clean encoder to transform samples from $\mathcal{D}_v$ to the embeddings and train a decoder to further transform the embeddings to the decoded vectors that have high cosine similarity with sk. Meanwhile, if the defender input these verification samples to a clean encoder, the distribution of cosine similarity between decoded vectors and sk should be a normal distribution with 0 as its mean value. We leverage this mechanism to design *SSLGuard*.

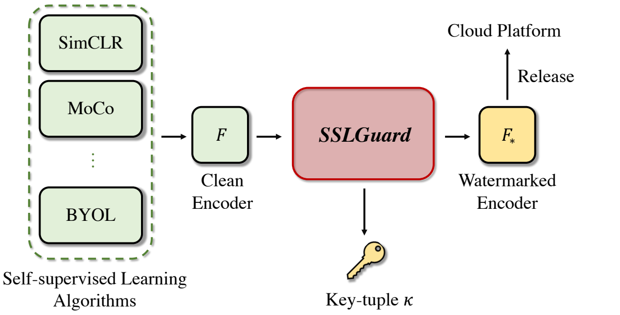


Figure 2: The workflow of *SSLGuard*. Given a clean SSL pre-trained encoder (colored in green), *SSLGuard* outputs a key-tuple and a watermarked encoder (colored in yellow). The defender can employ the watermarked encoder on the cloud platform or adopt the key-tuple to extract the watermark from a suspect encoder.

The workflow of *SSLGuard* is shown in Figure 2. Concretely, given a clean encoder F which is pre-trained by a certain SSL algorithm, *SSLGuard* will output a watermarked encoder F_* and a secret key-tuple κ as:

$$F_*, \kappa \leftarrow SSLGuard(F),$$
$$\kappa = \{\mathcal{D}_v, G, sk\}.$$

The secret key-tuple κ consists of three items: a verification dataset $\mathcal{D}_v$, a decoder G, and a secret vector sk. G is an MLP that maps the embeddings generated from the encoder to a new latent space (same dimension as sk) to calculate the cosine similarity with sk. Concretely, given an input image x, the decoded vector sk'_x can be defined as:

$$sk'_x = G(E(x)), x \in \mathcal{D},$$

where $sk'_x \in \mathbb{R}^m$ is a vector whose dimension is the same as the secret vector sk and $\mathcal{D}$ is a given dataset, and E is an encoder (i.e., F or F_*, etc).

SSLGuard contains two processes, i.e., watermark injection and extraction. For the injection process, *SSLGuard* uses a secret key-tuple κ to inject the watermark into a clean en-

Table 1: List of notations.

Notation	Description
F, F_*, F_s	Clean/Watermarked/Shadow encoder
$\mathcal{D}_t$, $\mathcal{D}_s$	Target/Shadow dataset
$\mathcal{D}_p$, $\mathcal{D}_v$	Private/Verification dataset
T, M	Trigger, Mask
κ, G	Key-tuple, Decoder
sk, sk'_x	Secret vector, Decoded vector
DA	Downstream accuracy
WR	Watermark rate

coder F and outputs watermarked encoder F_* as:

$$F_* \leftarrow \texttt{Inject}(F, \kappa).$$

The defender can release F_* to the cloud platform and keep κ secret.

For the extraction process, given a suspect encoder F', the defender can use κ to extract decoded vectors sk'_x from F' by:

$$\{sk'_x\} \leftarrow \texttt{Extract}(F', \kappa), x \in \mathcal{D}_v,$$

where $\{sk'_x\}$ is a set of decoded vectors. Then, the defender can measure the cosine similarity between $\{sk'_x\}$ and sk, and judge if a suspect encoder F' is a copy by:

$$\texttt{Verify}(F') = \begin{cases} 1, & \text{WR} > th_v \\ 0, & \text{otherwise} \end{cases},$$

here we adopt watermark rate (WR) as the metric to denote the ratio of the verified samples whose outputs sk'_x are close to sk. Concretely, WR is defined as:

$$\text{WR} = \frac{1}{|D_v|} \sum_{x \in D_v} \mathbb{1}(\text{sim}(sk'_x, sk) > th_w).$$

In summary, we need two thresholds here: th_v and th_w. th_w is used to calculate WR, and th_v is a threshold to verify the copyright. We set $th_w = 0.5$ and $th_v = 0.5$ by default. Note that the th_w can be set to a smaller value as we show in Section 5 that the WR is 0 for the clean encoders. The overview of *SSLGuard* is depicted in Figure 3. Concretely, we first train a watermarked encoder that contains the information of the verification dataset and the secret vector. The clean encoder serves as a query-based API to guide the training process. The shadow encoder is used to simulate the model stealing process to better preserve the watermark under model stealing attacks. The watermarked encoder should keep the utility of the clean encoder while preserving the watermark injected in it.

4.2 Preparation

To watermark a pre-trained encoder, the defender should prepare a private dataset $\mathcal{D}_p$, a mask M, and a random trigger T. The mask M is a binary matrix that contains the position information of trigger T, which means M and T have the same size as x_p. Following [18, 56], we inject the trigger into private samples x_p by:

$$\mathcal{P}(x_p, T) = (1 - M) \circ x_p + M \circ T, x_p \in \mathcal{D}_p,$$

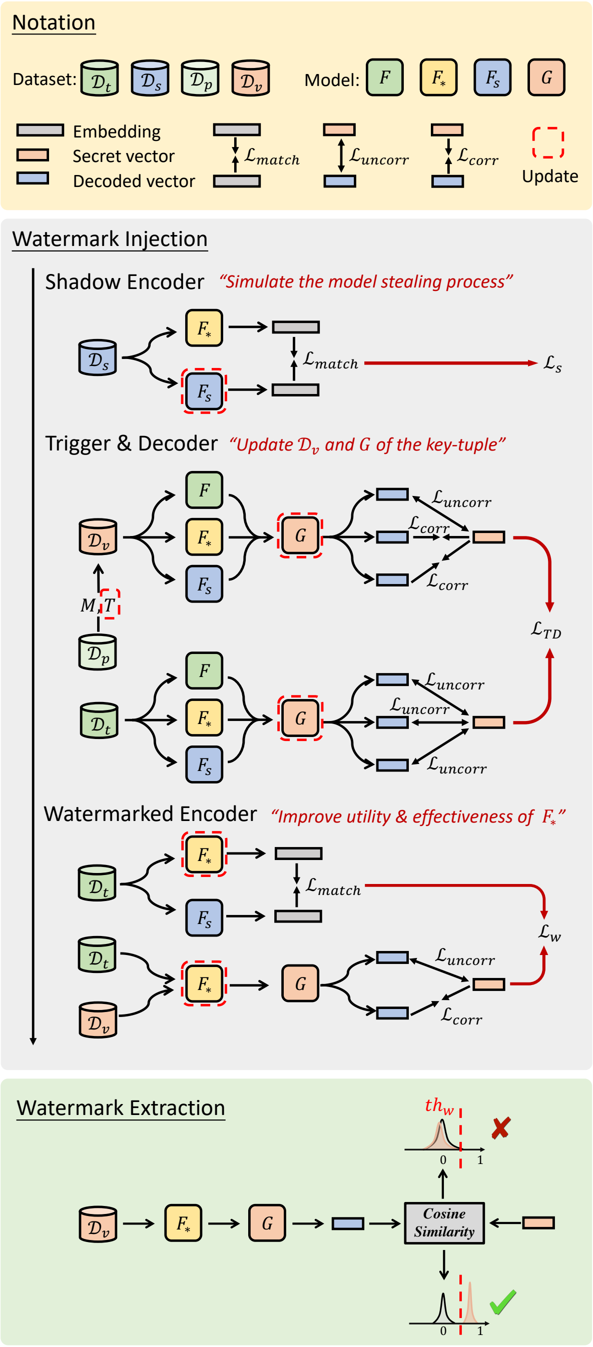


Figure 3: The overview of *SSLGuard*.

where $\circ$ denotes the element-wise product.

Given the trigger T, we can generate the verification dataset as:

$$\mathcal{D}_v = \{x_v | x_v = \mathcal{P}(x_p, T), x_p \in \mathcal{D}_p\}.$$

Here we define three loss functions, i.e., *correlated loss* $\mathcal{L}_{corr}$, *uncorrelated loss* $\mathcal{L}_{uncorr}$, and *embedding match loss* $\mathcal{L}_{match}$ to achieve three goals. Our first goal is to let the decoded vectors transformed from the verification dataset $\mathcal{D}_v$ to be similar as the secret vector sk, and we define *correlated* *loss function* as:

$$\mathcal{L}_{corr}(\mathcal{D}_v, E) = \frac{-\sum_{x\sim\mathcal{D}_v} \mathrm{sim}(sk'_x, sk)}{|\mathcal{D}_v|}, \quad (2)$$

where $\mathrm{sim}(\cdot,\cdot)$ is a similarity function. If not otherwise specified, we use cosine similarity as the similarity function. The goal of $\mathcal{L}_{corr}$ is to train an encoder and an decoder together to transform x into sk'_x, where sk'_x is correlated with sk. The more similar sk'_x and sk are, the smaller $\mathcal{L}_{corr}$ will be.

Secondly, given a clean dataset $\mathcal{D}$ and an encoder E, the decoder G transforms decoded vectors to the orthogonal direction of sk for uncorrelated samples $x \in \mathcal{D}$. In other words, *SSLGuard* will adopt $\mathcal{L}_{uncorr}$ to the decoded keys that should not be encoded in the similar direction of sk. Therefore, we could get another loss function, *uncorrelated loss function*, as:

$$\mathcal{L}_{uncorr}(\mathcal{D}, E) = (\frac{\sum_{x\sim\mathcal{D}} \mathrm{sim}(sk'_x, sk)}{|\mathcal{D}|})^2. \quad (3)$$

Finally, we here define an *embedding match loss function* to match the embeddings generated from two encoders E' and E'':

$$\mathcal{L}_{match}(\mathcal{D}, E', E'') = \frac{-\sum_{x\sim\mathcal{D}} \mathrm{sim}(E'(x), E''(x))}{|\mathcal{D}|}. \quad (4)$$

SSLGuard leverages $\mathcal{L}_{match}$ to maintain the utility of the watermarked encoder and simulate the model stealing attacks.

4.3 Watermark Injecting

As shown in Figure 3, *SSLGuard* adopts three encoders: a clean encoder $F(x;\theta)$, a watermarked encoder $F_*(x;\theta_w)$ and a shadow encoder $F_s(x;\theta_s)$. Meanwhile, *SSLGuard* also uses three datasets: a target dataset $\mathcal{D}_t$, a shadow dataset $\mathcal{D}_s$, and a verification dataset $\mathcal{D}_v$. In the following part, we will introduce our loss functions for each module.

Shadow Encoder. For the shadow encoder, its task is to mimic the model stealing attacks. Here we use $\mathcal{D}_s$ to simulate the query process. The loss function of shadow encoder is:

$$\mathcal{L}_s = \mathcal{L}_{match}(\mathcal{D}_s, F_*, F_s). \quad (5)$$

Trigger and Decoder. Given a verification dataset, we aim to optimize a trigger T and a decoder G to extract sk from both the watermarked encoder and the shadow encoder, but not the clean encoder. The corresponding loss can be defined as:

$$\mathcal{L}_1 = \mathcal{L}_{uncorr}(\mathcal{D}_v, F) + \mathcal{L}_{corr}(\mathcal{D}_v, F_*) + \mathcal{L}_{corr}(\mathcal{D}_v, F_s). \quad (6)$$

Besides, for the clean encoder F, watermarked encoder F_*, and the shadow encoder F_s, the decoder should not map the decoded keys closely to sk from the target dataset, the loss to achieve this goal can be defined as:

$$\mathcal{L}_2 = \mathcal{L}_{uncorr}(\mathcal{D}_t, F) + \mathcal{L}_{uncorr}(\mathcal{D}_t, F_*) + \mathcal{L}_{uncorr}(\mathcal{D}_t, F_s). \quad (7)$$

Given the above losses, the final loss function for trigger and decoder can be defined as:

$$\mathcal{L}_{TD} = \mathcal{L}_1 + \mathcal{L}_2. \quad (8)$$

Watermarked Encoder. For the watermarked encoder, we want it to keep the utility of the clean encoder. Therefore, for the samples from $\mathcal{D}_t$, we force the embeddings from F and F_* to become similar through $\mathcal{L}_{match}$. The loss $\mathcal{L}_3$ can be defined as:

$$\mathcal{L}_3 = \mathcal{L}_{match}(\mathcal{D}_t, F, F_*). \quad (9)$$

Meanwhile, the decoder G should successfully extract sk from the verification dataset $\mathcal{D}_v$ instead of the target dataset $\mathcal{D}_t$. The corresponding loss $\mathcal{L}_1$ to achieve this goal is defined as:

$$\mathcal{L}_4 = \mathcal{L}_{uncorr}(\mathcal{D}_t, F_*) + \mathcal{L}_{corr}(\mathcal{D}_v, F_*). \quad (10)$$

The final loss function for the watermarked encoder is:

$$\mathcal{L}_w = \mathcal{L}_3 + \mathcal{L}_4. \quad (11)$$

Optimization Problem. After designing all loss functions, we formulate *SSLGuard* as an optimization problem. Concretely, we update the parameters as follows:

$$\begin{aligned} \theta_s &\leftarrow \mathrm{Optimizer}(\theta_s, \bigtriangledown_{\theta_s}\mathcal{L}_s, \eta_s), \\ T, G &\leftarrow \mathrm{Optimizer}(T, G, \bigtriangledown_{T,G}\mathcal{L}_{TD}, \eta_{TD}), \\ \theta_w &\leftarrow \mathrm{Optimizer}(\theta_w, \bigtriangledown_{\theta_w}\mathcal{L}_w, \eta_w), \end{aligned} \quad (12)$$

where η_s, η_{TD}, and η_w are learning rates of shadow encoder, watermarked encoder, trigger, and decoder, respectively. We note that we update θ_s, T, G, and θ_w sequentially in one iteration, and we stop the optimization until the iteration reaches the max iteration number.

5 Evaluation

5.1 Experimental Setup

Datasets. We use the following 7 datasets to conduct our experiments.

- **ImageNet [43].** The ImageNet dataset contains 1.2 million training images distributed in 1,000 classes. Each sample has size $224 \times 224 \times 3$.
- **CIFAR-10 [3]** The CIFAR-10 dataset has $60,000$ images in 10 classes. Among them, there are $50,000$ images for training and $10,000$ images for testing. The size of each sample is $32 \times 32 \times 3$.
- **CIFAR-100 [3].** Similar to CIFAR-10, The CIFAR-100 dataset contains $60,000$ images with size $32 \times 32 \times 3$ in 100 classes, and there are 500 training images and 100 testing images in each class.

- **STL-10 [16].** The STL-10 dataset consists of 5,000 training samples and 8,000 test samples in 10 classes. Besides, it also contains 100,000 unlabeled samples. Note that the images on STL-10 are acquired from labeled samples on ImageNet.[2] The size of each sample is $96 \times 96 \times 3$.
- **GTSRB [49].** German Traffic Sign Recognition Benchmark (GTSRB) contains 39,209 training images and 12,630 test images. It contains 43-category traffic signs.
- **MNIST [4].** MNIST is a handwritten digits dataset which contains 60,000 training examples and 10,000 test examples in 10 classes. Each sample has size $28 \times 28 \times 1$.
- **FashionMNIST [55].** FashionMNIST (F-MNIST) is a Zalando's article image dataset that has 10 classes. Is has 60,000 training examples and 10,000 test examples. Each example is a grayscale image with size $28 \times 28 \times 1$.

We resize samples in all datasets to $224 \times 224 \times 3$ in the experiment. We use ImageNet as the pre-training dataset; STL-10, CIFAR-10, F-MNIST, and MNIST as the downstream datasets; and STL-10, CIFAR-10, CIFAR-100, and GTSRB as the query dataset. Note that for the STL-10 dataset, we randomly split the unlabeled samples (100,000) of it into two parts (each containing 50,000 samples). We consider the first part as the unlabeled STL-10 dataset and the second part as the same distribution unlabeled STL-10 dataset which is denoted as STL-10 (s).

Pre-training Encoder. In our experiment, we adopt real-world contrastive learning pre-trained encoders as the victim encoders. Concretely, we download the checkpoints of the encoders from the official website (i.e., SimCLR[3] and MoCo v2[4]) or the public platform (i.e., BYOL[5]). All the encoders are ResNet-50 pre-trained on ImageNet.

Downstream Classifier. We use a 3-layer MLP as the downstream classifier with 512 and 256 neurons in its hidden layer. For each downstream task, we freeze the parameters of the pre-trained encoders and train the downstream classifier for 20 epochs using Adam optimizer [29] with 0.005 learning rate.

SSLGuard. We reload the clean encoder and fine-tune it to be the watermarked encoder. Note that we freeze the weights in batch normalization layers following the settings by Jia et al. [28]. We consider the unlabeled STL-10 dataset (with only 50,000 images as mentioned above) as both $\mathcal{D}_s$ and $\mathcal{D}_t$, and adopt a ResNet-50 as the shadow encoder's architecture. We sample 100 images from 5 random classes on ImageNet as our $\mathcal{D}_p$. Note that each class contains 20 images and the $\mathcal{D}_p$ for watermarking SimCLR, MoCo v2, and BYOL are non-overlapping. For each sample in $\mathcal{D}_p$, 35% space will be patterned by the trigger.

We leverage the SGD optimizer with 0.01 learning rate to train both the watermarked encoder and shadow encoder for 50 epochs. The batch size in our experiment is 8. The dimension of sk is 256. For the trigger, we randomly generate a $224 \times 224 \times 3$ tensor from uniform distribution in $[0,1]$ as the initial trigger. We use a 3-layer MLP as the decoder G. The numbers of G's neurons are 512, 256, and 256, respectively. We use the SGD optimizer with 0.005 learning rate to update both the decoder and the trigger.

5.2 Clean Downstream Accuracy

Given three *clean* SSL pre-trained encoders (i.e., pre-trained by SimCLR, MoCo v2, and BYOL on ImageNet), we first measure their downstream accuracy, denoted as *clean downstream accuracy (CDA)*, for different tasks. We consider four downstream classification tasks, i.e., STL-10, CIFAR-10, MNIST, and F-MNIST. The CDA are shown in Table 2. We observe that the SSL pre-trained encoders can achieve remarkable performance on different downstream tasks, which means the SSL pre-trained encoders can learn high-level semantic information from one task (i.e. ImageNet), and the informative embeddings can generalize to other tasks (i.e., STL-10 and CIFAR-10). Meanwhile, the cost of pre-training SSL encoders is expensive (see Table 3), such observation further demonstrates the necessity of protecting the copyright of the SSL pre-trained encoders. Note that we adopt CDA as our baseline accuracy. Later we measure an encoder's performance by comparing its DA with CDA.

Table 2: Clean downstream accuracy (CDA).

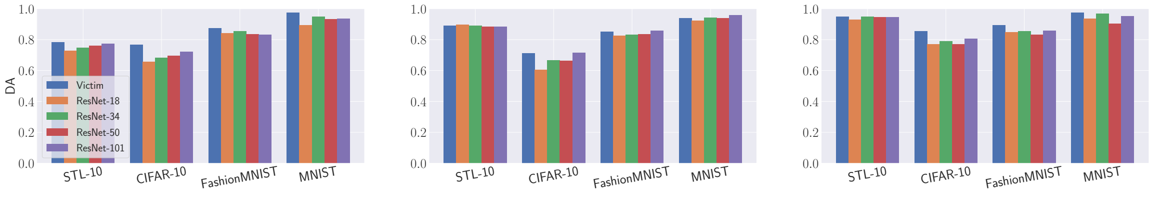
Downstream Task	SimCLR	MoCo v2	BYOL
STL-10	0.783	0.889	0.948
CIFAR-10	0.766	0.712	0.855
MNIST	0.974	0.940	0.974
F-MNIST	0.874	0.852	0.894

5.3 Model Stealing Attacks

Since the SSL pre-trained encoders (clean encoders) are powerful, we then evaluate whether they are vulnerable to model stealing attacks. To build a surrogate encoder, we consider three key information, i.e., the surrogate encoder's architecture, the distribution of the query dataset, and the similarity function used to "copy" the victim encoder.

Surrogate Encoder's Architecture. We first investigate the impact of the surrogate encoder's architecture. Note that here we adopt the unlabeled STL-10 dataset (with 50,000 unlabeled samples) as the query dataset and cosine similarity as the similarity function to measure the difference between the victim and surrogate encoders' embeddings. Since the architecture of the victim encoder can be non-public, the attacker may try different surrogate encoder architectures to perform the model stealing attack. Concretely, we assume attackers may leverage ResNet-18, ResNet-34, ResNet-50, and ResNet-101 as the surrogate encoder's architecture. If the output dimension is different from ResNet-50,

[2] https://cs.stanford.edu/~acoates/stl10/
[3] https://github.com/google-research/simclr
[4] https://github.com/facebookresearch/moco
[5] https://github.com/yaox12/BYOL-PyTorch

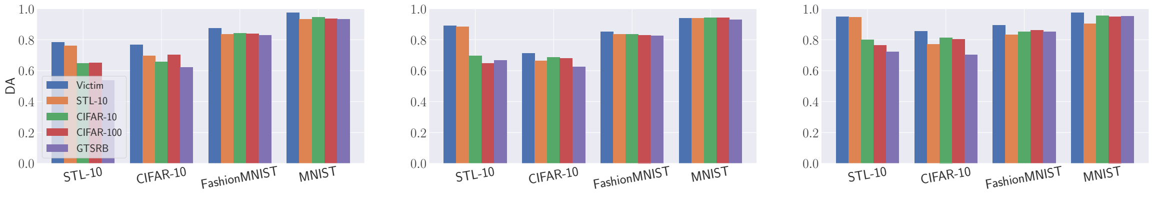


(a) SimCLR

(b) MoCo v2

(c) BYOL

Figure 4: The performance of surrogate encoders trained with different architectures.

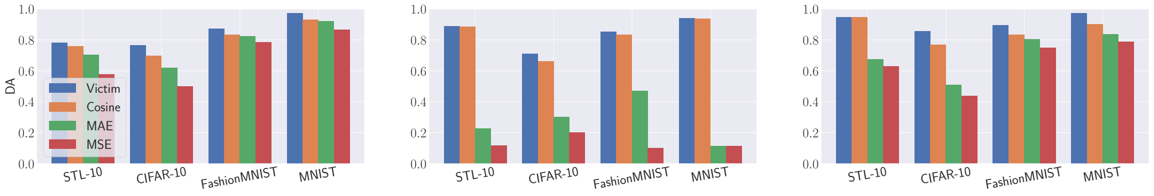


(a) SimCLR

(b) MoCo v2

(c) BYOL

Figure 5: The performance of surrogate encoders trained with different query datasets.



(a) SimCLR

(b) MoCo v2

(c) BYOL

Figure 6: The performance of surrogate encoders trained with different loss functions.

e.g., ResNet-18/ResNet-34 outputs 512-dimensional embeddings, we leverage an extra linear layer to transform them into 2048-dimension. The DA of surrogate encoders is summarized in Figure 4. A general trend is that the deeper the surrogate encoder's architecture, the better performance it can achieve on the downstream tasks. For instance, for SimCLR (Figure 4a), the DA on STL-10 and CIFAR-10 are 0.728 and 0.657 when the surrogate encoder's architecture is ResNet-18, while the DA increases to 0.759 and 0.697 when the surrogate encoder's architecture is changed to ResNet-50. This may be because a deeper model architecture can provide a wider parameter space and greater representation ability. Therefore, in general, deeper surrogate encoder's architectures can better "copy" the functionality from victim encoders. Note that in the following experiments, the adversary uses ResNet-50 as the surrogate encoder's architecture by default as it has comparable performance to ResNet-101 while requiring fewer resources.

Distribution of The Query Dataset. Secondly, we evaluate the impact of the query dataset's distribution. In the real-world scenario, the adversary may or may not have the query dataset that is from the same distribution as the victim encoder's pre-training dataset. Here the adversary leverages ResNet-50 as the surrogate model's architecture and cosine similarity as the similarity function. Regarding the query dataset, the adversary may leverage the training dataset of CIFAR-10, CIFAR-100, and GTSRB and the unlabeled dataset of STL-10 as the query dataset to perform the attacks. The results are shown in Figure 5. First, we observe that the model stealing attack is more effective with the same distribution query dataset. For instance, given the victim model trained by SimCLR (Figure 5a), when the downstream task is STL-10 classification, the DA for the surrogate encoders are 0.759, 0.646, 0.651, and 0.538 when the query dataset is STL-10, CIFAR-10, CIFAR-100, and GTSRB, respectively. This demonstrates that the same distribution query dataset can better steal the functionality of the victim encoder.

Another observation is that the distribution of the surrogate dataset may also influence DA on different tasks. For instance, given the victim model trained by BYOL (Figure 5c), when the downstream task is CIFAR-10 classification, the DA is 0.814 with CIFAR-10 as the query dataset, while only 0.769 with STL-10 as the query dataset. However, when the downstream task is STL-10 classification, the DA is 0.799 with CIFAR-10 as the query dataset but increases to 0.946 with STL-10 as the query dataset. Therefore, if the adversary is aware of the downstream task, they can construct a query dataset that is close to the downstream tasks to improve the stealing performance.

Similarity Function. Finally, we investigate the effect of similarity functions used in model stealing attacks. Besides cosine similarity, the adversary can also use mean absolute error (MAE) and mean square error (MSE) to match the victim encoder's embeddings. Here we assume that the adversary leverages ResNet-50 as the surrogate model's architecture and STL-10 as the query dataset. The results are shown in Figure 6. We can see that cosine similarity outperforms MAE and MSE in most settings. For instance, given the victim model trained by MoCo v2 (Figure 6b), the DA are all below 0.5 when using MAE and MSE. This can be credited to the normalization effect of cosine similarity, which helps to better learn the embeddings [21]. This indicates that cosine similarity may better facilitate the stealing process.

Monetary Cost. We compare the monetary costs of pre-training an SSL encoder and stealing an SSL encoder. We first measure the training cost of clean encoders. To pre-train a ResNet-50 encoder, SimCLR needs 60 hours with 32 TPU v3s, MoCo v2 uses 212 hours with 8 NVIDIA V100 GPUs, and BYOL takes 72 hours with 32 NVIDIA V100 GPUs (the training information is from the official or open-source implementation as mentioned in Section 5.1). The cost of model stealing contains two parts: querying the victim encoders and training the surrogate encoders locally. We use the GPU price from Google cloud[6] to calculate the price for pre-training (i.e., We run our experiments on one NVIDIA A100 GPU whose price is \$2.934 per hour). Meanwhile, we refer to the querying price, \$1 per 1,000 queries, from AWS.[7] We adopt the unlabeled STL-10 dataset (50,000 samples), cosine similarity, and different architectures to launch model stealing attacks. The monetary costs are shown in Table 3. We observe that the cost of stealing the pre-trained encoder is much smaller than pre-training it from scratch. For instance, pre-trains a BYOL ResNet-50 encoder takes \$5,713.92 while stealing it with a ResNet-101 encoder only takes \$72.49. This indicates that an adversary can "copy" the victim encoder with much less cost.

Table 3: Monetary Cost (\$). Here Res denotes ResNet.

	Pre-training	Stealing			
		Res18	Res34	Res50	Res101
SimCLR	**1,920.00**	58.24	61.10	66.67	74.50
MoCo v2	**4,206.08**	58.13	61.09	66.55	74.37
BYOL	**5,713.92**	58.16	60.84	64.28	72.49

5.4 *SSLGuard*

In this section, we adopt *SSLGuard* to inject the watermark into clean encoders pre-trained by SimCLR, MoCo v2, and BYOL. We aim to validate four properties of *SSLGuard*, i.e., effectiveness, utility, undetectability, and efficiency. We will discuss the robustness of *SSLGuard* separately in Section 5.5.

Effectiveness. We first evaluate the effectiveness of *SSLGuard*. Concretely, we check whether the model owner can extract the watermark from the watermarked encoders. Ideally, the watermark should be successfully extracted from the watermarked encoder F_* and shadow encoder F_s, but not the clean encoder F. We use the generated key-tuple κ to measure the watermark rate (WR) for F, F_*, and F_s on three SSL algorithms. As shown in Table 4, the WR of F_* and F_s are all 1.00, which means encoder F_* and F_s both contain the information of $\mathcal{D}_v$ and *sk*. Meanwhile, the WR of F is 0.00. This means *SSLGuard* is generic and does not judge a clean encoder to be a watermarked encoder.

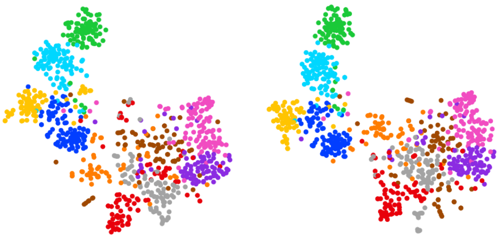

(a) F^{byol} (b) F_*^{byol}

Figure 7: The t-SNE visualizations of features output from F^{byol} and F_*^{byol} when we input 800 samples in 10 classes randomly chosen from the STL-10 training dataset. Each point represents an embedding. Each color represents one class.

Table 4: Effectiveness.

Encoder	SimCLR	MoCo v2	BYOL
F	0.00	0.00	0.00
F_*	**1.00**	**1.00**	**1.00**
F_s	**1.00**	**1.00**	**1.00**

Fidelity. One of the initial intentions of *SSLGuard* is to maintain the utility of the original downstream task. To verify its fidelity, we first take BYOL as an example and visualize embeddings output from F^{byol} (the clean encoder pre-trained by BYOL) and F_*^{byol} using t-Distributed Neighbor Embedding (t-SNE) [52], which is depicted in Figure 7. We observe that the t-SNE results of F^{byol} and F_*^{byol} are almost identical and the embeddings are successfully separated by both encoders. This demonstrates that watermarked encoder trained by *SSLGuard* can faithfully reproduce the embeddings generated from the clean encoder. Also, we train downstream classifiers by using three watermarked encoders F_*^{simclr}, F_*^{moco} and F_*^{byol} on STL-10, CIFAR-10, F-MNIST, and MNIST. Table 5 shows the DA in different scenarios. We observe that the DA of the watermarked encoders is almost the same as the clean encoders. For instance, compared to F^{simclr}, the DA for F_*^{simclr} only drops up to 0.009 from CDA. The evaluation shows that our watermarking algorithm *SSLGuard* does not sacrifice the utility of clean encoders on different downstream tasks.

Undetectability. We then check if the watermark can be extracted by a *no-matching* key-tuple. Through *SSLGuard*, we generate three key-tuples: κ^{simclr}, κ^{moco} and κ^{byol}. We use one of the key-tuples to verify other watermarked encoders, such as using κ^{simclr} to judge F_*^{moco}. As shown in Table 6, we see that the WR are all 0.00 in *no-match* pairs, which means we cannot use a non-matching κ to verify a watermarked encoder.

[6] https://cloud.google.com/compute/gpus-pricing
[7] https://aws.amazon.com/rekognition/pricing

Table 5: Fidelity (DA). The Value in the parenthesis denotes the difference between CDA.

Task	F_*^{simclr}	F_*^{moco}	F_*^{byol}
STL-10	0.781 (**-0.002**)	0.888 (**-0.001**)	0.940 (**-0.008**)
CIFAR-10	0.765 (**-0.001**)	0.701 (**-0.011**)	0.857 (**+0.002**)
MNIST	0.965 (**-0.009**)	0.956 (**+0.016**)	0.966 (**+0.002**)
F-MNIST	0.878 (**+0.004**)	0.845 (**-0.007**)	0.894 (**+0.000**)

Table 6: Undetectability.

Key-tuple	F_*^{simclr}	F_*^{moco}	F_*^{byol}
κ^{simclr}	**1.00**	0.00	0.00
κ^{moco}	0.00	**1.00**	0.00
κ^{byol}	0.00	0.00	**1.00**

Efficiency. *SSLGuard* injects watermark into SimCLR, MoCo v2, and BYOL using 17.5hrs, 17.36hrs, and 10.70hrs, respectively, which are only 29.17%, 8.19%, and 14.86% of the time cost to pre-train SSL encoders, and the watermark extraction time is only 1.51s, 2.08s, and 1.82s, respectively. Note also that we use only a single GPU (A100) in the watermark injection process, which is much less than the requirement for pre-training the SSL encoders. This demonstrates that *SSLGuard* can inject and extract watermark efficiently.

5.5 Robustness

We now quantify the robustness of *SSLGuard*. Concretely, we evaluate *SSLGuard* against model stealing and the following watermark removal attacks: Input preprocessing, output perturbing, and model modification. For instance, the attacker can add noise to the input samples or output embeddings. Also, the attacker can modify the parameters of the encoder by overwriting, pruning, and fine-tuning. Since watermark removal attacks may affect the performance of the encoders, and the attacker aims to "clean" the encoder but keep its functionality, we measure DA and WR simultaneously of these surrogate encoders. We note that the victim encoders are watermarked encoders, and we leverage SimCLR, MoCo, and BYOL to denote F_*^{simclr}, F_*^{moco}, and F_*^{byol} in this subsection. Regarding the downstream accuracy, we only show the results on BYOL (SimCLR and MoCo have similar trends).

5.5.1 Input Preprocessing

Here we consider that the attacker may add i.i.d. Gaussian noise to each input image by $x' = x + \varepsilon_1 \cdot \mathcal{N}(0,1)$. We evaluate DA on four downstream tasks and WR when we use different ε_1. The results of WR are shown in Figure 8a and DA are shown in Figure 9a. We first observe that DA drops as ε_1 increases. For instance, the DA on CIFAR-10 for drops from 0.932 to 0.865 when ε_1 increases from 0.05 to 0.15. On the other hand, the WR are all 1.00 for different ε_1 on SimCLR, MoCo, and BYOL, respectively. This may because when we inject the trigger into $\mathcal{D}_p$, the distribution of $\mathcal{D}_v$ is too special, so our watermarked encoder can remember these special samples, which is robust to the input noise attacks.

5.5.2 Output Perturbing

The adversary can also add some perturbations to the embeddings before returning them as the outputs. Here we consider two kinds of perturbations, i.e., random noising and truncation.

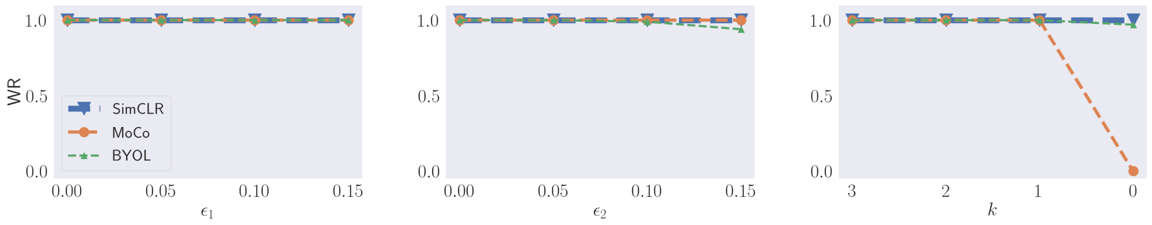
Output Noising. The adversary may return the perturbing embeddings by adding i.i.d. Gaussian noise as $h' = h + \varepsilon_2 \cdot \mathcal{N}(0,1)$, where h is the original embedding, h' is the perturbed embedding, and ε_2 is a hyper-parameter to control the noise level. Then, we evaluate DA and WR on different ε_2. From Figure 9b, we observe that DA decreases when ε_2 increases. For instance, when ε_2 increases from 0.05 to 0.15, DA on STL-10 drops from 0.940 to 0.905. However, the WR remains above 0.50 for all watermarked encoders (see Figure 8b), which means when we fed the embeddings with noise into the decoders, the secret vector can still be successfully extracted. Therefore, the attackers cannot remove the watermark even if they add random noise to the embeddings at the expense of decreasing the model's performance.

Truncation. The adversary may decrease the precision of the embeddings by leveraging truncation. For instance, the adversary retains k decimal places for each value in the embeddings, e.g., if $k = 3$, the adversary modifies the value 1.2368 to 1.236, and changes 1.2368 to 1 when $k = 0$. Figure 8c and Figure 9c shows WR and DA under different k. We observe that DA has a sharp drop when k decreases from 1 to 0 (SimCLR and MoCo show similar trends). Meanwhile, WR are all above 0.5 instead MoCo, i.e., WR of MoCo drops to 0.00 when $k = 0$, but the DA on STL-10 is only 0.10. Therefore, attackers cannot remove the watermark from the encoder while remaining its functionality.

5.5.3 Model Modification

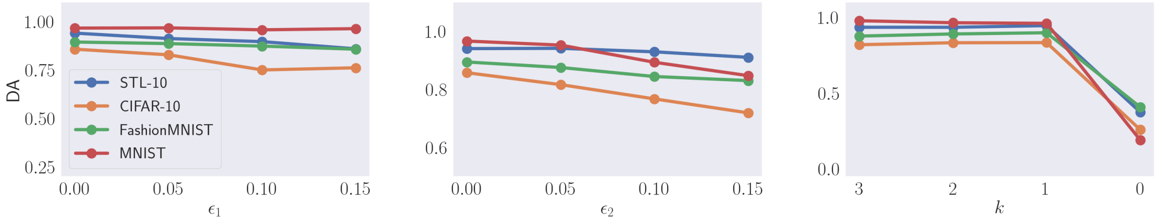
When attackers have the white-box access to the encoder, they can try to remove the watermark through modifying the encoder's parameters. In this section, we consider three methods of model modification: watermark overwriting, model pruning, and fine-tuning.

Overwriting. The attacker can also leverage *SSLGuard* to inject a new watermark into an SSL encoder whether or not they knows that the encoder has already been injected with a watermark. The attacker aims to generate a new watermarked encoder F_*' from F_* with a different key-tuple. We want to confirm if our original watermark can remain in F_*' as well. For each F_*', we measure the DA on different downstream tasks and the WR of the original key-tuple. The results are shown in Table 7. We observe that although we overwrite the watermarked encoder with a new key-tuple to generate a new encoder, the original watermark is still preserved, i.e., the WR of the original watermark in the new encoder is 1.00. This indicates that the original watermark can still be preserved even if the adversary overwrites a new watermark to the model.



(a) Input noising **(b) Output noising** **(c) Output truncation**

Figure 8: The WR on different watermark removal attacks.



(a) Input noising **(b) Output noising** **(c) Output truncation**

Figure 9: The DA on different watermark removal attacks. The victim encoder is BYOL.

Table 7: Overwriting.

		SimCLR	MoCo	BYOL
DA	STL-10	0.785	0.888	0.954
	CIFAR-10	0.765	0.685	0.863
	MNIST	0.962	0.955	0.977
	F-MNIST	0.885	0.837	0.905
WR	Overwriting key	1.00	1.00	0.98
	Original key	**1.00**	**1.00**	**1.00**

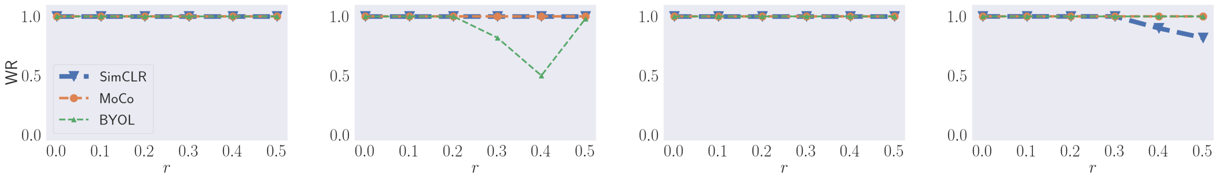
Pruning. Pruning is an effective technology for model compression [59]. It is also considered a watermark removal attack since many neurons may be disabled which reduce the effectiveness of the watermark [37]. In this part, we leverage global and local unstructured pruning to the watermarked encoder. In the global pruning setting, we set r fraction of weights of the convolutional layers which has the smallest absolute value in all layers to 0. Compared to the global pruning, i.e., put together all the connections across different layers and compare them, local pruning aims to prune a proportion of connections with the smallest absolute value in the same layer. We show the WR and DA in the first two subfigures of Figure 10 and Figure 11, respectively. We observe that DA and WR drops a little as the ratio increases in global pruning. However, for local pruning, there is a larger downward trend in DA. For instance, DA is 0.954 when $r = 0.1$ and 0.871 when $r = 0.5$, this is because local pruning cannot preserve the global information in the model properly. In general, most of the WR are 1.0, which means *SSLGuard* is robust to different pruning settings. We also notice a special case here, i.e., on BYOL, when $r = 0.4$, the WR is 0.50. This is the worst case in our experiment, which demonstrates that we use watermark verification threshold $th_w = 0.5$ in *SSLGuard* is reasonable. Also note that for all clean encoders we evaluate in this paper, the WR is 0. This means the th_w can be set to a smaller value to better verify the watermarked encoder as we discussed in Section 4.1.

Fine-tuning. After pruning, the adversary can fine-tune the surrogate encoders under the victim encoder's supervision, which is following the setting in [26]. This process is also called fine-pruning [35]. The goal of fine-tuning is to regain DA's drop. We fine-tune all the weights of the pruned encoders (global and local) by the MSE loss function. We note that we freeze the BatchNorm layers of the pruned encoders due to reducing inaccurate batch statistics estimation caused by a small batchsize [54]. The WR are shown in Figure 10c and Figure 10d, and the DA are shown in Figure 11c and Figure 11d. We observe that fine-tuning can recover the lost information from the victim encoder. For instance, when $r = 0.3$ in local pruned model, DA on STL-10 is 0.917. After fine-tuning the pruned model, DA comes to 0.954. Meanwhile, WR increases as DA recovers. This means *SSLGuard* is robust to the fine-tuning.

5.5.4 Model Stealing

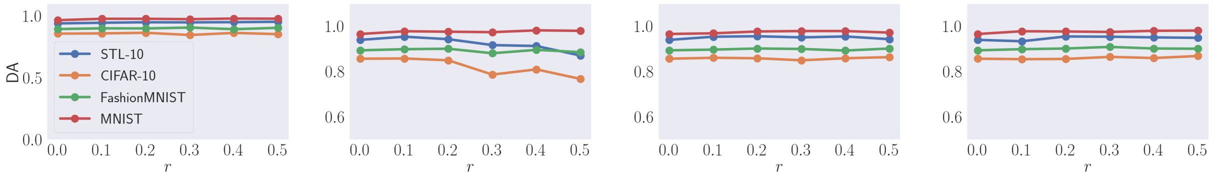
We then quantify the robustness of *SSLGuard* through the lens of model stealing attacks. Note that we only consider the most powerful surrogate encoder's architectures and most effective query datasets. Concretely, based on the evaluation in Section 5.2, we consider ResNet-50 and ResNet-101 as the surrogate encoder's architectures and STL-10 as the query dataset. We name the three attacks `Steal-1`, `Steal-2`, and `Steal-3`. The details of each attack are shown in Table 8.

The WR and DA for different attacks are shown in Table 9. We observe that although the model stealing attack is effective against the watermarked encoder, we can still verify the ownership of the surrogate model as the WR is also high.



(a) Global pruning (b) Local pruning (c) Global fine-tuning (d) Local fine-tuning

Figure 10: The WR of pruned and fine-tuned encoders.



(a) Global pruning (b) Local pruning (c) Global fine-tuning (d) Local fine-tuning

Figure 11: The DA of pruned and fine-tuned encoders. The victim encoder is BYOL.

Table 8: Details of different model stealing attacks.

Attacks	Query dataset	Architecuture	Loss function
`Steal-1`	STL-10	ResNet-50	Cosine
`Steal-2`	STL-10	ResNet-101	Cosine
`Steal-3`	STL-10 (s)	ResNet-50	Cosine

For instance, for `Steal-2` against the watermarked encoder pre-trained by BYOL, we denote it as S_2^{byol}, the DA is 0.937 and 0.815 on STL-10 and CIFAR-10, while the WR is 1.00, which indicates that the watermark injected by *SSLGuard* can still preserve in the surrogate encoder stolen by the adversary. We also have similar observations on `Steal-1` and `Steal-3`, which demonstrate the robustness of *SSLGuard* under model stealing attacks.

6 Discussion

The Necessity of The Shadow Encoder. The reason why *SSLGuard* can extract watermarks from the surrogate encoder is that it locally simulates a model stealing process by using a shadow dataset and shadow encoder. In this part, we aim to demonstrate the need for such a design. We discard the shadow encoder, and inject the watermark into a clean pre-trained encoder on SimCLR, MoCo v2, and BYOL. Then we get the corresponding key-tuples. The key-tuples can extract watermarks successfully. However, when We mount `Steal-1` to the watermarked encoders to generate three surrogate encoders (i.e., S^{simclr}, S^{moco}, and S^{byol}), the WR are all 0.00, which means the watermark may not be verified. Meanwhile, DA for S^{byol} are 0.945, 0.735 and, 0.843, and 0.926 on STL-10, CIFAR-10, F-MNIST, and MNIST, respectively. This indicates that the adversary can successfully steal the victim encoder as the DA for the surrogate encoder are close to the target encoder. In conclusion, *SSLGuard* cannot work well without the shadow encoder as the adversary can steal a surrogate encoder with high utility while bypassing the watermark verification process. Therefore, the shadow encoder is crucial for defending against model stealing attacks.

Table 9: The DA and WR of model stealing attacks against the watermarked encoders.

Attacks	Metric		SimCLR	MoCo	BYOL
`Steal-1`	DA	STL-10	0.721	0.890	0.938
		CIFAR-10	0.685	0.628	0.791
		F-MNIST	0.832	0.809	0.830
		MNIST	0.928	0.923	0.915
	WR		**1.00**	**0.96**	**1.00**
`Steal-2`	DA	STL-10	0.727	0.871	0.937
		CIFAR-10	0.677	0.628	0.815
		F-MNIST	0.840	0.827	0.865
		MNIST	0.935	0.919	0.961
	WR		**0.99**	**0.90**	**1.00**
`Steal-3`	DA	STL-10	0.732	0.874	0.923
		CIFAR-10	0.677	0.658	0.784
		F-MNIST	0.827	0.823	0.851
		MNIST	0.932	0.940	0.922
	WR		**1.00**	**0.95**	**0.98**

The Choice of Mask. In our experiments, we set the covering space of the mask as 35%. We also leverage different masks M, i.e., 5% and 50% to inject watermark into BYOL, then we mount `Steal-1` to the watermarked encoders, the WR are 0.99 and 1.00. The results show that the WR is similar when we leverage different covering spaces of the masks, which indicates that *SSLGuard* is effective under different masks.

Extension to Other Types of Datasets. In this paper, we only focus on encoders pre-trained on image datasets. To extend *SSLGuard* into encoders pre-trained on other types of datasets such as texts or graphs [19, 57], the main challenge is to define a suitable trigger pattern in the language or graph domain. Then we can apply a similar method to watermark those models. We leave it as our future work to further explore the effectiveness of *SSLGuard* on other domains such as texts or graphs.

7 Related Work

Privacy and Security for SSL. There have been more and more studies on the privacy and security of self-supervised learning. Jia et al. [27] sum up 10 security and privacy problems for SSL. Among them, only a small part has been studied. Liu et al. [34] study membership inference attacks against contrastive learning-based pre-train encoder. Concretely, Liu et al. [34] leverage data augmentations over the original samples to generate multiple augmented views to query the target encoder and obtain the embeddings. Then, the authors measure the similarities among the embeddings. The intuition is that, if the sample is a member, then the similarities should be high since many augmented views of the sample are used during the training procedure, which makes them embedded closer. He and Zhang [24] perform the first privacy analysis of contrastive learning. Concretely, the authors observe that the contrastive models are less vulnerable to membership inference attacks, while more vulnerable to attribute inference attacks. The reason is that contrastive models are more generalized with less overfitting level, which lead to fewer membership inference risks, but the representation learned by contrastive learning are more informative, thus leaking more attribute information. Jia et al. [28] propose the first backdoor attack against SSL pre-trained encoders. By injecting the trigger pattern in the pre-training process of an encoder that correlated to a specific downstream task, the backdoored encoder can behave abnormally for this downstream task. The author further shows that triggers for multiple tasks can be simultaneously injected into the encoder.

DNNs Copyright Protection. In recent years, several techniques for DNNs copyright protection have been proposed. Among them, DNNs watermarking is one of the most representative algorithms. Jia et al. [26] propose an entangled watermarking algorithm that encourage the classifiers to represent training data and watermarks similarly. The goal of the entanglement is to force the adversary to learn the knowledge of the watermarks when he steals the model. DNN fingerprinting is another protection method. Unlike watermarking, the goal of fingerprinting is to extract a specific property from the model. Cao et al. [8] introduce a fingerprinting extraction algorithm, namely IPGuard. IPGuard regards the data points near the classification boundary as the model's fingerprint. If a suspect classifier predicts the same labels for these points, then it will be judged as a surrogate classifier. Chen et al. [12] propose a testing framework for supervised learning models. They propose six metrics to measure whether a suspect model is a copy of the victim model. Among these metrics, four of them need white-box access, and black-box access is enough for the rest.

8 Conclusion

In this paper, we first quantify the copyright breaching threats of SSL pre-trained encoders through the lens of model stealing attacks. We empirically show that the SSL pre-trained encoders are highly vulnerable to model stealing attacks. This is because the rich information in the embeddings can be leveraged to better capture the behavior of the victim encoder. To protect the copyright of SSL pre-trained encoder, we propose *SSLGuard*, a robust black-box watermarking algorithm for the SSL pre-trained encoders. Concretely, given a secret vector, *SSLGuard* injects a watermark into a clean pre-trained encoder and outputs a watermarked version. The shadow training technique is also applied to preserve the watermark under potential model stealing attacks. Extensive evaluations show that *SSLGuard* is effective in embedding and extracting watermarks and robust against model stealing and different types of watermark removal attacks such as input noising, output perturbing, overwriting, model pruning, and fine-tuning.

References

[1] https://openai.com/api/. 2, 4

[2] https://www.clarifai.com/. 2, 4

[3] https://www.cs.toronto.edu/~kriz/cifar.html. 7

[4] http://yann.lecun.com/exdb/mnist/. 8

[5] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. In *USENIX Security Symposium (USENIX Security)*, pages 1615–1631. USENIX, 2018. 2, 4

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*. NeurIPS, 2020. 2

[7] T. Tony Cai, Jianqing Fan, and Tiefeng Jiang. Distributions of Angles in Random Packing on Spheres. *Journal of Machine Learning Research*, 2013. 5

[8] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. IPGuard: Protecting Intellectual Property of Deep Neural Networks via Fingerprinting the Classification Boundary. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*, pages 14–25. ACM, 2021. 14

[9] Nicholas Carlini and David Wagner. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. *CoRR abs/1705.07263*, 2017. 4

[10] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. Model Extraction and Active Learning. *CoRR abs/1811.02054*, 2018. 4

[11] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. Exploring Connections Between Active Learning and Model Extraction. In *USENIX Security Symposium (USENIX Security)*, pages 1309–1326. USENIX, 2020. 4

[12] Jialuo Chen, Jingyi Wang, Tinglan Peng, Youcheng Sun, Peng Cheng, Shouling Ji, Xingjun Ma, Bo Li, and Dawn Song. Copy, Right? A Testing Framework for Copyright Protection of Deep Learning Models. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2022. 14

[13] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A Simple Framework for Contrastive Learning of Visual Representations. In *International Conference on Machine Learning (ICML)*, pages 1597–1607. PMLR, 2020. 1, 3

[14] Xiaoyi Chen, Ahmed Salem, Michael Backes, Shiqing Ma, Qingni Shen, Zhonghai Wu, and Yang Zhang. BadNL: Backdoor Attacks Against NLP Models with Semantic-preserving Improvements. In *Annual Computer Security Applications Conference (ACSAC)*, pages 554–569. ACSAC, 2021. 4

[15] Xinlei Chen, Haoqi Fan, Ross B. Girshick, and Kaiming He. Improved Baselines with Momentum Contrastive Learning. *CoRR abs/2003.04297*, 2020. 3

[16] Adam Coates, Andrew Y. Ng, and Honglak Lee. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 215–223. JMLR, 2011. 8

[17] David DeFazio and Arti Ramesh. Adversarial Model Extraction on Graph Neural Networks. *CoRR abs/1912.07721*, 2019. 4

[18] Yunjie Ge, Qian Wang, Baolin Zheng, Xinlu Zhuang, Qi Li, Chao Shen, and Cong Wang. Anti-Distillation Backdoor Attacks: Backdoors Can Really Survive in Knowledge Distillation. In *ACM International Conference on Multimedia (MM)*, pages 826–834. ACM, 2021. 6

[19] John M. Giorgi, Osvald Nitski, Bo Wang, and Gary D. Bader. DeCLUTR: Deep Contrastive Learning for Unsupervised Textual Representations. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 879–895. ACL, 2021. 13

[20] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations (ICLR)*, 2015. 4

[21] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Ávila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*. NeurIPS, 2020. 3, 10

[22] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. Momentum Contrast for Unsupervised Visual Representation Learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9726–9735. IEEE, 2020. 1, 3

[23] Xinlei He, Rui Wen, Yixin Wu, Michael Backes, Yun Shen, and Yang Zhang. Node-Level Membership Inference Attacks Against Graph Neural Networks. *CoRR abs/2102.05429*, 2021. 4

[24] Xinlei He and Yang Zhang. Quantifying and Mitigating Privacy Risks of Contrastive Learning. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 845–863. ACM, 2021. 4, 14

[25] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High Accuracy and High Fidelity Extraction of Neural Networks. In *USENIX Security Symposium (USENIX Security)*, pages 1345–1362. USENIX, 2020. 4

[26] Hengrui Jia, Christopher A. Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. Entangled Watermarks as a Defense against Model Extraction. In *USENIX Security Symposium (USENIX Security)*, pages 1937–1954. USENIX, 2021. 2, 4, 12, 14

[27] Jinyuan Jia, Hongbin Liu, and Neil Zhenqiang Gong. 10 Security and Privacy Problems in Self-Supervised Learning. *CoRR abs/2110.15444*, 2021. 14

[28] Jinyuan Jia, Yupei Liu, and Neil Zhenqiang Gong. BadEncoder: Backdoor Attacks to Pre-trained Encoders in Self-Supervised Learning. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2022. 2, 4, 8, 14

[29] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, 2015. 8

[30] Kalpesh Krishna, Gaurav Singh Tomar, Ankur P. Parikh, Nicolas Papernot, and Mohit Iyyer. Thieves on Sesame Street! Model Extraction of BERT-based APIs. In *International Conference on Learning Representations (ICLR)*, 2020. 2, 4

[31] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial Examples in the Physical World. *CoRR abs/1607.02533*, 2016. 4

[32] Zheng Li, Chengyu Hu, Yang Zhang, and Shanqing Guo. How to Prove Your Model Belongs to You: A Blind-Watermark based Framework to Protect Intellectual Property of DNN. In *Annual Computer Security Applications Conference (ACSAC)*, pages 126–137. ACM, 2019. 2, 4

[33] Zheng Li and Yang Zhang. Membership Leakage in Label-Only Exposures. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 880–895. ACM, 2021. 4

[34] Hongbin Liu, Jinyuan Jia, Wenjie Qu, and Neil Zhenqiang Gong. EncoderMI: Membership Inference against Pre-trained Encoders in Contrastive Learning. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2021. 2, 4, 14

[35] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks. In *Research in Attacks, Intrusions, and Defenses (RAID)*, pages 273–294. Springer, 2018. 12

[36] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into Transferable Adversarial Examples and Black-box Attacks. *CoRR abs/1611.02770*, 2016. 4

[37] Nils Lukas, Edward Jiang, Xinda Li, and Florian Kerschbaum. SoK: How Robust is Image Classification Deep Neural Network Watermarking? In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2022. 2, 12

[38] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. Adversarial Frontier Stitching for Remote Neural Network Watermarking. *CoRR abs/1711.01894*, 2017. 4

[39] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff Nets: Stealing Functionality of Black-Box Models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4954–4963. IEEE, 2019. 2, 4

[40] Nicolas Papernot, Patrick D. McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical Black-Box Attacks Against Machine Learning. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*, pages 506–519. ACM, 2017. 2, 4

[41] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision. In *International Conference on Machine Learning (ICML)*, pages 8748–8763. PMLR, 2021. 3

[42] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. DeepSigns: A Generic Watermarking Framework for IP Protection of Deep Learning Models. *CoRR abs/1804.00750*, 2018. 4

[43] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *CoRR abs/1409.0575*, 2015. 1, 7

[44] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden Trigger Backdoor Attacks. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 11957–11965. AAAI, 2020. 4

[45] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models. In *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2019. 2, 4

[46] Yun Shen, Xinlei He, Yufei Han, and Yang Zhang. Model Stealing Attacks Against Inductive Graph Neural Networks. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2022. 2, 4

[47] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership Inference Attacks Against Machine Learning Models. In *IEEE Symposium on Security and Privacy (S&P)*, pages 3–18. IEEE, 2017. 2, 4

[48] Liwei Song and Prateek Mittal. Systematic Evaluation of Privacy Risks of Machine Learning Models. In *USENIX Security Symposium (USENIX Security)*. USENIX, 2021. 4

[49] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark: A Multi-class Classification Competition. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1453–1460. IEEE, 2011. 8

[50] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing Machine Learning Models via Prediction APIs. In *USENIX Security Symposium (USENIX Security)*, pages 601–618. USENIX, 2016. 2, 4

[51] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding Watermarks into Deep Neural Networks. In *International Conference on Multimedia Retrieval (ICMR)*, pages 269–277. ACM, 2017. 4

[52] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 2008. 10

[53] Bang Wu, Xiangwen Yang, Shirui Pan, and Xingliang Yuan. Model Extraction Attacks on Graph Neural Networks: Taxonomy and Realization. *CoRR abs/2010.12751*, 2020. 4

[54] Yuxin Wu and Kaiming He. Group Normalization. In *European Conference on Computer Vision (ECCV)*, pages 3–19. Springer, 2018. 12

[55] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR abs/1708.07747*, 2017. 8

[56] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y. Zhao. Latent Backdoor Attacks on Deep Neural Networks. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2041–2055. ACM, 2019. 4, 6

[57] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph Contrastive Learning with Augmentations. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*. NeurIPS, 2020. 13

[58] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph. Stoecklin, Heqing Huang, and Ian Molloy. Protecting Intellectual Property of Deep Neural Networks with Watermarking. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*, pages 159–172. ACM, 2018. 2

[59] Michael Zhu and Suyog Gupta. To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression. In *International Conference on Learning Representations (ICLR)*, 2018. 12