I Can Still Steal Your Encoder: A Defense-Penetrating Encoder-Stealing Attack

Rongbin Xiao[1], Changyu Dong[1] *, Jie Zhang[2], Yan Pang[1], Zihan Xie[1], and Han Wu[3]

[1] Guangzhou University, Guangzhou, China rongbin.xiao@e.gzhu.edu.cn, changyu.dong@gmail.com, yanpangee@gmail.com, zihan.xie@e.gzhu.edu.cn
[2] University of Bath, Bath, UK jz2558@bath.ac.uk
[3] University of Southampton, Southampton, UK H.Wu@soton.ac.uk

Abstract. The rise of Encoder-as-a-Service (EaaS) has made pre-trained encoders accessible for various AI tasks, but this has introduced significant security concerns, particularly with model stealing attacks. While defenses like the B4B mechanism [6] have been proposed to protect against such attacks, we reveal critical vulnerabilities in B4B's strategies. B4B employs techniques such as embedding space coverage estimation, cost-based perturbation, and embedding transformations to thwart attackers. However, we introduce the first defense-penetrating attack that bypasses these protections. Our attack effectively circumvents all three defense mechanisms, enabling attackers to steal high-quality encoders with minimal degradation in performance. Extensive experiments show that the stolen encoder performs almost as well as the original, highlighting the weaknesses in B4B and similar defenses. Our work exposes significant gaps in the security of EaaS systems and calls for more robust, active defense strategies against model stealing.

Keywords: Model Stealing · Defense-Penetrating Attack · Representation Learning.

1 Introduction

Recent advancements in artificial intelligence (AI) and deep learning (DL) have led to the widespread adoption of representation learning techniques, which train encoders through either supervised [25] or unsupervised [13, 28] methods. These encoders extract rich and versatile features, enabling superior performance across a wide range of downstream tasks. However, the cost of training powerful encoders is prohibitively high. As a result, high-performance encoders are typically pre-trained by leading AI companies with substantial computational resources and are shared for commercial use through cloud-based platforms, a model known as Encoder-as-a-Service (EaaS). EaaS represents a significant evolution from traditional Machine-Learning-as-a-Service (MLaaS). Instead of providing downstream classifiers as cloud services, EaaS enables customers to

* Corresponding author

query APIs for feature embeddings generated by pre-trained encoders. These feature embeddings (representations) are then used to train or test customer-specific classifiers. Leading technology companies, such as OpenAI, Google, have capitalized on this trend by offering embedding APIs for text and image data.

Despite the convenience and accessibility offered by EaaS, the deployment of these services raises significant security concerns, particularly in the context of model stealing attacks (MSAs) [26, 19]. MSAs aim to replicate the parameters or functionality of a victim model by leveraging its outputs through continuous queries. Successful MSAs not only pose a threat to the intellectual property of the victim model but can also serve as a stepping stone for further attacks, such as adversarial attacks [12], backdoor attacks [16], and membership inference attacks [24]. Previous studies on stealing attacks targeting encoders, i.e. encoder-stealing attacks [7, 17, 23] have highlighted that encoders are more vulnerable than classifiers [21, 11, 27]. This increased vulnerability stems from the fact that encoders produce high-dimensional embeddings, which inherently expose more internal details of the model.

In response to these threats, defenses against encoder-stealing attacks have been developed. However, most of these approaches are reactive in nature. These reactive defenses do not actively prevent encoder theft but focus on post-theft detection using techniques such as watermarking [5, 18] or performing dataset inference [8] to identify stolen copies. By the time theft is detected, the damage has already been done.

In practice, active defense mechanisms that thwart stealing attempts are more desirable. Early active defenses [7, 17] used simple perturbation strategies to reduce the effectiveness of model stealing attacks, but they also significantly degraded the encoder's performance, making them unsuitable for real-world deployment. Recently, the B4B mechanism was proposed by [6]. This state-of-the-art (SOTA) defense presents a big step forward, disrupting the attacker's ability to extract sensitive representations while maintaining the encoder's utility for legitimate users.

That said, our thorough investigation reveals that the B4B mechanism is not as robust as claimed. It is vulnerable to exploitation and circumvention by adversaries. Its vulnerability has remained largely unexposed because existing attacks focus primarily on scenarios without defenses in place. In this paper, we introduce the first defense-penetrating attack that undermines the key strategies employed by B4B. The B4B mechanism is built on three core principles: (1) It adds Gaussian noise to the output embeddings, with the goal of misleading the adversary's model training and degrading the performance of stolen models. (2) To maximize utility for legitimate users, it continuously monitors queries and establishes a cost function to penalize malicious users. (3) To prevent evasion through Sybil attacks, it applies transformations to embeddings, ensuring that each user's embedding differs significantly from others. However, we demonstrate that all three of these defense strategies can be easily circumvented or nullified.

In summary, our work makes the following contributions:

- We identified critical vulnerabilities in existing active defenses against encoder-stealing attacks, demonstrating that these defenses are not as secure as claimed. This highlights a significant security risk for EaaS systems that rely on these defenses.
- We introduced the first defense-penetrating attack for encoder stealing. While our focus is on the B4B defense mechanism, the attack is broadly applicable to other defense strategies.
- We conducted extensive experiments to evaluate the effectiveness of our attack. The results show that the stolen encoder performs almost as well on downstream tasks as one taken without any defense mechanisms in place.

2 Related Work

Encoder-Stealing Attacks Model stealing attacks [26, 15] aim to replicate the parameters or functionality of a victim model $f_v(x;\theta)$. To achieve this, an adversary queries the victim model with a set of inputs and collects the corresponding outputs. These query-response pairs are then used as training data to construct a stolen model $f_s(x;\theta)$. Literatures [17, 7] show that model stealing targeting encoders is also feasible and effective. Attackers aim to replicate the victim encoder by training a local copy that produces representations similar to those of the victim model. Two primary training strategies are employed for encoder stealing: (1) direct alignment of representations via Mean Squared Error (MSE) loss between the victim and the stolen encoder; and (2) self-supervised training with a contrastive loss, such as InfoNCE [20] or SoftNN [10], using the victim encoder's outputs as pseudo-labels. Recent works, such as StolenEncoder [17], have further optimized the attack process by exploiting the observation that victim encoders generate similar outputs for augmented views of the same image. This allows attackers to minimize queries to the victim encoder by augmenting a single input locally and training the stolen encoder on these augmentations. Cont-Steal [23] utilizes contrastive learning techniques to enhance stealing performance by aligning the stolen embedding of an image with its target embedding and distancing embeddings of different images, achieving superior accuracy of downstream tasks compared to conventional attack methods against encoders.
Encoder-Stealing Defenses Most of the existing defenses are reactive. They employ techniques such as watermarking [5, 18], and attempt to embed unique patterns into the victim encoder. Subsequently, watermark verification can determine if a suspect encoder contains a specific watermark embedded by the victim. Similarly, dataset inference [9] leverages the unique data distribution of the victim's training set to distinguish stolen encoders from independently trained ones. These reactive methods, however, are limited to post-hoc detection rather than proactively preventing attacks.

Early active defenses, such as those proposed by [7, 17], attempt to perturb or truncate the outputs of the victim encoder to disrupt the attacker's training process. These methods introduce noise into the stolen embeddings, making them less effective for training the stolen model. However, such approaches often

degrade the quality of the representations provided to legitimate users, rendering them impractical for real-world applications. In contrast to these simple defenses, the B4B defense [6] introduces bucket occupancy tracking and noise injection mechanisms, designed to minimally impact legitimate users while thwarting attackers. Despite these improvements, our work shows that current defenses, including B4B, remain vulnerable to more sophisticated attack strategies.

3 Overview of the B4B Defense

Model stealing attacks aim to replicate the functionality of a victim model f_v trained on a dataset D_v. In this scenario, the attacker has black-box access to the victim model and constructs a stealing dataset $D_s = \{q_i, f_v(q_i)\}_{i=1}^n$, consisting of queries q_i and their corresponding outputs $f_v(q_i)$ returned by the victim model. This dataset is then used to train a stolen model f_s. Model stealing attacks have been demonstrated against various types of models, including classifiers and encoders. Specifically, when the victim model is an encoder, such attacks are referred to as encoder-stealing attacks.

B4B is an active defense framework designed to counter encoder-stealing attacks, offering protection to encoders while minimizing disruption for legitimate users. It is based on the observation that malicious queries from adversaries tend to be more diverse than those from benign users. As a result, embeddings for malicious queries span a much larger part of the embedding space compared to those used by legitimate users focused on specific downstream tasks. Building on this insight, B4B is composed of three primary components: Coverage Estimation, Cost-based Perturbation, and Per-User Representation Transformations. Each component plays a vital role in detecting and penalizing adversaries, all while preserving the experience of legitimate users.

Coverage Estimation. B4B utilizes Locality Sensitive Hashing (LSH) based on random projection [2, 22] to approximate the portion of the embedding space covered by each user. A locality sensitive hash function $\mathcal{H}$ maps inputs into 2^n buckets, ensuring that similar items are assigned to the same bucket with high probability. For a given embedding space $\mathcal{E}$, and each query q_i submitted by a user U, B4B hashes the embedding $r_i = f_v(q_i)$ using $\mathcal{H}$ and counts the number of non-empty buckets. This count represents the embedding space coverage for the user, denoted as $\tilde{\mathcal{E}}_f^U$. As the number of queries grows, $\tilde{\mathcal{E}}_f^U$ increases, and B4B posits that the coverage for benign users grows more slowly compared to that of malicious users.

Cost-based Perturbation. B4B adds Gaussian noise into the returned embedding to the user:

$$\hat{r} = r + \mathcal{N}(0, \sigma^2),$$

where $\hat{r}$ represents the noise-injected embedding. To effectively prevent encoder theft while minimizing the impact on legitimate users, the noise magnitude is

dynamically adjusted using a cost function:

$$\mathcal{C}_{\lambda,\alpha,\beta}(\tilde{\mathcal{E}}_f^U) = \lambda \times \left(\exp^{\ln \frac{\alpha}{\lambda} \times \tilde{\mathcal{E}}_f^U \times \beta^{-1}} - 1\right).$$

In B4B, the standard deviation of the Gaussian noise, σ, is set to $\mathcal{C}_{\lambda,\alpha,\beta}(\tilde{\mathcal{E}}_f^U)$. As a result, users with higher embedding space coverage $\tilde{\mathcal{E}}_f^U$ are penalized with noisier embeddings, increasing the deterrence for malicious users.

Per-User Transformations against Sybil Attacks. Since the noise introduced increases with embedding space coverage, adversaries can bypass this by using Sybil attacks, where they create multiple fake accounts (i.e. Sybils), each submitting only a small number of queries. As a result, the coverage remains low, and so does the noise. To counter this, the B4B defense assigns a distinct transformation T_U to each user. The output embeddings are passed through these transformations before being returned to the user. The goal of the transformation is to preserve legitimate utility while preventing the pooling of embeddings across different users. Concrete instantiations of transformations for B4B include Affine, Padding, Shuffle, and Binary. These transformations can be flexibly combined, further broadening the possible transformations applied to each user.

4 The Defense-Penetrating Attack

In this section, we present our Defense-Penetrating Attack. The attack comprises three key strategies: A1: embedding space coverage reduction; A2: noise reduction; and A3: embedding unification. Fig. 1 shows an overview of our Defense-Penetrating Attack. The design details of each strategy are outlined below.

4.1 A1: Embedding Space Coverage Reduction

B4B estimates the coverage of the user embedding space $\tilde{\mathcal{E}}_f^U$ by counting the proportion of occupied hash buckets through LSH, and adding a greater penalty to user embeddings as bucket occupancy increases. From an adversary's perspective, the objective is to find queries whose embeddings are more likely to be clustered into fewer buckets, thereby reducing the added penalty.

There might be many adversarial strategies that can achieve embedding space coverage reduction. Here we focus on a simple strategy, downscaling, which proved effective in our tests. In particular, we first resize a query image to a lower resolution, then convert the lower resolution image back to the original size. The rationale behind this is that downscaling constrains the input space, which in turn limits the output to a smaller sub-space of $\mathcal{E}$. Ultimately, this increases the probability that two embeddings fall into the same bucket.

The effectiveness of downscaling is illustrated in Fig. 2. Here the original query images are from ImageNet (224×224). We can see that downscaling can effectively decrease bucket occupancy. Moreover, a smaller resolution results in a slower increase in bucket occupancy. In Fig. 3, we show a t-SNE plot for the embedding space. We can see that with a smaller downscaling parameter (4×4),

Table 1. Downstream Task Accuracy of Stolen Encoder for Different Downscaling Parameters at 50k Query Cost in SimSiam.

	4×4	8×8	16×16	32×32	48×48	96×96
CIFAR10	48.530	57.730	67.190	**67.980**	59.120	31.270
STL10	42.338	48.613	53.588	57.400	**58.025**	28.375

the occupancy is smaller than that of a larger parameter (32×32), which is in turn smaller than that with no downscaling.

When the downscaling parameters are set too small, the added noise is reduced; however, the representations become overly concentrated in the embedding space, significantly restricting the portion of the space that the stolen encoder can effectively capture. Conversely, when the parameters are set too large, the added noise severely impacts the stolen encoder's ability to fit the correct representations. Both scenarios result in poor performance on downstream tasks, as demonstrated in Table 1. Consequently, for subsequent encoder-stealing experiments in Section 5.2, we selected image downscaling parameters of 32×32 and 48×48 to compare with the conventional attack.

4.2 A2: Noise Reduction

B4B adds Gaussian noise from $\mathcal{N}(0, \sigma^2)$ to output embeddings. Since the mean of the noise is 0, a well-known method to reduce the noise is for each query q_i, repeatedly query it n times to get $\hat{r}_i^1, \cdots, \hat{r}_i^n$, and obtain $\bar{r}_i = \frac{1}{n}\sum_{j=1}^{n} \hat{r}_i^j =$

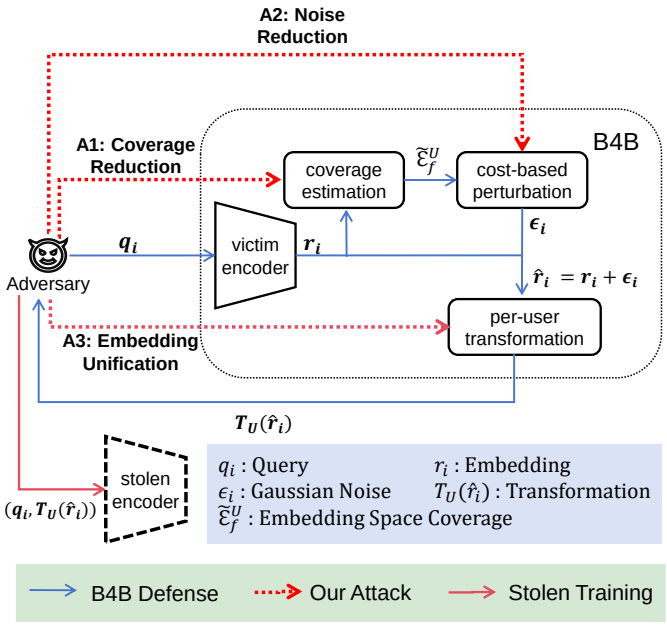

Fig. 1. Overview of Defense-Penetrating Attack against B4B.

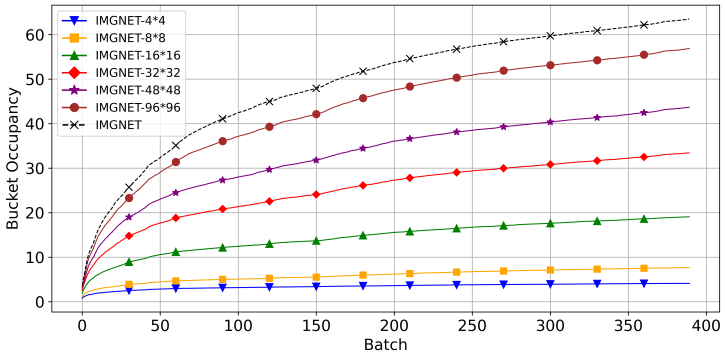

Fig. 2. Bucket Occupancy for Different Downscaling Parameters. The x-axis represents the batch being queried, and the y-axis represents the bucket occupancy.

$r_i + \mathbb{E}_{\epsilon \sim \mathcal{N}(0,\sigma^2)}(\epsilon)$. However, this will reduce the number of effective queries to $1/n$ of the budget. The encoder obtained by the adversary may not perform well due to fewer queries.

Our strategy to counter the problem is based on two insights: (1) the level of noise increases monotonically; (2) up to a scale, small noise does not have a negative impact on training. The details are as the following:

1. At the very beginning of the attack, the adversary queries a sentinel sample q_0, and obtains the embedding r_0. Note that r_0 is without any noise because $\tilde{\mathcal{E}}_f^U$ is lazily updated after answering a batch (default: 256) of queries. The adversary then queries distinct samples, records the returned embeddings, and keeps track of the number of queries.
2. At the beginning of the b-th batch, the adversary queries the sentinel sample q_0 again, and obtains the corresponding $\hat{r}_0^b$. It then compute $\epsilon_b = \hat{r}_0^b - r_0$ and the current standard deviation $\sigma_b = \|\epsilon\|_2$. if σ_b is less than a pre-chosen threshold σ_τ, the adversary keeps querying distinct samples in this batch; otherwise, the adversary repeatedly queries each sample n times and records the average of the n embeddings returned. In later batches, the adversary does not need to query q_0 again since the level of noise will not decrease.
3. The adversary repeats step 2 until the budget is exhausted, then trains an encoder using the (query, answer) pair accumulated during the query process.

To determine the threshold σ_τ, we notice that the noise $\epsilon \in \mathbb{R}^m$ is a random vector where each component ϵ_i follows a normal distribution, specifically $\epsilon_i \sim N(0, \sigma^2)$. Consequently, $\frac{\epsilon_i^2}{\sigma^2} \sim \chi^2(1)$ and $\sum_{i=1}^m \frac{\epsilon_i^2}{\sigma^2} \sim \chi^2(m)$. Our goal is to find σ_τ such that the L2 norm of the noise vector, $\|\epsilon\|$, is less than a constant α with a large probability $1 - \mathcal{P}$, i.e.

$$P(\|\epsilon\| < \alpha) = P\left(\sum_{i=1}^{m} \frac{\epsilon_i^2}{\sigma^2} < \frac{\alpha^2}{\sigma^2}\right) \geq 1 - \mathcal{P}_\tau.$$

Therefore σ_τ can be calculated by using the χ^2 inverse cumulative distribution function $\sigma_\tau = \frac{\alpha}{\sqrt{\text{chi2inv}(1-\mathcal{P}_\tau, m)}}$. We notice that σ_τ is related to α. Intuitively,

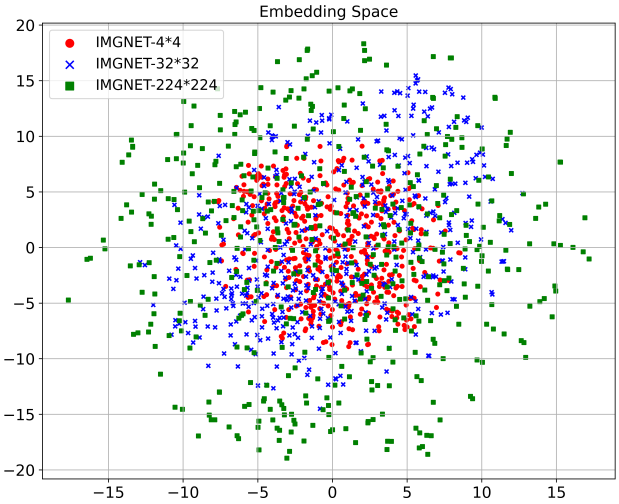

Fig. 3. t-SNE Plot for Three Downscaling Parameters.

the value of α determines the level of noise, and its impact on downstream tasks depends on the L2 norm of the embeddings. Hence, we define α relatively as $\alpha = \mathcal{R} \times \|r_0\|$ where r_0 is the embedding of the sentinel sample. Then σ_τ can be determined by hyperparameters $\mathcal{P}_\tau$ and $\mathcal{R}$.

Generalize to other active defenses. The noise reduction strategy can be generalized to other active defense mechanisms. For example, Feature Poisoning [17] and random perturbation [7] inject random noise from certain distributions into the output embeddings. Both of them can be penetrated in a similar way, i.e., by estimating the norm of noise and reducing the noise by repeated queries.

4.3 A3: Embedding Unification

To defend against potential Sybil attacks, B4B applies distinct transformations T_U to the embeddings of different users. However, we found that these transformed embeddings can be easily unified, thereby undermining the defense mechanism. More specifically, the adversary can first choose a Sybil U_0, then for other sybils U_i the adversary finds a transformation F_i such that $F_i(T_i(r)) \approx T_0(r)$. Then the embeddings obtained by each Sybil can still be pooled together for training the stolen encoder. In the main text, we focus on illustrative examples of Affine and Binary operations, while systematically presenting padding and shuffle methodologies in Appendix A.2.

Affine. The Affine transformation is defined as $T_i(r) = a_i \odot r + b_i$, where $\odot$ is the Hadamard product. B4B chooses random (a_i, b_i) for each U_i. In this case, $F_i(y) = m_i \odot y + c_i$ can be obtained as the following:

1. When U_0 is created, it queries x_1, x_2 and obtains $y_0^1 = T_0(r_1) = a_0 \odot r_1 + b_0$ and $y_0^2 = T_0(r_2) = a_0 \odot r_2 + b_0$, where $r_i = f_v(x_i)$.
2. When U_i is created, it sends the same x_1, x_2 and obtains $y_i^1 = a_i \odot r_1 + b_i$ and $y_i^2 = a_i \odot r_2 + b_i$.
3. The adversary computes and records $m_i = \frac{y_0^1 - y_0^2}{y_i^1 - y_i^2} = \frac{a_0}{a_i}$, $c = y_0^1 - m_i \odot y_i^1 = b_0 - m_i \odot b_i$

It is easy to see that $F_i(T_i(r)) = m_i \odot (a_i \odot r + b_i) + c_i = a_0 \odot r + m_i \odot b_i + b_0 - m_i \odot b_i = T_0(r)$.

Binary. The Binary transformation maps the original representation r into a binary vector based on a random partition of the embedding space. Specifically, B4B chooses n random pair$\{(a_i^j, b_i^j)\}_{j=1}^n$ for user U_i, where each a_i^j is a vector of the same dimension as the embedding r, and b_i^j is a scalar value. It also defines a binarization function $B(r, a_i^j, b_i^j) = \text{Sign}(a_i^j \cdot r + b_i^j)$ where Sign returns 1 for positive value and 0 for non-positive value. The returned binarized embedding of r is $T_i(r) = B(r, a_i^1, b_i^1)|| \cdots ||B(r, a_i^n, b_i^n)$. Note that this binarization process is lossy and has a greater impact on legitimate users' performance than other transformations. It is more difficult to invert because it is non-linear. That said, we developed an effective probabilistic algorithm for unifying binarized embeddings for different uses. Experimental results show that this algorithm achieves a 99.88% success rate. The details of the algorithm are as follows:

1. When U_0 is created, it queries t samples $x_1, \cdots, x_t$ and obtains $y_0^1, \cdots, y_0^t$, where $y_0^j = T_0(f_v(x_j))$ are binarized embedding.
2. When U_i is created, it queries the same t samples and obtains $y_i^1, \cdots, y_i^t$.
3. Build a lookup table L from U_i's binarized embedding to U_0's: for each index j, record $L[j] = (z_0, z_1)$, such that

$$z_b = \arg\max_{\beta \in \{0,1\}} \sum_{k=1}^{t} \mathbb{1}(y_i^k[j] = b \wedge y_0^k[j] = \beta), b \in \{0, 1\}.$$

   Here (z_0, z_1) stores the most likely value of $y_0^k[j]$ when $y_i^k[j]$'s value is 0 or 1.
4. Then $F_i(y)$ is defined as: for each j, if $y[j] = 0$, set $y[j] = L[j][0]$; else set $y[j] = L[j][1]$.

Generalize to other transformations. In general, our strategy can be summarized as querying the same (or related) samples by different users, and then finding the correlations. For any linear transformation, a perfect unification transformation F_i such that $F_i(T_i(r)) = T_0(r)$ can always be found by solving a set of linear equations after a sufficient number of queries. For non-linear transformations, we notice that the non-linearity should not impact legitimate users too much and the transformations are often deterministic. Therefore, although the details may vary, a good approximation can often be found so that $F_i(T_i(r)) \approx T_0(r)$ with a high probability.

4.4 Putting Them Together

While we introduce the strategies individually, they can be used in conjunction to achieve the best result. For example, A1 can reduce the rate of noise growth, however, the embedding space coverage still increases and eventually renders the query results unusable after a certain number of queries. Therefore, A1 can be used with A2 so that the noise reduction process can be applied to later queries. For A3, ideally, the adversary can create an unlimited number of Sybils so that the query results are also noise-free. However, in practice, this might not be possible due to real-world constraints. If the number of Sybils is limited, then the adversary can use A1 and/or A2 to reduce the noise when necessary.

5 Empirical Evaluation

5.1 Experimental Setup

Following the experimental setup for B4B, we utilize two widely-used encoder frameworks, SimSiam [4] and DINO [1], as the victim encoder. Pre-trained models for both frameworks are directly downloaded from their official repositories. The SimSiam encoder produces output representations with a dimensionality of 2048, while the DINO encoder generates 1536-dimensional representations. Hyperparameters for the B4B defense were set according to the calibrated values provided in the original paper.

For the stealing attack, the attacker has black-box access to the victim encoder. The stolen encoder adopts a ResNet-50 [14] as the backbone, with its fully connected layer modified to accommodate different embedding dimensions. We use ImageNet-100k as the query dataset and employ the InfoNCE [3] loss function to train the stolen encoder. To evaluate the performance of the stolen encoder, we utilize multiple downstream classification tasks, including FashionMNIST, CIFAR10, SVHN, STL10, and IMGNET-C10 (sample 10 random classes on ImageNet). Compared to the B4B setup, we add IMGNET-C10 as an additional evaluation dataset to assess the accuracy of the stolen encoder in high-resolution downstream tasks. Detailed experimental settings and hyperparameters are provided in Appendix B.

5.2 Evaluation of Our Attack Strategies

In this section, we report the experiment results using SimSiam as the victim encoder; DINO results are in Appendix D.

Embedding Space Coverage Reduction We conducted an end-to-end analysis using two downscaling parameters, 32 and 48, at query budgets of 50k and 100k. The effectiveness of stealing is measured by the accuracy of downstream classifiers, whose inputs are pre-processed with the stolen encoder. The results are summarized in Table 2. We can see that downscaling effectively circumvents

Table 2. Downstream Task Accuracy of Stolen Encoder Using A1 in SimSiam. QUERIES stands for the number of queries used for stealing. DEFENSE indicates whether the victim encoder has deployed B4B. DOWNSCALING column shows the downscaling parameters.

QUERIES	DEFENSE	DOWN-SCALING	F-MNIST	CIFAR10	SVHN	STL10	IMGNET-C10
50K	NONE	w/o	81.910	62.580	46.370	62.738	71.000
	NONE	32×32	82.950	69.180	48.248	58.050	52.000
	B4B	w/o	49.920	48.140	35.126	34.025	**56.600**
	B4B	32×32	**82.400**	**67.980**	**47.565**	**57.400**	48.600
	B4B	48×48	78.540	59.120	35.867	58.025	47.800
100K	NONE	w/o	83.510	66.580	52.443	68.750	76.000
	NONE	32×32	85.710	76.530	57.518	63.800	56.200
	B4B	w/o	36.540	15.420	19.587	15.900	16.600
	B4B	32×32	**84.780**	**74.190**	**55.336**	**62.575**	**55.200**
	B4B	48×48	64.370	41.610	26.318	42.700	43.000

the B4B defense mechanism and facilitates the theft of encoders. In all cases, the accuracy of downstream tasks increases when downscaling is used.

We can observe that when B4B is in place, the effectiveness of attack without downscaling declines when query budgets increase, due to excessive noise added to later query results. In contrast, with downscaling, the added noise is relatively low, allowing the performance of the stolen encoder to improve as the query count increases.

In addition, we observe that for low-resolution images (FashionMNIST, CIFAR10, SVHN), stealing with ImageNet data downscaled to 32×32 in the presence of B4B even outperforms the results of stealing with the original ImageNet data when there is no defense. We attribute this to downscaling makes the queries more concentrated so that the output representations are closer to those of the low-resolution images in the embedding space (as visualized in Figure 3), enabling more information to be extracted. However, this advantage comes at the cost of reduced effectiveness in high-resolution embedding spaces, as reflected by the lower classification accuracy on the IMGNET-C10 downstream task. This reveals a possible attack optimization strategy that is overlooked in the literature, i.e., adaptively choosing query data according to the downstream task.

Noise Reduction Noise reduction involves three hyperparameters: $\mathcal{R}$, $\mathcal{P}_\tau$, and the repetition n. The values of $\mathcal{R}$ and $\mathcal{P}_\tau$ determine the noise standard deviation threshold σ_τ, and n specifies the number of repeated queries used to reduce noise once the mechanism is activated. In our experiments, we fix $\mathcal{P}_\tau = 0.05$, $n = 8$, and vary the hyperparameter $\mathcal{R}$ to assess its impact on attack performance. The downstream task accuracies are summarized in Table 3. In the table, we also show $\mathcal{N}_{td}$, which denotes the number of queries that occur before the noise reduction mechanism is activated. The final number of effective queried samples

Table 3. Downstream Task Accuracy of Stolen Encoder Using A2 in SimSiam. The ATTACK column lists the adversarial attack methods, where CONV denotes the conventional attack [7], and A2 means noise reduction is used. $\mathcal{N}_{td}$ is the number of queried samples when triggering the noise reduction mechanism. We fix the hyperparameter $\mathcal{P}_\tau = 0.05$, $n = 8$ while varying $\mathcal{R}$.

QUERIES	ATTACK	DEFENSE	$\mathcal{R}$	$\mathcal{N}_{td}$	F-MNIST	CIFAR10	SVHN	STL10	IMGNET-C10
50K	CONV	NONE	N/A	N/A	81.910	62.580	46.370	62.738	71.000
	CONV	B4B	N/A	N/A	49.920	48.140	35.126	34.025	56.600
	A2	B4B	0.1	9216	**78.540**	53.130	37.458	52.813	62.000
			0.3	13120	77.720	56.040	38.829	55.175	65.800
			0.5	15680	78.080	**56.790**	38.810	**56.375**	**66.000**
			0.7	17408	78.360	56.570	**41.272**	54.025	65.400
			1.0	28352	77.950	56.040	40.174	54.613	63.200
100K	CONV	NONE	N/A	N/A	83.510	66.580	52.443	68.750	76.000
	CONV	B4B	N/A	N/A	36.540	15.420	19.587	15.900	16.600
	A2	B4B	0.1	9216	79.460	56.700	38.795	55.575	**68.000**
			0.3	13120	79.780	**56.880**	41.395	53.688	63.600
			0.5	15680	**79.900**	56.760	40.220	**56.900**	64.200
			0.7	17408	79.680	58.500	**44.426**	56.513	65.000
			1.0	28352	77.770	54.300	39.486	52.888	66.800

$\mathcal{N}_q$ can be calculated using the following formula:

$$\mathcal{N}_q = \frac{T - \mathcal{N}_{td}}{n} + \mathcal{N}_{td},$$

where T represents the total query budget, from the results, we observe a significant improvement in the accuracy of the downstream task accuracy. This shows that our attack greatly reduces the effectiveness of the B4B defense. Moreover, compared to downscaling, noise reduction does not hinder the effective learning of the stolen encoder within the high-resolution embedding space. This results in improved classification accuracy in the downstream task of IMGNET-C10.

Embedding Unification We simulate Sybil attacks with embedding unification and measure the performance of the stolen encoder under varying numbers of attacker users. For Affine, Padding, and Shuffle transformations, the embeddings can be perfectly unified, and the downstream task accuracies are reported in Table 4. We observed a significant increase in downstream task classification accuracy as the number of users increased from 1 to 8. However, as the user

Table 4. Downstream Task Accuracy of Stolen Encoder Utilizing Sybil Attack in SimSiam. The query budget is set to 100k and is evenly split across all users.

DATASET \ USER	1	2	4	8	16	32	64
F-MNIST	36.540	61.560	81.220	83.090	83.800	83.570	84.060
CIFAR10	15.420	45.130	61.810	62.370	64.340	66.450	65.240
SVHN	19.587	30.501	47.960	43.289	50.884	49.927	47.760
STL10	15.900	40.038	56.250	65.025	67.338	68.600	67.650
IMGNET-C10	16.600	56.800	67.600	74.000	77.400	77.200	77.400

count continued to rise, the accuracy began to plateau. This plateauing can be explained by the exponential cost function of the B4B defense. When the number of users increased from 1 to 8, the total query budget was distributed among them, leading to a reduced query volume and, consequently, lower bucket occupancy. This reduction resulted in less noise applied to each user's output representations. As the user count increased further, the penalty imposed by the cost function became negligible, essentially rendering its impact on the representations insignificant, which resulted in relatively stable downstream task accuracy. For the Binary case, we assess the effectiveness of Sybil attacks with embedding unification by calculating the average L1 distance between stolen and victim encoder embeddings (results in Appendix C).

6 Conclusions

In this work, we have highlighted critical vulnerabilities in existing defense mechanisms against model stealing attacks in the context of Encoder-as-a-Service systems. Specifically, we demonstrated that the B4B defense, while a promising approach, is not as robust as previously claimed. Our proposed defense-penetrating attack successfully bypasses B4B's key strategies, including perturbation, query monitoring, and embedding transformations, allowing attackers to extract high-quality representations with minimal performance loss. Through extensive experimentation, we have shown that the stolen encoder performs nearly as well as the original model, underlining the significant weaknesses in current defense strategies. These findings highlight the necessity for enhanced security measures in the rapidly evolving landscape of AI services. They also suggest that a reassessment of current defense frameworks may be beneficial in addressing the increasing sophistication of emerging threats.

7 Acknowledgements

This work was supported in part by the National Key Research and Development Program of China under Grant 2023YFB2704300, in part by the Guangzhou Basic and Applied Basic Research Foundation under Grant 2024A03J0324, and in part by the Science and Technology Projects in Guangzhou 2024 Guangzhou School (Institute) Enterprise Joint Funding Project under Grant 2024A03J0166.

References

1. Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., Joulin, A.: Emerging properties in self-supervised vision transformers. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 9650–9660 (2021)
2. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. pp. 380–388 (2002)

3. Chen, T., Kornblith, S., Norouzi, M., Hinton, G.: A simple framework for contrastive learning of visual representations. In: International conference on machine learning. pp. 1597–1607. PMLR (2020)
4. Chen, X., He, K.: Exploring simple siamese representation learning. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 15750–15758 (2021)
5. Cong, T., He, X., Zhang, Y.: Sslguard: A watermarking scheme for self-supervised learning pre-trained encoders. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 579–593 (2022)
6. Dubiński, J., Pawlak, S., Boenisch, F., Trzcinski, T., Dziedzic, A.: Bucks for buckets (b4b): Active defenses against stealing encoders. Advances in Neural Information Processing Systems **36** (2024)
7. Dziedzic, A., Dhawan, N., Kaleem, M.A., Guan, J., Papernot, N.: On the difficulty of defending self-supervised learning against model extraction. In: International Conference on Machine Learning. pp. 5757–5776. PMLR (2022)
8. Dziedzic, A., Duan, H., Kaleem, M.A., Dhawan, N., Guan, J., Cattan, Y., Boenisch, F., Papernot, N.: Dataset inference for self-supervised models. Advances in Neural Information Processing Systems **35**, 12058–12070 (2022)
9. Dziedzic, A., Duan, H., Kaleem, M.A., Dhawan, N., Guan, J., Cattan, Y., Boenisch, F., Papernot, N.: Dataset inference for self-supervised models. Advances in Neural Information Processing Systems **35**, 12058–12070 (2022)
10. Frosst, N., Papernot, N., Hinton, G.: Analyzing and improving representations with the soft nearest neighbor loss. In: International conference on machine learning. pp. 2012–2020. PMLR (2019)
11. Gong, X., Chen, Y., Yang, W., Mei, G., Wang, Q.: Inversenet: Augmenting model extraction attacks with training data inversion. In: IJCAI. pp. 2439–2447 (2021)
12. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: International Conference on Learning Representations (ICLR) (2015)
13. He, K., Fan, H., Wu, Y., Xie, S., Girshick, R.: Momentum contrast for unsupervised visual representation learning. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 9729–9738 (2020)
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
15. Karmakar, P., Basu, D.: Marich: A query-efficient distributionally equivalent model extraction attack. Advances in Neural Information Processing Systems **36** (2024)
16. Li, Y., Jiang, Y., Li, Z., Xia, S.T.: Backdoor learning: A survey. IEEE Transactions on Neural Networks and Learning Systems **35**(1), 5–22 (2022)
17. Liu, Y., Jia, J., Liu, H., Gong, N.Z.: Stolenencoder: stealing pre-trained encoders in self-supervised learning. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 2115–2128 (2022)
18. Lv, P., Li, P., Zhu, S., Zhang, S., Chen, K., Liang, R., Yue, C., Xiang, F., Cai, Y., Ma, H., et al.: Ssl-wm: A black-box watermarking approach for encoders pre-trained by self-supervised learning. In: 31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024 (2024)
19. Oliynyk, D., Mayer, R., Rauber, A.: I know what you trained last summer: A survey on stealing machine learning models and defences. ACM Computing Surveys **55**(14s), 1–41 (2023)
20. Oord, A.v.d., Li, Y., Vinyals, O.: Representation learning with contrastive predictive coding. arXiv preprint arXiv:1807.03748 (2018)

21. Orekondy, T., Schiele, B., Fritz, M.: Knockoff nets: Stealing functionality of black-box models. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 4954–4963 (2019)
22. Paulevé, L., Jégou, H., Amsaleg, L.: Locality sensitive hashing: A comparison of hash function types and querying mechanisms. Pattern recognition letters **31**(11), 1348–1358 (2010)
23. Sha, Z., He, X., Yu, N., Backes, M., Zhang, Y.: Can't steal? cont-steal! contrastive stealing attacks against image encoders. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16373–16383 (2023)
24. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: 2017 IEEE symposium on security and privacy (SP). pp. 3–18. IEEE (2017)
25. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2818–2826 (2016)
26. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction APIs. In: 25th USENIX security symposium (USENIX Security 16). pp. 601–618 (2016)
27. Yuan, X., Chen, K., Huang, W., Zhang, J., Zhang, W., Yu, N.: Data-free hard-label robustness stealing attack. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 38, pp. 6853–6861 (2024)
28. Zhu, L., Liao, B., Zhang, Q., Wang, X., Liu, W., Wang, X.: Vision mamba: Efficient visual representation learning with bidirectional state space model. In: International Conference on Machine Learning (ICML) (2024)

A The Defense-Penetrating Attack

A.1 Spherical Variance after Downscaling

The relationship between downscaling and clustering of embeddings can also be observed in Table 5. Here we normalize embeddings into unit vectors, and then calculate the corresponding spherical variance, which serves as an indicator of the degree of clustering of points on a hypersphere. We can see that a smaller downscaling parameter yields a lower spherical variance, indicating greater point clustering.

Table 5. Spherical Variance for Different Downscaling Parameters.

Downscaling	4×4	16×16	32×32	96×96	224×224
Spherical Variance	0.04602	0.12658	0.18371	0.30190	0.34512

A.2 Embedding Unification of Residual Transformations

Padding. The Padding transformation in B4B is defined as choosing a random fixed-length vector ρ_i for each user U_i, then $T_i(r) = r||\rho_i$, where $||$ means concatenation. Unifying this transformation is relatively straightforward:

1. When U_i is created, it queries x_1, x_2 and obtains $y_i^1 = T_i(r_1) = r_1||\rho_i$ and $y_i^2 = T_i(r_2) = r_2||\rho_i$. The adversary finds and records ρ_i that is the longest common vector at the end of y_i^1 and y_i^2.
2. $F_i(y)$ is defined as removing ρ_i from y then appending ρ_0 to the result (or simiply every user including U_0 removes ρ_i).

Shuffle. The Shuffle transformation in B4B is defined as choosing a random permutation π_i for each user U_i, then $T_i(r) = \pi_i(r)$. Unifying this transformation is also easy:

1. When U_0 is created, it queries x and obtains $y_0 = \pi_0(r)$, where $r = f_v(x)$.
2. When U_i is created, it queries the same x and obtains $y_i = \pi_i(r)$.
3. The permutation $\pi_0 \circ \pi_i^{-1}$ can be constructed as an index mapping table L, by iterating through each index j and find k such that $y_i[j] = y_0[k]$, then record $L[j] = k$.
4. $F_i(y)$ is defined as: outputting a new vector such that the $y[j]$ is moved to the k-th position in the new vector, where $k = L[j]$.

B Details on Experimental Setup

B.1 Hyperparameter Settings for B4B Defense Mechanism

The hyperparameters for the B4B defense mechanism include the number of buckets in the LSH and the coefficients λ, α, and β in the cost function. We follow the calibrated hyperparameter values provided in the original B4B paper. Specifically, the number of buckets is set to 2^{12}, consistent with the B4B configuration. For experiments with the SimSiam architecture, the cost function hyperparameters are set as $\lambda = 10^{-6}$, $\alpha = 1$, and $\beta = 0.6$. For experiments with the DINO architecture, the hyperparameters are adjusted to $\lambda = 10^{-6}$, $\alpha = 1000$, and $\beta = 0.6$. The α value is increased by a factor of 1000 for DINO due to its representation norms being approximately 10^3 times larger than those of SimSiam.

B.2 Training Details of the Stolen Encoder

The architecture of the stolen model is also kept consistent with that used in the B4B defense. Specifically, we use ResNet-50 as the backbone and modify its fully connected layer to accommodate different representation output dimensions. The stealing dataset is a subset of ImageNet-1k, containing 100 examples per class. The stolen model is trained using the InfoNCE loss function for 150 epochs, with a batch size of 256. The learning rate is initialized to 0.1 and decays linearly to 0 over the course of training.

B.3 Training Details of the Downstream Classifier

We evaluate the performance of the stolen encoder on five downstream datasets: FashionMNIST, CIFAR10, SVHN, STL10, and IMGNET-C10. These datasets include both simple and complex tasks, providing a comprehensive assessment of the stolen encoder's capabilities. Table 6 summarizes the details of these downstream datasets. For the classification training of downstream tasks, the ResNet-50 backbone is frozen, and only the parameters of the added linear classifier are optimized. Each task is trained for 100 epochs, with a batch size of 256. The learning rate is initialized to 0.1 and decays linearly to 0 throughout the training process.

C Evaluation of Binary's Unification

The results, presented in Table 7, demonstrate that as the number of users increases, the L1 distance decreases. This signifies that the stolen encoder becomes increasingly similar to the victim encoder. These findings suggest that (1) embeddings from multiple users can be effectively unified by our algorithm and utilized collectively to train the stolen encoder, and (2) the Sybil attack successfully mitigates the noise introduced by B4B in the embedding process. Consequently, our attack proves to be effective.

Table 6. Dataset Specifications. IMGNET-C10 is a custom subset of ImageNet containing 10 randomly sampled classes. Each class contains 1000 training samples and 300 testing samples. The ImageNet dataset is a complex dataset with variable resolution, so we resized the resolution to 224 in our downstream classification experiment.

DATASET	CLASS	RESOLUTION	TRAIN	TEST
F-MNIST	10	$1 \times 28 \times 28$	60,000	10,000
CIFAR10	10	$3 \times 32 \times 32$	50,000	10,000
SVHN	10	$3 \times 32 \times 32$	73,257	26,032
STL10	10	$3 \times 96 \times 96$	5,000	8,000
IMGNET-C10	10	$3 \times 224 \times 224$	10,000	3,000

Table 7. The Average L1 Distance Between the Binary Embeddings Generated by the Stolen Encoder and the Victim Encoder under Sybil Attack in SimSiam. The query cost is set to 100k and is evenly split across all users.

DATASET \ USER	1	2	4	8	16	32	64
F-MNIST	1620.40	831.71	203.55	5.93	4.96	4.26	4.27
CIFAR10	1440.97	375.85	73.98	4.04	4.47	3.66	3.16
SVHN	1350.33	221.24	114.85	6.93	5.92	5.49	4.70
STL10	1482.12	750.10	44.63	4.61	4.72	4.55	4.17
IMGNET-C10	395.21	342.58	46.63	35.74	40.61	33.88	23.15

D Results for DINO

Table 8. Downstream Task Accuracy of Stolen Encoder Using A1 in DINO. QUERIES stands for the number of queries used for stealing. DEFENSE indicates whether the victim encoder has deployed B4B. DOWNSCALING column shows the downscaling parameters.

QUERIES	DEFENSE	DOWN-SCALING	F-MNIST	CIFAR10	SVHN	STL10	IMGNET-C10
50K	NONE	w/o	89.490	76.040	67.014	78.863	83.000
	NONE	32×32	89.700	74.670	67.701	67.763	55.600
	B4B	w/o	86.050	62.270	43.047	62.975	38.000
	B4B	32×32	**89.450**	**75.140**	**67.939**	**68.438**	**58.800**
	B4B	48×48	87.240	65.680	60.391	64.812	55.400
100K	NONE	w/o	90.580	79.110	72.388	83.175	87.200
	NONE	32×32	90.590	80.540	72.568	71.800	63.400
	B4B	w/o	86.610	61.680	39.478	63.138	27.000
	B4B	32×32	**90.650**	**80.120**	**71.458**	**72.800**	**62.000**
	B4B	48×48	85.680	61.340	50.196	59.313	47.000

Table 9. Downstream Task Accuracy of Stolen Encoder Using A2 in DINO. The ATTACK column lists the adversarial attack methods, where CONV denotes the conventional attack, and A2 means noise reduction is used. $\mathcal{N}_{td}$ is the number of queried samples when triggering the noise reduction mechanism. We fix the hyperparameter $\mathcal{P}_\tau = 0.05$, $n = 8$ while varying $\mathcal{R}$.

QUERIES	ATTACK	DEFENSE	$\mathcal{R}$	$\mathcal{N}_{td}$	F-MNIST	CIFAR10	SVHN	STL10	IMGNET-C10
50K	CONV	NONE	N/A	N/A	89.490	76.040	67.014	78.863	83.000
	CONV	B4B	N/A	N/A	86.050	62.270	43.047	62.975	38.000
	A2	B4B	0.1	7260	88.630	65.830	57.621	66.138	67.800
			0.3	8288	88.320	66.360	57.925	**68.375**	65.000
			0.5	9537	88.630	67.320	59.027	67.863	65.000
			0.7	10164	88.580	**67.490**	58.920	68.050	**71.800**
			1.0	11253	**88.770**	65.720	**59.223**	67.413	65.800
100K	CONV	NONE	N/A	N/A	90.580	79.110	72.388	83.175	87.200
	CONV	B4B	N/A	N/A	87.200	61.680	39.478	63.138	27.000
	A2	B4B	0.1	7260	88.830	67.040	54.648	**68.325**	66.000
			0.3	8288	89.710	**69.000**	57.011	68.275	66.000
			0.5	9537	88.150	66.970	55.789	67.438	**66.400**
			0.7	10164	**88.880**	67.100	**57.545**	67.863	55.800
			1.0	11253	88.200	67.560	56.684	67.775	61.600

Table 10. Downstream Task Accuracy of Stolen Encoder Utilizing Sybil Attack in DINO. The query budget is set to 100k and is evenly split across all users.

DATASET \ USER	1	2	4	8	16	32	64
F-MNIST	88.610	86.980	88.470	90.560	90.210	90.810	90.460
CIFAR10	61.680	66.970	73.140	77.860	78.520	78.840	78.970
SVHN	39.478	50.038	57.084	69.645	69.645	72.772	72.837
STL10	63.138	67.888	75.312	81.738	82.575	82.963	82.650
IMGNET-C10	27.000	58.800	77.400	84.800	84.800	87.000	87.800