REaaS: Enabling Adversarially Robust Downstream Classifiers via Robust Encoder as a Service

Wenjie Qu[1], Jinyuan Jia[2], Neil Zhenqiang Gong[3]
[1] Huazhong University of Science and Technology, wen_jie_qu@outlook.com
[2] University of Illinois Urbana-Champaign, jinyuan@illinois.edu
[3] Duke University, neil.gong@duke.edu

***Abstract*—*Encoder as a service* is an emerging cloud service. Specifically, a service provider first pre-trains an encoder (i.e., a general-purpose feature extractor) via either supervised learning or self-supervised learning and then deploys it as a cloud service API. A client queries the cloud service API to obtain feature vectors for its training/testing inputs when training/testing its classifier (called *downstream classifier*). A downstream classifier is vulnerable to *adversarial examples*, which are testing inputs with carefully crafted perturbation that the downstream classifier misclassifies. Therefore, in safety and security critical applications, a client aims to build a robust downstream classifier and certify its robustness guarantees against adversarial examples.**

What APIs should the cloud service provide, such that a client can use any certification method to certify the robustness of its downstream classifier against adversarial examples while minimizing the number of queries to the APIs? How can a service provider pre-train an encoder such that clients can build more certifiably robust downstream classifiers? We aim to answer the two questions in this work. For the first question, we show that the cloud service only needs to provide two APIs, which we carefully design, to enable a client to certify the robustness of its downstream classifier with a minimal number of queries to the APIs. For the second question, we show that an encoder pre-trained using a spectral-norm regularization term enables clients to build more robust downstream classifiers.

I. Introduction

In an *encoder as a service*, a service provider (e.g., OpenAI, Google, and Amazon) pre-trains a general-purpose feature extractor (called *encoder*) and deploys it as a cloud service; and a client queries the cloud service APIs for the feature vectors of its training/testing inputs when training/testing a downstream classifier. For instance, the encoder could be pre-trained using supervised learning on a large amount of labeled data or self-supervised learning [1], [2] on a large amount of unlabeled data. A client could be a smartphone, IoT device, self-driving car, or edge device in the era of *edge computing*. Encoder as a service has been widely deployed by industry, e.g., OpenAI's GPT-3 API [3] and Clarifai's General Embedding API [4]. In the *Standard Encoder as a Service (SEaaS)*, the service provides a single API (called *Feature-API*) for clients and the encoder is pre-trained without taking the robustness of downstream classifiers into consideration. A client sends its training/testing inputs to the Feature-API, which returns their feature vectors to the client.

Wenjie Qu performed this research when he was an intern in Gong's group.

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.24444
www.ndss-symposium.org

A downstream classifier is vulnerable to *adversarial examples* [5], [6]. Suppose a testing input is correctly classified by the downstream classifier. An attacker adds a small carefully crafted perturbation to the testing input to induce misclassification. Such testing input with carefully crafted perturbation is called an adversarial example. Therefore, in security and safety critical applications such as user authentication and traffic sign recognition, a client desires to build a downstream classifier robust against adversarial examples. Many methods have been developed for an attacker to craft adversarial examples and the community keeps developing new, stronger ones. Therefore, instead of defending against a specific class of adversarial examples, a client aims to defend against all bounded adversarial perturbations via building a *certifiably robust* downstream classifier. A classifier is certifiably robust if its predicted label for a testing input is unaffected by arbitrary perturbation added to the testing input once its size (measured by ℓ_2-norm in this work) is less than a threshold, which is known as *certified radius*. A larger certified radius indicates better certified robustness against adversarial examples.

In general, there are two categories of complementary methods to build a certifiably robust classifier and derive its certified radius for a testing input, i.e., *base classifier (BC) based certification* [7], [8], [9], [10] and *smoothed classifier (SC) based certification* (also known as *randomized smoothing*) [11], [12], [13]. BC based certification aims to directly derive the certified radius of a given classifier (called *base classifier*) for a testing input. BC based certification requires white-box access to the base classifier as it often requires propagating the perturbation from the input layer to the output layer of the base classifier layer by layer. SC based certification first builds a *smoothed classifier* based on the base classifier via adding random noise (e.g., Gaussian noise) to a testing input and then derives the certified radius of the smoothed classifier for the testing input. To increase the testing inputs' certified radii, SC based certification often requires training the base classifier using training inputs with random noise. Moreover, to derive the predicted label and certified radius for a testing input, SC based certification requires the base classifier to predict the labels of multiple noisy versions of the testing input.

SEaaS faces three challenges when a client aims to build a certifiably robust downstream classifier and derive its certified

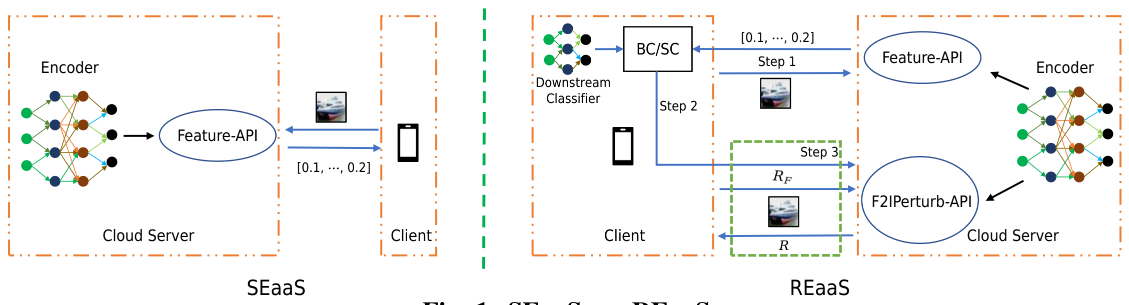


Fig. 1: SEaaS vs. REaaS.

radii for testing inputs. The first challenge is that a client cannot use BC based certification. In particular, the composition of the encoder and the client's downstream classifier is the base classifier that the client needs to certify in BC based certification. However, the client does not have white-box access to the encoder deployed on the cloud server, making BC based certification not applicable. The second challenge is that, although a client can use SC based certification by treating the composition of the encoder and its downstream classifier as a base classifier, it incurs a large communication cost for the client and a large computation cost for the cloud server. Specifically, the client needs to query the Feature-API once for each noisy training input in each training epoch of the downstream classifier because SC based certification trains the base classifier using noisy training inputs. Therefore, the client requires e queries to the Feature-API *per* training input, where e is the number of epochs used to train the downstream classifier. Moreover, to derive the predicted label and certified radius for a testing input, SC based certification requires the base classifier to predict the labels of N noisy testing inputs. Therefore, the client requires N queries to the Feature-API *per* testing input. Note that N is often a large number (e.g., 10,000) [13]. The large number of queries to the Feature-API imply 1) large communication cost, which is intolerable for resource-constrained clients such as smartphone and IoT devices, and 2) large computation cost for the cloud server. The third challenge is that SC based certification achieves suboptimal certified radii. This is because the base classifier is the composition of the encoder and a client's downstream classifier, but a client cannot train/fine-tune the encoder as it is deployed on the cloud server.

Our work: We propose *Robust Encoder as a Service (REaaS)* to address the three challenges of SEaaS. Figure 1 compares SEaaS with REaaS. Our key idea is to provide another API called *F2IPerturb-API*.[1] A downstream classifier essentially takes a feature vector as input and outputs a label. Our F2IPerturb-API enables a client to treat its downstream classifier *alone* as a base classifier and certify the robustness of its base or smoothed downstream classifier in the *feature space*. Specifically, a client performs three steps to derive the certified radius of a testing input in REaaS. First, the client obtains the feature vector of the testing input via querying the Feature-API. Second, the client views its downstream classifier alone as a base classifier and derives a *feature-space certified radius* R_F for the testing input using any BC/SC certification method. The client's base or smoothed downstream classifier predicts the same label for the testing input if the ℓ_2-norm of the adversarial perturbation added to the testing input's feature vector is less than R_F. Third, the client sends the testing input and its feature-space certified radius R_F to query the F2IPerturb-API, which returns the corresponding *input-space certified radius* R to the client. Our input-space certified radius R guarantees the client's base or smoothed downstream classifier predicts the same label for the testing input if the ℓ_2-norm of the adversarial perturbation added to the testing input is less than R.

The key challenge of implementing our F2IPerturb-API is how to find the largest input-space certified radius R for a given testing input and its feature-space certified radius R_F. To address the challenge, we formulate finding the largest R as an optimization problem, where the objective function is to find the maximum R and the constraint is that the feature-space perturbation is less than R_F. However, the optimization problem is challenging to solve due to the highly non-linear constraint. To address the challenge, we propose a binary search based solution. The key component of our solution is to check whether the constraint is satisfied for a specific R in each iteration of binary search. Towards this goal, we derive an upper bound of the feature-space perturbation for a given R and we treat the constraint satisfied if the upper bound is less than R_F. Our upper bound can be computed efficiently.

F2IPerturb-API addresses the first two challenges of SEaaS. Specifically, BC based certification is applicable in REaaS. Moreover, SC based certification requires much less queries to the APIs in REaaS. Specifically, for any certification method, a client only requires one query to Feature-API per training input and two queries (one to Feature-API and one to F2IPerturb-API) per testing input in our REaaS.

To address the third challenge of SEaaS, we propose a new method to pre-train a robust encoder, so a client can derive larger certified radii even though it cannot train/fine-tune the encoder. Our method can be combined with standard supervised learning or self-supervised learning to enhance the robustness of a pre-trained encoder. An encoder is more robust if it produces more similar feature vectors for an input and its adversarially perturbed version. Our key idea is to derive an upper bound for the Euclidean distance between the feature vectors of an input and its adversarial version, where our upper bound is a product of a *spectral-norm term* and the perturbation size. The spectral-norm term depends on the parameters of the encoder, but it does not depend on the input nor the adversarial perturbation. An encoder with a smaller spectral-norm term may produce more similar feature vectors for an input and its adversarial version. Thus, we use the spectral-norm term as a regularization term to regularize the pre-training of an encoder.

We perform a systematic evaluation on multiple datasets including CIFAR10, SVHN, STL10, and Tiny-ImageNet. Our

[1]'F' stands for Feature and 'I' stands for Input.

evaluation results show that REaaS addresses the three challenges of SEaaS. First, REaaS makes BC based certification applicable. Second, REaaS incurs orders of magnitude less queries to the cloud service than SEaaS for SC based certification. For instance, REaaS reduces the number of queries to the cloud service APIs respectively by $25\times$ and $5,000\times$ per training and testing input when a client trains its downstream classifier for $e = 25$ epochs and uses $N = 10,000$ for certification. Third, in the framework of REaaS, our robust pre-training method achieves larger *average certified radius (ACR)* for the testing inputs than existing methods to pre-train encoders for both BC and SC based certification. For instance, when the encoder is pre-trained on Tiny-ImageNet and the downstream classifier is trained on SVHN, the ACRs for MoCo (a standard non-robust self-supervised learning method) [1], RoCL (an adversarial training based state-of-the-art robust self-supervised learning method) [14], and our method are respectively 0.011, 0.014, and 0.275 when a client uses SC based certification.

In summary, we make the following contributions:

- We propose REaaS, which enables a client to build a certifiably robust downstream classifier and derive its certified radii using any certification method with a minimal number of queries to the cloud service.
- We propose a method to implement F2IPerturb-API.
- We propose a spectral-norm term to regularize the pre-training of a robust encoder.
- We extensively evaluate REaaS and compare it with SEaaS on multiple datasets.

II. Related Work

A. Adversarial Examples

We discuss adversarial examples [5], [15] in the context of encoder as a service. We denote by f a pre-trained encoder and g a downstream classifier. Given a testing input $\mathbf{x}$, the encoder outputs a feature vector for it, while the downstream classifier takes the feature vector as input and outputs a label. For simplicity, we denote by $f(\mathbf{x})$ the feature vector and $g \circ f(\mathbf{x})$ the predicted label for $\mathbf{x}$, where $\circ$ represents the composition of the encoder and downstream classifier. In an adversarial example, an attacker adds a carefully crafted small perturbation δ to $\mathbf{x}$ such that its predicted label changes, i.e., $g \circ f(\mathbf{x} + \delta) \neq g \circ f(\mathbf{x})$. The carefully perturbed input $\mathbf{x} + \delta$ is called an adversarial example. Many methods (e.g., [5], [6], [16]) have been proposed to find an adversarial perturbation δ for a given input $\mathbf{x}$. In our work, we focus on certified defenses, which aim to defend against any bounded adversarial perturbations no matter how they are found. Therefore, we omit the details on how an attacker can find an adversarial perturbation.

B. Certifying Robustness of a Classifier

Definition of certified radius: A classifier is certifiably robust against adversarial examples if its predicted label for an input is unaffected by any perturbation once its size is bounded [7], [12], [13]. Formally, a classifier h is certifiably robust if we have the following guarantee for an input $\mathbf{x}$:

$$h(\mathbf{x} + \delta) = h(\mathbf{x}), \forall \|\delta\|_2 < R, \tag{1}$$

where R is known as *certified radius*. Note that certified radius R may be different for different inputs $\mathbf{x}$, but we omit the explicit dependency on $\mathbf{x}$ in the notation for simplicity.

A certification method against adversarial examples aims to build a certifiably robust classifier and derive its certified radius R for any input $\mathbf{x}$. There are two general categories of certification methods, i.e., *base classifier (BC) based certification* [7], [8], [9], [10] and *smoothed classifier (SC) based certification* [11], [12], [13]. Both categories of methods may be adopted in different scenarios depending on certification needs. On one hand, BC based certification often produces *deterministic* guarantees (i.e., the derived certified radius is absolutely correct), while SC based certification often provides *probabilistic* guarantees (i.e., the derived certified radius may be incorrect with a small *error probability*). On the other hand, SC based certification often derives a larger certified radius than BC based certification due to its probabilistic guarantees.

Base classifier (BC) based certification: BC based certification aims to directly derive the certified radius R of a given classifier (called *base classifier*) for an input $\mathbf{x}$. These methods often propagate perturbation from the input $\mathbf{x}$ to the output of the base classifier layer by layer in order to derive the certified radius. Therefore, they require white-box access to the base classifier. Suppose F is a base classifier that maps an input $\mathbf{x}$ to one of c classes $\{1, 2, \cdots, c\}$. We use $H(\mathbf{x})$ to denote the base classifier's last-layer output vector for $\mathbf{x}$, where $H_l(\mathbf{x})$ represents the lth entry of $H(\mathbf{x})$ and $l = 1, 2, \cdots, c$. $F(\mathbf{x})$ denotes the predicted label for $\mathbf{x}$, i.e., $F(\mathbf{x}) = \text{argmax}_{l=1,2,\cdots,c} H_l(\mathbf{x})$. Next, we overview how to derive the certified radius R using CROWN [9], a state-of-the-art BC based certification method. CROWN shows that each entry $H_l(\mathbf{x})$ can be bounded by two linear functions $H_l^L(\mathbf{x})$ and $H_l^U(\mathbf{x})$. Suppose the base classifier predicts label y for $\mathbf{x}$ when there is no adversarial perturbation, i.e., $F(\mathbf{x}) = y$. CROWN finds the largest r such that the lower bound of the yth entry (i.e., $\min_{\|\delta\|_2 < r} H_y^L(\mathbf{x} + \delta)$) is larger than the upper bounds of all other entries (i.e., $\max_{l \neq y} \max_{\|\delta\|_2 < r} H_l^U(\mathbf{x} + \delta)$) and views it as the certified radius R for the input $\mathbf{x}$. The complete details of CROWN can be found in Appendix B. In the context of encoder as a service, the composition of the encoder and downstream classifier (i.e., $g \circ f$) is a base classifier F, whose certified radius a client aims to derive. However, in SEaaS, a client does not have white-box access to the encoder g since it is deployed on the cloud server. As a result, a client cannot use BC based certification to derive the certified radius of $g \circ f$.

Smoothed classifier (SC) based certification: SC based certification first builds a *smoothed classifier* based on the base classifier and then derives the certified radius R of the smoothed classifier. In SEaaS, a client builds a smoothed classifier h based on the base classifier $g \circ f$ via adding random Gaussian noise $\mathcal{N}(0, \sigma^2\mathbf{I})$ to an input $\mathbf{x}$, where σ is the standard deviation of the Gaussian noise. Specifically, given a testing input $\mathbf{x}$, the client constructs N noisy inputs $\mathbf{x} + \mathbf{n}_1, \mathbf{x} + \mathbf{n}_2, \cdots, \mathbf{x} + \mathbf{n}_N$, where $\mathbf{n}_i$ $(i = 1, 2, \cdots, N)$ is sampled from $\mathcal{N}(0, \sigma^2\mathbf{I})$. The client uses the base classifier $g \circ f$ to predict the label of each noisy input. Moreover, the client computes the *label frequency* N_l of each label l among the noisy inputs, i.e., $N_l = \sum_{j=1}^{N} \mathbb{I}(g \circ f(\mathbf{x} + \mathbf{n}_j) = l)$, where $\mathbb{I}$ is an indicator function. The smoothed classifier predicts the label with the largest label frequency for the original testing input $\mathbf{x}$. Moreover, the client can derive the certified radius R of the smoothed classifier for $\mathbf{x}$ based on the label frequencies. Due to the random sampling, the derived certified radius may be incorrect with an error probability α, which can be set by the client. In Appendix C, we take Cohen et al. [13] as an example to discuss more technical details on SC based certification.

To improve certified radius, the base classifier $g \circ f$ is often trained using noisy training inputs [13]. In encoder as a service (both SEaaS and REaaS), a client does not have white-box access to the encoder g and thus can only train its downstream classifier f using noisy training inputs. Specifically, for SEaaS, in each epoch of training the downstream classifier, a client adds random Gaussian noise from $\mathcal{N}(0, \sigma^2\mathbf{I})$ to each training input, queries the Feature-API to obtain the feature vector of each noisy training input, and uses the feature vectors to update the downstream classifier via stochastic gradient descent.

SC based certification faces two challenges in SEaaS. First, a client needs to query the cloud service many times, leading to a large communication cost for the client and a large computation cost for the cloud server. Specifically, for each testing input $\mathbf{x}$, a client needs to query the Feature-API N times to obtain the feature vectors of the N noisy inputs in order to compute the label frequencies. Moreover, the client queries the Feature-API e times *per* training input, where e is the number of epochs used to train the downstream classifier. We note that [17] proposed to prepend a denoiser to a base classifier instead of training it with noisy training inputs, which can reduce the number of queries from e to 1 per training input when applied to SEaaS. However, it is hard for a client with a small amount of data to train such a denoiser. Second, the derived certified radius is suboptimal because a client cannot fine-tune the encoder, which is not pre-trained to support certified robustness.

C. Pre-training an Encoder

1) Pre-training Non-robust Encoders: We discuss both standard supervised learning and self-supervised learning methods to pre-train encoders, which do not take robustness against adversarial examples into consideration.

Supervised learning: The idea of using supervised learning to pre-train an encoder is to first train a deep neural network classifier using labeled training data and then use the layers excluding the output layer as an encoder. Specifically, supervised learning defines a loss function $l(i)$ (e.g., cross-entropy loss) for each labeled training example $(\mathbf{x}_i, y_i)$, where y_i is the ground truth label of $\mathbf{x}_i$. Then, supervised learning iteratively trains a deep neural network classifier by minimizing the sum of the losses over the labeled training examples. Such paradigm of training a deep neural network classifier via supervised learning and using the layers excluding the output layer as an encoder is also known as *transfer learning* [18].

Self-supervised learning: Unlike supervised learning, self-supervised learning [1], [2] aims to pre-train an encoder using unlabeled data, which has attracted growing attention in the past several years in the AI community. A basic component of self-supervised learning is *data augmentation*. Specifically, given an image, the data augmentation component applies a series of (random) data augmentation operations (e.g., random cropping, color jitter, and flipping) sequentially to produce an *augmented image*. Roughly speaking, the main idea of self-supervised learning is to pre-train an encoder such that it produces similar feature vectors for two augmented images produced from the same image, but dissimilar feature vectors for two augmented images produced from different images. Next, we take MoCo [1], a state-of-the-art self-supervised learning algorithm, as an example to elaborate more details.

MoCo uses an auxiliary encoder (called *momentum encoder*) that has the same architecture as the encoder and a queue (denoted as Γ). In particular, the queue is used to cache the output of the momentum encoder for the augmented images and is dynamically updated. For simplicity, we respectively use f and f_e to denote the encoder and the momentum encoder, and use θ and θ_e to denote their encoder parameters. Suppose we have a mini-batch of unlabeled images which are denoted as $\mathbf{x}_i, i = 1, 2, \cdots, m$. We apply the data augmentation component to each image in the mini-batch twice. We use $\mathbf{x}_i^1$ and $\mathbf{x}_i^2$ to denote the two augmented images produced for the image $\mathbf{x}_i$, respectively. Given $\mathbf{x}_i^1$, $\mathbf{x}_i^2$, and the queue Γ, MoCo defines a loss function for $\mathbf{x}_i$ as follows:

$$\ell(i) = -\log\left(\frac{\exp(Sim(f(\mathbf{x}_i^1), f_e(\mathbf{x}_i^2))/\tau)}{\sum_{\mathbf{z}\in\Gamma\cup\{f_e(\mathbf{x}_i^2)\}} \exp(Sim(f(\mathbf{x}_i^1), \mathbf{z})/\tau)}\right), \quad (2)$$

where τ is a temperature parameter and $Sim(\cdot, \cdot)$ measures the similarity of two feature vectors (e.g., cosine similarity). MoCo uses gradient descent to minimize $\frac{1}{m} \cdot \sum_{i=1}^{m} \ell(i)$ to update the parameters θ in the encoder f. The queue Γ is dynamically updated in each step, where $\{f_e(\mathbf{x}_1^2), f_e(\mathbf{x}_2^2), \cdots, f_e(\mathbf{x}_m^2)\}$ are enqueued and the m "oldest" vectors are dequeued.

2) Pre-training Empirically Robust Encoders: Adversarial training [15], [16] is a standard method to train empirically robust classifiers in supervised learning. The key idea is to generate adversarial examples based on training examples during training, and use the adversarial examples to augment the training data. The layers excluding the output layer of the classifier are then used as an encoder. Several studies [19], [14], [20] generalized adversarial training to pre-train a robust encoder in self-supervised learning. Roughly speaking, the idea is to first generate adversarial examples that incur large loss and then use them to pre-train an encoder. For instance, Kim et al. [14] proposed *Robust Contrastive Learning (RoCL)* to pre-train robust encoders. Specifically, given a training image, RoCL uses Projected Gradient Descent (PGD) [16] to generate an adversarial perturbation for an augmented version of the training image such that the adversarially perturbed augmented version incurs a large loss. Then, RoCL pre-trains an encoder such that it outputs similar feature vectors for the adversarially perturbed augmented version and other augmented versions of the training image.

The goal of these studies is to pre-train *empirically* rather than *certifiably* robust encoders. As a result, the pre-trained encoders achieve suboptimal certified radii for a client as shown by our experimental results. In this work, we propose a new method, which can be combined with either supervised learning or self-supervised learning to pre-train a robust encoder that can achieve larger certified radii against adversarial examples for a client.

III. Problem Formulation

Threat model: We consider an attacker can use adversarial examples to induce misclassification for a client. Specifically, given a testing input, an attacker can add a carefully crafted perturbation to it such that the client's downstream classifier predicts a different label. To defend against adversarial examples, the client aims to build a certifiably robust classifier, which provably predicts the same label for a testing input no matter what perturbation is added to it once the ℓ_2-norm of the perturbation is less than a threshold (called *certified radius*). Note that we do not constrain on what method an attacker can use to find the perturbation since we aim to defend against all bounded perturbations.

Problem definition: We aim to design an encoder as a service. In particular, when designing an encoder as a service, we essentially aim to answer two key questions: 1) what APIs should the cloud service provide for a client? and 2) how to pre-train the encoder?

Design goals: We aim to design an encoder as a service to achieve three goals, *generality*, *efficiency*, and *robustness*, which we elaborate in the following:

- **Generality.** As we discussed in Section II-B, BC and SC based certification methods are complementary and may be adopted by different clients due to their different needs. We say an encoder as a service achieves the generality goal if a client can use any certification method to build a certifiably robust classifier and derive its certified radius for any given testing input. We note that SEaaS can only support SC based certification.
- **Efficiency.** We use the number of queries sent to the cloud service to measure the communication cost between a client and the cloud server. Moreover, we use computation time to measure the computation cost for a client and the cloud server. We aim to design an encoder as a service to achieve a small communication cost and computation cost.
- **Robustness.** A certified radius measures the certified robustness of a classifier for a testing input. We use the average certified radius of testing inputs to measure the certified robustness of a classifier. We aim to design an encoder as a service that enables a client to build a downstream classifier with a large average certified radius in both BC and SC based certification.

We note that SEaaS does not achieve any of the three goals. Specifically, SEaaS cannot support BC based certification; SEaaS incurs a large communication cost between a client and the cloud server as well as a large computation cost for the cloud server in SC based certification; and SEaaS achieves suboptimal average certified radius for SC based certification because certified robustness is not taken into consideration when pre-training the encoder.

IV. Our REaaS

A. Overview

To achieve the generality and efficiency goals, our key idea is to enable a client to treat its own downstream classifier as a base classifier and certify the robustness of its base downstream classifier or smoothed downstream classifier in the *feature space*. Towards this goal, other than the *Feature-API* provided in SEaaS, our REaaS provides another API (called *F2IPerturb-API*). In particular, since a downstream classifier takes a feature vector as input, we propose a client first derives a feature-space certified radius R_F of its base downstream classifier or smoothed downstream classifier for a testing input. Then, the client transforms the feature-space certified radius R_F to the input-space certified radius R by querying the F2IPerturb-API. To achieve the robustness goal, we further propose a new method to pre-train a robust encoder, which uses a spectral-norm term to regularize the pre-training of an encoder. Our pre-trained encoder aims to produce similar feature vectors for an input and its adversarially perturbed version.

B. Feature-API and F2IPerturb-API

1) Feature-API: We first introduce the input and output of Feature-API, and then its implementation.

Input and output for a client: In Feature-API, the input from a client is an image $\mathbf{x}$ and the output returned to the client is the input's feature vector $\mathbf{v}$. Formally, Feature-API is represented as $\mathbf{v} = \textit{Feature-API}(\mathbf{x})$.

Implementation on the server: Given an input $\mathbf{x}$, the cloud server uses a pre-trained encoder f to compute its feature vector $\mathbf{v}$. In particular, we have $\mathbf{v} = f(\mathbf{x})$.

We note that SEaaS only has this Feature-API.

2) F2IPerturb-API: Like Feature-API, we first introduce the input and output of F2IPerturb-API, and then its implementation on the cloud server.

Input and output for a client: F2IPerturb-API transforms a feature-space certified radius to an input-space certified radius. The input from a client contains an input image $\mathbf{x}$ and a feature-space certified radius R_F for $\mathbf{x}$. In particular, the client's downstream classifier predicts the same label for $\mathbf{x}$ once the ℓ_2-norm of the perturbation to $\mathbf{x}$'s feature vector $\mathbf{v}$ is bounded by R_F. The client can use any BC or SC based method to derive R_F for $\mathbf{x}$ by treating its downstream classifier alone as a base classifier and $\mathbf{x}$'s feature vector $\mathbf{v}$ as an "input" to the base classifier.

The output of F2IPerturb-API is an input-space certified radius R such that when the ℓ_2-norm of the adversarial perturbation added to the input image $\mathbf{x}$ is smaller than R, the ℓ_2-norm of the perturbation introduced to the feature vector $\mathbf{v}$ is smaller than R_F. Formally, given an input image $\mathbf{x}$ and a feature-space certified radius R_F, F2IPerturb-API is represented as follows: $R = \textit{F2IPerturb-API}(\mathbf{x}, R_F)$.

Implementation on the server: A larger R enables the client to derive a larger certified radius. The key challenge of implementing the F2IPerturb-API is how to derive the largest R for a given input $\mathbf{x}$ and R_F. To address the challenge, we formulate the input-space certified radius R as the solution to the following optimization problem:

$$R = \max_{r} r \tag{3}$$

$$\textit{s.t.} \max_{\|\delta\|_2 < r} \|f(\mathbf{x}+\delta) - f(\mathbf{x})\|_2 < R_F, \tag{4}$$

Algorithm 1: *F2IPerturb-API*

```
Input from client: image $\mathbf{x}$ and feature-space certified radius $R_F$
Output for client: image-space certified radius $R$
$\rho^L, \rho^U \leftarrow 0$, large value (e.g., 10)
while $\rho^U - \rho^L > \beta$ do
    $\rho_k = \frac{\rho^L + \rho^U}{2}$
    for $i = 1, 2, \cdots, d$ do
        $f_i^L, f_i^U \leftarrow \textsc{Crown}(i, \mathbf{x}, f)$
        $L_i = \min_{\|\delta\|_2 \le \rho_k} f_i^L(\mathbf{x}+\delta) - f_i(\mathbf{x})$
        $U_i = \max_{\|\delta\|_2 \le \rho_k} f_i^U(\mathbf{x}+\delta) - f_i(\mathbf{x})$
    end for
    $R'_F = \sqrt{\sum_{i=1}^d \max(L_i^2, U_i^2)}$
    if $R'_F < R_F$ then
        $\rho^L = \rho_k$
    else
        $\rho^U = \rho_k$
    end if
end while
return $\rho^L$
```

where f is an encoder and δ is an adversarial perturbation. However, the optimization problem is challenging to solve because the constraint is highly non-linear when the encoder is a complex neural network. To address the challenge, we propose a binary search based method to solve R in the optimization problem. In particular, we search in the range $[\rho_k^L, \rho_k^U]$ in the kth round of binary search, where we set ρ_1^L to be 0 and ρ_1^U to be a large value (e.g., 10 in our experiments) in the first round. Moreover, we denote $\rho_k = \frac{\rho_k^L + \rho_k^U}{2}$ for simplicity. In the kth round, we check whether $r = \rho_k$ satisfies the constraint in Equation (4). If the constraint is satisfied, then we can search the range $[\rho_k, \rho_k^U]$ in the $(k+1)$th round, i.e., $\rho_{k+1}^L = \rho_k$ and $\rho_{k+1}^U = \rho_k^U$. Otherwise, we search the range $[\rho_k^L, \rho_k]$ in the $(k+1)$th round, i.e., $\rho_{k+1}^L = \rho_k^L$ and $\rho_{k+1}^U = \rho_k$. We stop the binary search when $\rho_k^U - \rho_k^L \le \beta$ and treat ρ_k^L as R, where β is a parameter characterizing the binary-search precision.

Our binary search based solution faces a key challenge, i.e., how to check whether $r = \rho_k$ satisfies the constraint in Equation (4). Our key idea to address the challenge is to derive an upper bound for the left hand side of the constraint (i.e., $\max_{\|\delta\|_2 < \rho_k} \|f(\mathbf{x}+\delta) - f(\mathbf{x})\|_2$) and decide that the constraint is satisfied if the upper bound is smaller than R_F, where the upper bound can be efficiently computed for any ρ_k. Suppose the encoder f maps an input $\mathbf{x}$ to a d-dimensional feature vector $f(\mathbf{x})$, where $f_i(\mathbf{x})$ represents the ith entry of $f(\mathbf{x})$. An encoder f is essentially a deep neural network. Therefore, according to CROWN [9], we have the following lower bound and upper bound for $f_i(\mathbf{x}+\delta)$ when $\|\delta\|_2 < \rho_k$:

$$\min_{\|\delta\|_2 < \rho_k} f_i^L(\mathbf{x}+\delta) \le f_i(\mathbf{x}+\delta) \le \max_{\|\delta\|_2 < \rho_k} f_i^U(\mathbf{x}+\delta), \quad (5)$$

where f_i^L and f_i^U are two linear functions and $i = 1, 2, \cdots, d$. In Appendix D, we show that Equation 5 is tight when f consists of one linear layer. As $\min_{\|\delta\|_2 \le \rho_k} f_i^L(\mathbf{x}+\delta) \le \min_{\|\delta\|_2 < \rho_k} f_i^L(\mathbf{x}+\delta)$ and $\max_{\|\delta\|_2 < \rho_k} f_i^U(\mathbf{x}+\delta) \le \max_{\|\delta\|_2 \le \rho_k} f_i^U(\mathbf{x}+\delta)$, we have the following when $\|\delta\|_2 < \rho_k$:

$$\min_{\|\delta\|_2 \le \rho_k} f_i^L(\mathbf{x}+\delta) \le f_i(\mathbf{x}+\delta) \le \max_{\|\delta\|_2 \le \rho_k} f_i^U(\mathbf{x}+\delta), \quad (6)$$

Therefore, we have the following inequalities for $\forall \|\delta\|_2 < \rho_k$:

$$f_i(\mathbf{x}+\delta) - f_i(\mathbf{x}) \ge L_i, \quad (7)$$
$$f_i(\mathbf{x}+\delta) - f_i(\mathbf{x}) \le U_i, \quad (8)$$

where $L_i = \min_{\|\delta\|_2 \le \rho_k} f_i^L(\mathbf{x}+\delta) - f_i(\mathbf{x})$ and $U_i = \max_{\|\delta\|_2 \le \rho_k} f_i^U(\mathbf{x}+\delta) - f_i(\mathbf{x})$. Based on the above two inequalities, we have the following:

$$\max_{\|\delta\|_2 < \rho_k} (f_i(\mathbf{x}+\delta) - f_i(\mathbf{x}))^2 \le \max(L_i^2, U_i^2). \quad (9)$$

Therefore, we can derive an upper bound for $\max_{\|\delta\|_2 < \rho_k} \|f(\mathbf{x}+\delta) - f(\mathbf{x})\|_2$ as follows:

$$\begin{aligned} &\max_{\|\delta\|_2 < \rho_k} \|f(\mathbf{x}+\delta) - f(\mathbf{x})\|_2 && (10)\\ \le &\sqrt{\sum_{i=1}^d \max_{\|\delta\|_2 < \rho_k} (f_i(\mathbf{x}+\delta) - f_i(\mathbf{x}))^2} && (11)\\ \le &\sqrt{\sum_{i=1}^d \max(L_i^2, U_i^2)} && (12)\\ \triangleq &R'_F. && (13)\end{aligned}$$

If the upper bound R'_F is smaller than R_F, then we have $r = \rho_k$ satisfies the constraint in Equation (4). We note that $r = \rho_k$ may also satisfy the constraint even if the upper bound R'_F is no smaller than R_F. However, such cases do not influence the correctness of our binary search. Note that $\min_{\|\delta\|_2 \le \rho_k} f_i^L(\mathbf{x}+\delta)$ and $\max_{\|\delta\|_2 \le \rho_k} f_i^U(\mathbf{x}+\delta)$ have closed-form solutions for $i = 1, 2, \cdots, d$ [9]. Therefore, L_i, U_i, and the upper bound R'_F can be computed efficiently. In other words, we can efficiently check whether $r = \rho_k$ satisfies the constraint in Equation (4) for any ρ_k. Algorithm 1 shows our F2IPerturb-API, where the function CROWN obtains the lower bound and upper bound linear functions for each $f_i(\mathbf{x})$.

Our binary search based solution correctly finds a lower bound of the optimal R of the optimization problem in Equation (3) because the constraint in Equation (4) is guaranteed to be satisfied in each round of our binary search.

Image rescaling: We note that, in our above discussion on the two APIs, a client's input image size is the same as the input size of the cloud server's encoder. When the size of a client's input image is different, the cloud server can rescale it to be the input size of its encoder using the standard bilinear interpolation. The bilinear interpolation can be viewed as a linear transformation. In particular, suppose $\mathbf{x}_b$ and $\mathbf{x}_a$ respectively represent the image before and after rescaling. Then, we have $\mathbf{x}_a = \mathbf{W} \cdot \mathbf{x}_b$, where $\mathbf{W}$ is the matrix used to represent the linear transformation. The cloud server can implement this linear transformation (i.e., rescaling) by adding a linear layer whose weight matrix is $\mathbf{W}$ before the encoder. Moreover, the cloud server can view the linear layer + the encoder as a "new" encoder to implement the two APIs.

C. Pre-training Robust Encoder

Our REaaS is applicable to any encoder. However, a more robust encoder enables a client to derive a larger certified radius for its testing input. Therefore, we further propose a method to pre-train robust encoders. An encoder f is more robust if it produces more similar feature vectors for an input and its adversarially perturbed version, i.e., if $f(\mathbf{x}+\delta)$ and $f(\mathbf{x})$ are more similar. In particular, based on our implementation of the F2IPerturb-API, if $\|f(\mathbf{x}+\delta)-f(\mathbf{x})\|_2$ is smaller for any adversarial perturbation δ, then F2IPerturb-API would return a larger input-space certified radius to a client for a given feature-space certified radius. Therefore, our key idea is to reduce $\|f(\mathbf{x}+\delta)-f(\mathbf{x})\|_2$ when pre-training an encoder f. Next, we derive an upper bound of $\|f(\mathbf{x}+\delta)-f(\mathbf{x})\|_2$, based on which we design a regularization term to regularize the pre-training of an encoder.

A neural network (e.g., an encoder) can often be decomposed into the composition of a series of linear transformations [5]. In particular, we can do so if each layer of the neural network (e.g., linear layer, convolutional layer, and batch normalization layer) can be expressed as a linear transformation. We denote an encoder as the composition of n linear transformations, i.e., $f(\cdot)=T^n \circ T^{n-1} \circ \cdots \circ T^1(\cdot)$. [5] showed that the difference between the outputs of any neural network f (f is an encoder in our case) for an input and its adversarially perturbed version can be bounded as follows:

$$\|f(\mathbf{x}+\delta)-f(\mathbf{x})\|_2 \leq \prod_{j=1}^{n} \left\|T^j\right\|_s \cdot \|\delta\|_2, \tag{14}$$

where $\mathbf{x}$ is an input, δ is an adversarial perturbation, and $\|\cdot\|_s$ represents spectral norm. The product of the spectral norms of the n linear transformations (i.e., $\prod_{j=1}^{n}\left\|T^j\right\|_s$) is independent with input $\mathbf{x}$ and adversarial perturbation δ. Therefore, our idea is to add $\prod_{j=1}^{n}\left\|T^j\right\|_s$ as a regularization term (called *spectral-norm regularization*) when pre-training an encoder. Minimizing such regularization term may enforce the encoder to produce more similar feature vectors for an input and its adversarially perturbed version, i.e., $\|f(\mathbf{x}+\delta)-f(\mathbf{x})\|_2$ may be smaller. In particular, we minimize the following loss function for each mini-batch of inputs when pre-training an encoder:

$$\frac{1}{m} \cdot \sum_{i=1}^{m} \ell(i) + \lambda \cdot \prod_{j=1}^{n} \left\|T^j\right\|_s, \tag{15}$$

where $\ell(i)$ is a loss for a training input in pre-training, m is batch size, and λ is a hyper-parameter used to balance the two terms. For instance, when using supervised learning to train a classifier, whose layers excluding the output layer are used as an encoder, the loss $\ell(i)$ is often the cross-entropy loss; when using self-supervised learning algorithm MoCo [1] to pre-train an encoder, $\ell(i)$ is defined in Equation (2). We adopt the *power method* [21] to estimate the spectral norms of the linear transformations when pre-training an encoder.

D. Certifying Robustness for a Client

In REaaS, a client can treat its own downstream classifier as a base classifier. We discuss how a client can use our two APIs to train a base downstream classifier and derive the certified radius of the base downstream classifier in BC based certification or the smoothed downstream classifier in SC based certification for a testing input.

TABLE I: Comparing the communication and computation cost per training/testing input in SEaaS and REaaS. e is the number of epochs used to train a base downstream classifier. N is the number of noisy inputs per testing input in SC. T_f (or T_g) and M_f (or M_g) are respectively the number of layers and the maximum number of neurons in a layer in an encoder (or a downstream classifier). K_f (or K_g) is the number of parameters in an encoder (or a downstream classifier).

(a) Communication cost

Service	#Queries			
	Per training input		Per testing input	
	BC	SC	BC	SC
SEaaS	N/A	e	N/A	N
REaaS	1		2	

(b) Computation cost

Service	Entity	Computational complexity			
		Per training input		Per testing input	
		BC	SC	BC	SC
SEaaS	Cloud server	N/A	$O(e \cdot K_f)$	N/A	$O(N \cdot K_f)$
	Client		$O(e \cdot K_g)$		$O(N \cdot K_g)$
REaaS	Cloud server	$O(K_f)$	$O(K_f)$	$O(K_f + T_f^2 \cdot M_f^3)$	$O(K_f + T_f^2 \cdot M_f^3)$
	Client	$O(e \cdot K_g)$	$O(e \cdot K_g)$	$O(K_g + T_g^2 \cdot M_g^3)$	$O(N \cdot K_g)$

BC based certification: When training a base downstream classifier, a client queries the Feature-API to obtain a feature vector for each training input. Then, given the feature vectors and the corresponding training labels, the client can use any training method (e.g., standard supervised learning) to train a base downstream classifier. Given a testing input, the client queries the Feature-API to obtain its feature vector and uses the base downstream classifier to predict its label. Moreover, the client can use any BC based certification method to derive a feature-space certified radius for the testing input by treating its feature vector as an "input" to the base downstream classifier. Then, the client queries the F2IPerturb-API to transform the feature-space certified radius to an input-space certified radius.

SC based certification: Similar to BC based certification, a client queries the Feature-API to obtain a feature vector for each training input when training a base downstream classifier. However, unlike BC based certification, the client adds noise to the training feature vectors in SC based certification. In particular, the client adds random noise (e.g., Gaussian noise) to each feature vector in each mini-batch of training feature vectors in each training epoch. Note that the client does not need to query the Feature-API again for the noisy feature vector. Given a testing input, the client queries the Feature-API to obtain its feature vector and uses the smoothed downstream classifier to predict its label and derive its feature-space certified radius. In particular, the client constructs N noisy feature vectors by adding random noise to the feature vector and uses it's base downstream classifier to predict their labels. Based on the predicted labels, the client can derive the predicted label and feature-space certified radius for the original feature vector. Then, the client queries the F2IPerturb-API to transform the feature-space certified radius to an input-space certified radius.

E. Theoretical Communication and Computation Cost Analysis

Communication cost: The number of queries to the APIs characterizes the communication cost for a client and the cloud server. In both BC and SC based certification, a client only queries the Feature-API once for each training input in REaaS. Therefore, the number of queries per training input is 1 in REaaS. In both BC and SC based certification, a client only queries the Feature-API and F2IPerturb-API once to derive the predicted label and certified radius for a testing input. Therefore, the number of queries per testing input is 2 in REaaS. Note that the client only sends an image $\mathbf{x}$ to the cloud server when querying the Feature-API, while it also sends the feature-space certified radius R_f to the cloud server when querying the F2IPerturb-API. However, R_f is a real number whose communication cost is negligible, compared to that of the image $\mathbf{x}$. Thus, we consider querying Feature-API and querying F2IPerturb-API have the same communication cost in our analysis for simplicity. Table Ia compares the number of queries per training/testing input in SEaaS and REaaS. Compared with SEaaS, REaaS makes BC based certification applicable and incurs a much smaller communication cost in SC based certification.

Computation cost: Table Ib compares the computational complexity of REaaS and SEaaS for the cloud server and a client. In both REaaS and SEaaS, the computation cost for the cloud server to process a query to the Feature-API is linear to the number of encoder parameters, i.e., $O(K_f)$, where K_f is the number of parameters in the encoder. In REaaS, we use binary search to process a query to the F2IPerturb-API. Given the initial search range $[\rho_1^L, \rho_1^U]$ and binary-search precision β, the number of rounds of binary search is $\lceil \log_2(\frac{\rho_1^U - \rho_1^L}{\beta}) \rceil$. In practice, we can set ρ_1^L, ρ_1^U, and β to be constants, e.g., $\rho_1^L = 0$, $\rho_1^U = 10$, and $\beta = 10^{-50}$, and thus $\lceil \log_2(\frac{\rho_1^U - \rho_1^L}{\beta}) \rceil$ can be viewed as a constant. From [9], the computational complexity is $O(T_f^2 \cdot M_f^3)$ in each round of binary search, where T_f and M_f are respectively the number of layers and the maximum number of neurons in a layer in an encoder. Thus, the computational complexity for the cloud server to process a query to the F2IPerturb-API is $O(T_f^2 \cdot M_f^3)$.

On the client side, the computational complexity of gradient descent is $O(K_g)$ for each training input per epoch when training a base downstream classifier in both BC and SC based certification, where K_g is the number of parameters in the base downstream classifier. Therefore, the computational complexity of training a base downstream classifier is $O(e \cdot K_g)$ per training input, where e is the number of training epochs. The computational complexity for a client to derive the feature-space certified radius of a testing input is $O(T_g^2 \cdot M_g^3)$ in BC based certification [9], where T_g and M_g are respectively the number of layers and the maximum number of neurons in a layer in the base downstream classifier. Moreover, the computational complexity of using the base downstream classifier to predict a label for a (noisy) feature vector is $O(K_g)$.

As SEaaS does not support BC based certification, we focus on comparing the computation cost of SEaaS and REaaS for SC based certification. First, we observe that the computation cost per training/testing input is the same for a client in SEaaS and REaaS. Second, REaaS incurs a smaller computation cost per training input for the cloud server than SEaaS, because REaaS incurs much fewer queries than SEaaS. Third, REaaS often incurs a smaller computation cost per testing input for the cloud server than SEaaS, because N is often large to achieve a large certified radius as shown in our experiments.

V. EVALUATION

A. Experimental Setup

Datasets: We use CIFAR10 [22], SVHN [23], STL10 [24], and Tiny-ImageNet [25] in our experiments. CIFAR10 has 50,000 training and 10,000 testing images from ten classes. SVHN contains 73,257 training and 26,032 testing images from ten classes. STL10 contains 5,000 training and 8,000 testing images from ten classes, as well as 100,000 unlabeled images. Tiny-ImageNet contains 100,000 training and 10,000 testing images from 200 classes.

We rescale each image in all datasets to 32×32 by the standard bi-linear interpolation. Therefore, the input image size in a downstream dataset is the same as the input size of the pre-trained encoder. However, we will also explicitly explore the scenarios in which the input image size of a downstream dataset is different from the input size of the pre-trained encoder.

Pre-training encoders: We use STL-10 and Tiny-ImageNet as pre-training datasets to pre-train encoders. We adopt these two datasets because they contain more images than CIFAR10 and SVHN. In particular, we use the unlabeled data of STL10 to pre-train an encoder when STL10 is used as a pre-training dataset. When Tiny-ImageNet is used as a pre-training dataset, we use its training dataset to pre-train an encoder. Unless otherwise mentioned, we adopt MoCo [1] as the pre-training algorithm in SEaaS, while we adopt MoCo with our spectral-norm regularization as the pre-training algorithm in REaaS, since they only need unlabeled data. Moreover, we adopt the public implementation of MoCo [26] in our experiments. When calculating the spectral norm of the encoder during pre-training, we run 10 iterations of power iteration in each mini-batch. The architecture of the encoder can be found in Table XIII in Appendix. We pre-train an encoder for 500 epochs with a learning rate 0.06 and a batch size 512.

Training downstream classifiers: As we have four datasets, we use the other three datasets as downstream datasets when a dataset is used as a pre-training dataset. Moreover, when a dataset is used as a downstream dataset, we adopt its training dataset as the downstream training dataset and testing dataset as the downstream testing dataset. We use the downstream training dataset to train a base downstream classifier. In particular, in BC based certification, we use standard supervised learning to train a base downstream classifier on the feature vectors of the training inputs. We note that some works [27], [28] proposed new methods to train a base classifier to improve its certified robustness in BC based certification. These methods are also applicable in our REaaS, but we do not evaluate them since our focus is to show the applicability of BC based certification in REaaS instead of its optimal certified robustness. For SC based certification, we train a base downstream classifier via adding Gaussian noise $\mathcal{N}(0, \sigma^2\mathbf{I})$ to the training inputs in SEaaS and the feature vectors of the training inputs in REaaS.

We use a fully connected neural network with two hidden layers as a base downstream classifier. We respectively adopt

ReLU and Softmax as the activation functions in the two hidden layers and the output layer. The number of neurons in both hidden layers is 256. We train a base downstream classifier for 25 epochs using cross-entropy loss, a learning rate of 0.06, and a batch size of 512.

Certification methods: For BC based certification, we adopt CROWN [9] to derive the certified radius of a base downstream classifier for a testing input in REaaS. We adopt the public implementation of CROWN [29]. For SC based certification, we adopt Gaussian noise based randomized smoothing [13] to build a smoothed classifier and derive its certified radius for a testing input. In SEaaS, a client treats the composition of the encoder and its downstream classifier as a base classifier, while a client treats its downstream classifier alone as a base classifier in REaaS. We use the public code [30] for Gaussian noise based randomized smoothing. Appendix B and C show the technical details of CROWN and Gaussian noise based randomized smoothing, respectively.

Evaluation metrics: Recall that REaaS aims to achieve three design goals. We can evaluate the generality goal by showing that REaaS supports both BC and SC based certification. For the efficiency goal, we use *#Queries per training (or testing) input* to measure the communication cost between a client and the cloud server. Moreover, we use *running time per testing input* on the cloud server to measure its computation cost. We do not consider running time on a client as it is the same in SEaaS and REaaS. Note that #Queries per training input also characterizes the computation cost per training input for the cloud server as it is linear to the number of queries. For the robustness goal, we use *average certified radius (ACR)* of the correctly classified testing examples to measure the certified robustness of a base or smoothed classifier.

Note that there often exists a trade-off between robustness and accuracy for a classifier. Therefore, we further consider accuracy under adversarial perturbation as an evaluation metric. In particular, we consider the widely adopted *certified accuracy @ a perturbation size*, which is the fraction of testing inputs in a downstream testing dataset whose labels are correctly predicted and whose certified radii are no smaller than the given perturbation size. Certified accuracy @ a perturbation size is the least testing accuracy that a classifier can achieve no matter what adversarial perturbation is added to each testing input once its ℓ_2-norm is at most the given perturbation size. The certified accuracy @ a perturbation size decreases as the perturbation size increases. ACR is the area under the certified accuracy vs. perturbation size curve (details are shown in Appendix A). Therefore, ACR can also be viewed as a metric to measure the robustness-accuracy trade-off of a classifier, where a larger ACR indicates a better trade-off.

Parameter settings: F2IPerturb-API has the following three parameters: ρ_1^L and ρ_1^U which specify the range of R in the first round of binary search, and β which characterizes the binary-search precision. We set ρ_1^L to be 0 and set ρ_1^U to be 10. Note that they do not impact experimental results once ρ_1^L is set to 0 and ρ_1^U is set to a large value (e.g., 10). We set the default value of β as 0.001. We note that β has a negligible impact on certified accuracy and ACR. In particular, the absolute difference between the certified accuracy (or ACR) when $\beta = 0.001$ and that when β is an arbitrarily small value (e.g., 10^{-50}) is smaller than 0.001.

TABLE II: ACR and #Queries in SEaaS and REaaS.

(a) Pre-training dataset is Tiny-ImageNet

Service	Certification method	Downstream dataset	ACR	#Queries	
				Per training input	Per testing input
SEaaS	BC	CIFAR10	N/A		
		SVHN			
		STL10			
	SC	CIFAR10	0.157	25	1×10^5
		SVHN	0.226		
		STL10	0.134		
REaaS	BC	CIFAR10	0.138	1	2
		SVHN	0.258		
		STL10	0.090		
	SC	CIFAR10	0.171		
		SVHN	0.275		
		STL10	0.143		

(b) Pre-training dataset is STL10

Service	Certification method	Downstream dataset	ACR	#Queries	
				Per training input	Per testing input
SEaaS	BC	CIFAR10	NA		
		SVHN			
		Tiny-ImageNet			
	SC	CIFAR10	0.155	25	1×10^5
		SVHN	0.244		
		Tiny-ImageNet	0.016		
REaaS	BC	CIFAR10	0.139	1	2
		SVHN	0.272		
		Tiny-ImageNet	0.027		
	SC	CIFAR10	0.173		
		SVHN	0.278		
		Tiny-ImageNet	0.033		

Randomized smoothing has the following three parameters: the number of Gaussian noise N, standard deviation σ of the Gaussian noise, and error probability α. Following prior work [13], unless otherwise mentioned, we set $N = 100,000$, $\sigma = 0.5$, and $\alpha = 0.001$. We set the default value of the hyperparameter λ in our pre-training method as 0.00075. We normalize pixel values to $[0, 1]$.

B. Experimental Results

We first show that REaaS achieves our three design goals, but SEaaS does not. Then, we show the impact of relevant factors on REaaS. In particular, we consider 1) different ways to pre-train an encoder, 2) image scaling, and 3) different hyperparameters of certification methods such as N, σ, and α for randomized smoothing. Note that we fix all other parameters to their default values when studying the impact of one parameter on REaaS.

TABLE III: Comparing the running time per testing input for the cloud server in SC for SEaaS and REaaS. The pre-training dataset is Tiny-ImageNet.

Service	Downstream dataset	Running time (s) per testing input
SEaaS	CIFAR10	73.77
	SVHN	72.65
	STL10	73.48
REaaS	CIFAR10	1.05
	SVHN	1.06
	STL10	1.04

TABLE IV: Training without noise vs. training with noise for SC in SEaaS. The pre-training dataset is Tiny-ImageNet.

Downstream dataset	ACR	
	Training with noise	Training without noise
CIFAR10	0.157	0.106
SVHN	0.226	0.155
STL10	0.134	0.088

TABLE V: Impact of N on ACR for SC in SEaaS. The pre-training dataset is Tiny-ImageNet.

Downstream dataset	N			
	100	1,000	10,000	100,000
CIFAR10	0.091	0.132	0.148	0.157
SVHN	0.130	0.186	0.211	0.226
STL10	0.079	0.111	0.127	0.134

REaaS achieves the generality, efficiency, and robustness goals: In SC based certification, a client respectively adds Gaussian noise to images and their feature vectors to train a base downstream classifier in SEaaS and REaaS. Thus, the certified robustness of the smoothed classifiers are not comparable even if we use the same standard deviation σ of Gaussian noise in SEaaS and REaaS. Therefore, we try multiple values of σ and report the largest ACR for each service. Moreover, we select σ values such that the largest ACR is not reached at the smallest or largest value of σ, to ensure the largest ACR is found for each service. In particular, we try $\sigma = 0.125, 0.25, 0.5, 0.75, 1$ for both SEaaS and REaaS. We note that σ controls a tradeoff between certified accuracy without attacks (i.e., perturbation size is 0) and robustness. Specifically, a smaller σ can achieve a larger certified accuracy without attacks but also make the curve drop more quickly (i.e., less robust). ACR measures such trade-off, and thus we adopt the σ that achieves the largest ACR for each method when comparing the certified accuracy of SC based certification in SEaaS and REaaS.

Table II compares ACR and #Queries per training/testing input in SEaaS and REaaS, while Table III compares the running time per testing input for the server in SC for SEaaS and REaaS. We have the following observations. First, REaaS supports both BC and SC. Therefore, REaaS achieves the generality goal. In contrast, SEaaS only supports SC. Second, REaaS achieves the efficiency goal as it is much more efficient than SEaaS. Specifically, #Queries per training/testing input in REaaS is orders of magnitude smaller than that in SEaaS for SC. We note that a client using SEaaS could choose to train a base downstream classifier without adding noise to its training inputs to reduce the #Queries per training input to 1 or use a small N to reduce the #Queries per testing input. However, the smoothed classifier achieves (much) smaller ACRs in such cases as shown in Table IV and V. Base on Table III, REaaS also incurs a much lower computation cost for the server than SEaaS.

Third, REaaS achieves the robustness goal as it achieves larger ACRs than SEaaS for SC. The reason is that, in SEaaS, the base classifier is the composition of an encoder and a base downstream classifier, but the client can only train the base downstream classifier with noise. In contrast, a client can build a smoothed classifier upon a base downstream classifier alone which can be trained with noise and the encoder is pre-trained in a robust way in REaaS. Figure 7 in Appendix further compares the certified accuracy vs. perturbation size of SC in SEaaS and REaaS. We find that REaaS can achieve a better trade-off between accuracy without attacks and robustness than SEaaS. Specifically, REaaS achieves larger certified accuracy than SEaaS when the perturbation size is small. Moreover, the gap between the certified accuracy of SEaaS and REaaS is much larger when the perturbation size is small than that when the perturbation size is large.

TABLE VI: Comparing the ACRs in REaaS for different downstream datasets when the encoders are pre-trained by different self-supervised learning methods. The pre-training dataset is Tiny-ImageNet.

(a) CIFAR10

Certification Method	Pre-training Method	ACR
BC	Non-robust MoCo	0.012
	RoCL	0.016
	Ours	0.138
SC	Non-robust MoCo	0.020
	RoCL	0.024
	Ours	0.171

(b) SVHN

Certification Method	Pre-training Method	ACR
BC	Non-robust MoCo	0.009
	RoCL	0.015
	Ours	0.258
SC	Non-robust MoCo	0.011
	RoCL	0.014
	Ours	0.275

(c) STL10

Certification Method	Pre-training Method	ACR
BC	Non-robust MoCo	0.011
	RoCL	0.014
	Ours	0.090
SC	Non-robust MoCo	0.015
	RoCL	0.020
	Ours	0.143

Impact of methods to pre-train encoders: We can use different methods to pre-train an encoder in REaaS. Table VI and VII show ACRs in REaaS when different self-supervised learning methods are used to pre-train encoders. In particular,

TABLE VII: Comparing the ACRs in REaaS for different downstream datasets when the encoders are pre-trained by different self-supervised learning methods. The pre-training dataset is STL10.

(a) CIFAR10

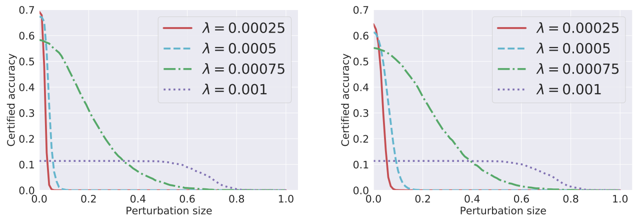
Certification Method	Pre-training Method	ACR
BC	Non-robust MoCo	0.010
	RoCL	0.012
	Ours	0.139
SC	Non-robust MoCo	0.014
	RoCL	0.017
	Ours	0.173

(b) SVHN

Certification Method	Pre-training Method	ACR
BC	Non-robust MoCo	0.006
	RoCL	0.009
	Ours	0.272
SC	Non-robust MoCo	0.007
	RoCL	0.012
	Ours	0.278

(c) Tiny-ImageNet

Certification Method	Pre-training Method	ACR
BC	Non-robust MoCo	0.003
	RoCL	0.004
	Ours	0.027
SC	Non-robust MoCo	0.003
	RoCL	0.004
	Ours	0.033



(a) BC (b) SC

Fig. 2: Impact of λ on certified accuracy vs. perturbation size for BC and SC in REaaS. The pre-training dataset is Tiny-ImageNet and the downstream dataset is CIFAR10.

we consider non-robust MoCo [1], RoCL [14], and our robust pre-training method (i.e., MoCo with our spectral-norm regularization). Table VIII shows ACRs of REaaS when different supervised learning methods are used to pre-train encoders. In particular, we consider a standard, non-robust supervised learning method, adversarial training [16] (we use the default parameter settings in the authors' public implementation), and our robust pre-training method (i.e., standard supervised learning with our spectral-norm regularization). We only show results when the pre-training dataset is Tiny-ImageNet for supervised pre-training methods, as STL10 dataset only has a small number of labeled training images which are insufficient to pre-train high-quality encoders using supervised learning. We try $\sigma = 0.125, 0.25, 0.5, 0.75, 1$ and report the largest ACR for each pre-training method. As the results show, our robust pre-training method achieves substantially larger ACRs

TABLE VIII: Comparing the ACRs in REaaS for different downstream datasets when the encoders are pre-trained by different supervised learning (SL) methods. The pre-training dataset is Tiny-ImageNet.

(a) CIFAR10

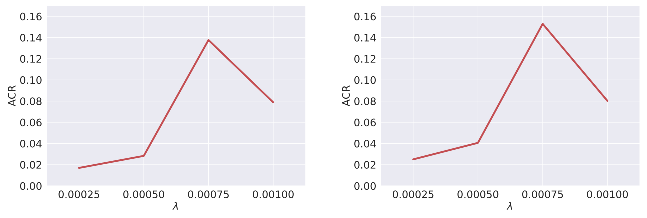
Certification Method	Pre-training Method	ACR
BC	Non-robust SL	0.019
	Adversarial Training	0.035
	Ours	0.174
SC	Non-robust SL	0.022
	Adversarial Training	0.041
	Ours	0.172

(b) SVHN

Certification Method	Pre-training Method	ACR
BC	Non-robust SL	0.008
	Adversarial Training	0.018
	Ours	0.268
SC	Non-robust SL	0.011
	Adversarial Training	0.021
	Ours	0.292

(c) STL10

Certification Method	Pre-training Method	ACR
BC	Non-robust SL	0.012
	Adversarial Training	0.026
	Ours	0.100
SC	Non-robust SL	0.018
	Adversarial Training	0.034
	Ours	0.114



(a) BC (b) SC

Fig. 3: Impact of λ on ACR for BC and SC in REaaS. The pre-training dataset is Tiny-ImageNet and the downstream dataset is CIFAR10.

than existing methods for both supervised learning and self-supervised learning. Our method is better than RoCL and adversarial training because they aim to train empirically robust rather than certifiably robust encoders, and is better than MoCo and standard supervised learning because the encoders pre-trained by them are non-robust.

Impact of hyperparameter λ: Figure 2 shows the impact of λ on certified accuracy in REaaS. We find that λ measures a trade-off between accuracy without attacks (i.e., perturbation size is 0) and robustness. In particular, when λ is smaller, the accuracy without attacks is larger, but the certified accuracy decreases more quickly as the perturbation size increases. Figure 3 shows

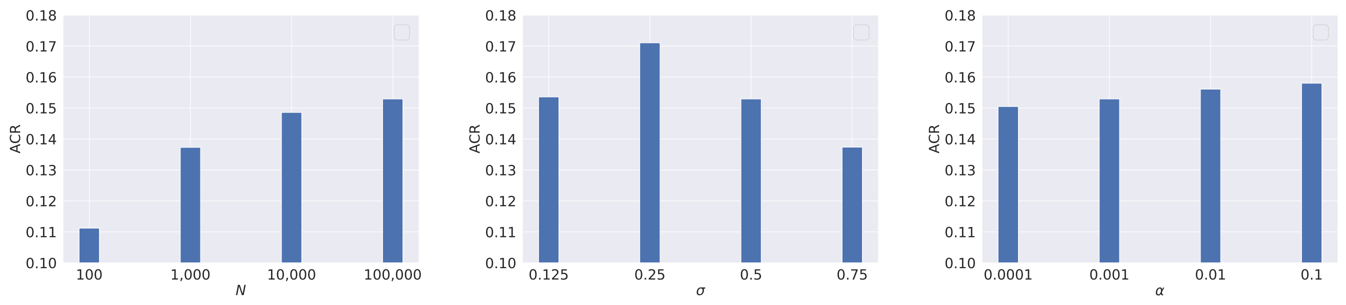
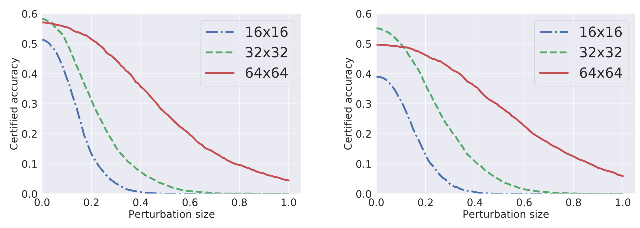


Fig. 4: Impact of N, σ, and α on ACR of SC in REaaS. The pre-training dataset is Tiny-ImageNet and the downstream dataset is CIFAR10.



(a) BC (b) SC

Fig. 5: Impact of downstream input size on certified accuracy vs. perturbation size for BC and SC in REaaS. The pre-training dataset is Tiny-ImageNet and the downstream dataset is (or created from) CIFAR10. The input size of the pre-trained encoder is 32x32.

the impact of λ on ACR. Our results show that, for both BC and SC, ACR first increases as λ increases and then decreases after λ is larger than a certain value. The reason is that a larger or smaller λ leads to a worse trade-off between accuracy without attacks and robustness as shown in Figure 2.

Impact of image rescaling: To study the impact of image rescaling, we create downstream datasets with different input image sizes via resizing images in CIFAR10. Table IX shows the results on ACR and Figure 5 shows the results on certified accuracy. We find that, when the size of the images in a downstream dataset is larger (or smaller) than the input size of the encoder, the downstream input-space ACR is larger (or smaller) for both BC and SC. The reason is that down-scaling (or up-scaling) the downstream input images to be the same size as the input size of the encoder reduces (or enlarges) the perturbation in the downstream image space.

Impact of N, σ, and α for SC: Figure 4 and 8 (in Appendix) shows the impact of N, σ, and α on ACR and certified accuracy of SC in REaaS. We have the following observations. First, both ACR and certified accuracy increase as N or α increases. The reason is that the estimated certified radii are larger when N or α is larger. Second, we find that σ achieves a trade-off between accuracy without attacks (i.e., perturbation size is 0) and robustness. In particular, a smaller σ can achieve a larger accuracy without attacks, but the curve drops faster as the perturbation size increases. Third, ACR first increases and then decreases as σ increases. The reason is that a smoothed classifier is less accurate without attacks when σ is larger and is less robust when σ is smaller.

TABLE IX: Impact of image rescaling on ACR in REaaS. The pre-training dataset is Tiny-ImageNet and the downstream dataset is (or created from) CIFAR10. The input size of the encoder is 32x32.

Certification Method	Size of Images in Downstream Dataset	ACR
BC	16x16	0.082
	32x32	0.138
	64x64	0.303
SC	16x16	0.068
	32x32	0.153
	64x64	0.305

REaaS vs. white-box access to the encoder: In REaaS, a client has black-box access to the encoder. We compare REaaS with the scenario where a client has white-box access to the encoder, e.g., the cloud server shares its encoder with a client. Specifically, with white-box access to the encoder, a client can use either BC or SC by treating the composition of the encoder and its downstream classifier as a base classifier. For BC, the client can use CROWN [29] to derive the certified radius of its base classifier for a testing input. For SC, the client can train/fine-tune the base classifier (both the encoder and downstream classifier) using training inputs with noise. The white-box scenario represents the upper-bound robustness a client can achieve. Therefore, comparing with the robustness in the white-box scenario enables us to understand how close our REaaS with the two APIs is to such upper bound. Table X compares the ACRs of REaaS and such white-box scenario. We find that REaaS can achieve comparable ACRs with the white-box scenario.

VI. Discussion

Extension to ℓ_p-norm adversarial perturbations: We focus on certified robustness against ℓ_2-norm adversarial perturbation in this work. The certified robustness can be extended to other ℓ_p-norms, e.g., via leveraging the relationship between ℓ_2-norm and other ℓ_p-norms. For instance, suppose the certified radius is R for an image in ℓ_2-norm; the certified radius in ℓ_1-norm and ℓ_∞-norm can respectively be computed as R and $\frac{R}{\sqrt{dim}}$,

TABLE X: Comparing the ACRs of REaaS and the white-box scenario for different downstream datasets. The pre-training dataset is Tiny-ImageNet.

(a) CIFAR10

Certification Method	Service	ACR
BC	White-box	0.157
	REaaS	0.138
SC	White-box	0.188
	REaaS	0.171

(b) SVHN

Certification Method	Service	ACR
BC	White-box	0.286
	REaaS	0.258
SC	White-box	0.302
	REaaS	0.275

(c) STL10

Certification Method	Service	ACR
BC	White-box	0.102
	REaaS	0.090
SC	White-box	0.151
	REaaS	0.143

where dim is the product of the number of pixels and the number of channels in the image. Figure 6 shows the certified accuracy of SC in REaaS for ℓ_1-norm and ℓ_∞-norm adversarial perturbations, where the ℓ_1-norm and ℓ_∞-norm certified radii are obtained from ℓ_2-norm certified radius with $N = 100,000$, $\sigma = 0.5$, and $\alpha = 0.001$.

Extending REaaS to natural language processing (NLP) domain: An attacker can make a text classifier predict an incorrect label for a text by substituting a small number of words as their synonyms [31], [32], [33]. Our REaaS can also be applied to enable adversarially robust downstream text classifiers against those attacks by slightly adapting our F2IPerturb-API (please refer to Appendix E for details). Given a text and a feature-space certified radius, our adapted F2IPerturb-API returns an input-space certified radius, which is the maximum number of words that can be substituted such that the downstream classifier's predicted label for the text is unchanged. Table XI shows our experimental results (please refer to Appendix E for details of the experimental setup). Our results show that our REaaS is also applicable to NLP domain.

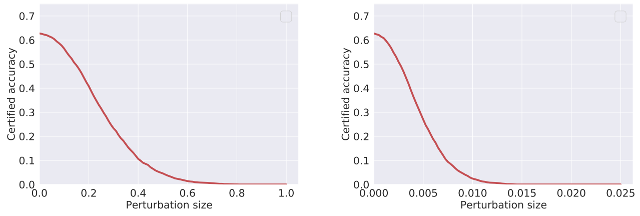
Encoder stealing: Our REaaS introduces a new F2IPerturb-API. A natural question is whether the new F2IPerturb-API makes the encoder more vulnerable to stealing attacks. We argue that the answer is probably no. The reason is that our new API returns a certified radius for a query image, which can also be obtained by an attacker via calling the existing Feature-API many times. However, an attacker may obtain such certified radius with less queries using our new API. We explore whether certified radii can be exploited to assist encoder stealing. In particular, we extend StolenEncoder [34], which uses Feature-API to steal encoder, to steal encoders using both Feature-API and F2IPerturb-API (see Appendix F for the details of StolenEncoder and its extended version as well as the experimental setup). Table XII shows our experimental results, where γ is a hyperparameter. Note that the total number of queries to the APIs made by the extended StolenEncoder is twice of StolenEncoder in our comparison. Our results show that the downstream classifiers built upon stolen encoders obtained by StolenEncoder and its extended version achieve comparable accuracy, which implies that certified radii may not be able to assist encoder stealing.

TABLE XI: ACR and #Queries of REaaS in NLP domain, where BC is used. The pre-training dataset is SST-2 [44] and the downstream dataset is IMDB [45].

ACR	#Queries	
	Per training input	Per testing input
2.517	1	2

TABLE XII: Comparing StolenEncoder and its extended version using our F2IPerturb-API. The pre-training dataset is Tiny-ImageNet and the downstream dataset is CIFAR10.

	StolenEncoder	Extended StolenEncoder			
γ	0	10^{-4}	10^{-3}	10^{-2}	10^{-1}
Stolen Accuracy (%)	62.3	62.6	63.4	61.5	59.4



(a) ℓ_1-norm (b) ℓ_∞-norm

Fig. 6: Certified accuracy vs. perturbation size of SC in REaaS under ℓ_1-norm and ℓ_∞-norm adversarial perturbations. The pre-training dataset is Tiny-ImageNet and the downstream dataset is CIFAR10.

Privacy-preserving encoder as a service: In both SEaaS and REaaS, a client sends his/her raw images to the cloud server. Therefore, an untrusted service provider may compromise the privacy of the client, especially when the downstream datasets contain sensitive images such as facial and medical images. We believe it is an interesting future work to develop privacy-preserving encoder as a service. For instance, we can leverage (local) differential privacy [35], [36], [37], secure hardware [38], and cryptography [39], [40] based methods.

Other attacks to pre-trained encoders: In this work, we focus on adversarial examples [5], [6]. Some recent studies [41], [42], [43] show that pre-trained encoders are also vulnerable to poisoning and backdoor attacks, which are orthogonal to our work. We believe it is an interesting future work to extend our framework to defend against those attacks.

VII. Conclusion and Future Work

In this work, we show that, via providing two APIs, a cloud server 1) makes it possible for a client to certify robustness of its downstream classifier against adversarial perturbations using any certification method and 2) makes it orders of magnitude more communication efficient and more computation efficient to certify robustness using smoothed classifier based certification. Moreover, when the cloud server pre-trains the encoder via considering our spectral-norm regularization term, it achieves better certified robustness for the clients' downstream classifiers. Interesting future work includes extending REaaS to poisoning and backdoor attacks as well as designing both robust and privacy-preserving encoder as a service.

Acknowledgements

We thank the anonymous reviewers for the constructive comments. This work was supported by NSF under grant No. 2131859, 2112562, and 1937786, as well as ARO under grant No. W911NF2110182.

References

[1] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *CVPR*, 2020.

[2] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *ICML*, 2020.

[3] GPT-3 API, https://openai.com/blog/customized-gpt-3/.

[4] "General Image Embedding Model," https://www.clarifai.com/models/general-image-embedding.

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

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

[7] E. Wong and J. Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *ICML*, 2018.

[8] A. Raghunathan, J. Steinhardt, and P. Liang, "Certified defenses against adversarial examples," in *ICLR*, 2018.

[9] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," in *NeurIPS*, 2018.

[10] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *USENIX Security*, 2018.

[11] X. Cao and N. Z. Gong, "Mitigating evasion attacks to deep neural networks via region-based classification," in *ACSAC*, 2017.

[12] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *IEEE S & P*, 2019.

[13] J. M. Cohen, E. Rosenfeld, and J. Z. Kolter, "Certified adversarial robustness via randomized smoothing," *ICML*, 2019.

[14] M. Kim, J. Tack, and S. J. Hwang, "Adversarial self-supervised contrastive learning," in *NeurIPS*, 2020.

[15] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *ICLR*, 2015.

[16] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *ICLR*, 2018.

[17] H. Salman, M. Sun, G. Yang, A. Kapoor, and J. Z. Kolter, "Denoised smoothing: A provable defense for pretrained classifiers," *NeurIPS*, 2020.

[18] S. J. Pan and Q. Yang, "A survey on transfer learning," *TKDE*, 2009.

[19] T. Chen, S. Liu, S. Chang, Y. Cheng, L. Amini, and Z. Wang, "Adversarial robustness: From self-supervised pre-training to fine-tuning," in *CVPR*, 2020.

[20] Z. Jiang, T. Chen, T. Chen, and Z. Wang, "Robust pre-training by adversarial contrastive learning." in *NeurIPS*, 2020.

[21] R. Mises and H. Pollaczek-Geiringer, "Praktische verfahren der gleichungsauflösung." *ZAMM-Journal of Applied Mathematics and Mechanics*, 1929.

[22] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," *Tech Report*, 2009.

[23] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NeurIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[24] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *AISTATS*, 2011.

[25] "Tiny-ImageNet," https://www.kaggle.com/c/tiny-imagenet/overview.

[26] "MoCo," https://github.com/facebookresearch/moco.

[27] M. Balunovic and M. Vechev, "Adversarial training and provable defenses: Bridging the gap," in *ICLR*, 2019.

[28] H. Zhang, H. Chen, C. Xiao, S. Gowal, R. Stanforth, B. Li, D. Boning, and C.-J. Hsieh, "Towards stable and efficient training of verifiably robust neural networks," *ICLR*, 2020.

[29] "CROWN," https://github.com/huanzhang12/CROWN-IBP.

[30] "Randomized Smoothing," https://github.com/locuslab/smoothing.

[31] M. Alzantot, Y. Sharma, A. Elgohary, B.-J. Ho, M. Srivastava, and K.-W. Chang, "Generating natural language adversarial examples," in *EMNLP*, 2018.

[32] S. Ren, Y. Deng, K. He, and W. Che, "Generating natural language adversarial examples through probability weighted word saliency," in *ACL*, 2019.

[33] P.-S. Huang, R. Stanforth, J. Welbl, C. Dyer, D. Yogatama, S. Gowal, K. Dvijotham, and P. Kohli, "Achieving verified robustness to symbol substitutions via interval bound propagation," *EMNLP-IJCNLP*, 2019.

[34] Y. Liu, J. Jia, H. Liu, and N. Z. Gong, "Stolenencoder: Stealing pre-trained encoders in self-supervised learning," in *CCS*, 2022.

[35] J. C. Duchi, M. I. Jordan, and M. J. Wainwright, "Local privacy and statistical minimax rates," in *FOCS*, 2013.

[36] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *CCS*, 2014.

[37] T. Wang, J. Blocki, N. Li, and S. Jha, "Locally differentially private protocols for frequency estimation," in *USENIX Security*, 2017.

[38] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptol. ePrint Arch.*, 2016.

[39] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data." in *NDSS*, 2015.

[40] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *ICML*, 2016.

[41] J. Jia, Y. Liu, and N. Z. Gong, "Badencoder: Backdoor attacks to pre-trained encoders in self-supervised learning," in *IEEE S&P*, 2022.

[42] N. Carlini and A. Terzis, "Poisoning and backdooring contrastive learning," in *ICLR*, 2021.

[43] H. Liu, J. Jia, and N. Z. Gong, "Poisonedencoder: Poisoning the unlabeled pre-training data in contrastive learning," in *USENIX Security*, 2022.

[44] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *EMNLP*, 2013.

[45] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *ACL*, 2011.

[46] C. J. Clopper and E. S. Pearson, "The use of confidence or fiducial limits illustrated in the case of the binomial," *Biometrika*, 1934.

[47] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, and C.-J. Hsieh, "Automatic perturbation analysis for scalable certified robustness and beyond," in *NeurIPS*, 2020.

[48] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *EMMLP*, 2013.

[49] R. Jia, A. Raghunathan, K. Göksel, and P. Liang, "Certified robustness to adversarial word substitutions," *arXiv preprint arXiv:1909.00986*, 2019.

TABLE XIII: Architecture of the neural network for the encoder.

Layer Type	Layer Parameters
Input 32×32	
Convolution	$16 \times 3 \times 3$, strides=$(1, 1)$, padding=1
Activation	ReLU
Convolution	$16 \times 4 \times 4$, strides=$(2, 2)$, padding=1
Activation	ReLU
Convolution	$32 \times 3 \times 3$, strides=$(1, 1)$, padding=1
Activation	ReLU
Convolution	$32 \times 4 \times 4$, strides=$(2, 2)$, padding=1
Activation	ReLU
Fully Connected	256
Output	

Appendix A
ACR is the Area under the Certified Accuracy vs. Perturbation Size Curve

Suppose the largest certified radius of a testing input is R_{max} and the certified radius of the testing inputs follows a probability distribution in the interval [0, R_{max}], whose probability density function is $p(R)$. The certified accuracy (CA) at a perturbation size R can be formally defined as follows:

$$CA(R) = \int_{R}^{R_{max}} p(r)dr. \tag{16}$$

By definition, the area under the certified accuracy vs. perturbation size curve is calculated as follows:

$$\begin{aligned} Area &= \int_{0}^{R_{max}} CA(R)dR & (17)\\ &= CA(R) \cdot R|_{0}^{R_{max}} - \int_{0}^{R_{max}} R \cdot dCA(R) & (18)\\ &= 0 + \int_{0}^{R_{max}} R \cdot p(R) \cdot dR & (19)\\ &= ACR, & (20)\end{aligned}$$

where ACR is the average certified radius of the testing inputs.

Appendix B
Technical Details of CROWN [9]

Suppose F is a base classifier that maps an input $\mathbf{x}$ to one of c classes $\{1, 2, \cdots, c\}$. For instance, the composition of the encoder and downstream classifier $g \circ f$ is the base classifier F in SEaaS, and a client can treat the downstream classifier g as the base classifier F in REaaS. We use $H(\mathbf{x})$ to denote the base classifier's last-layer output vector for $\mathbf{x}$, where $H_l(\mathbf{x})$ represents the lth entry of $H(\mathbf{x})$ and $l = 1, 2, \cdots, c$. $F(\mathbf{x})$ denotes the predicted label for $\mathbf{x}$, i.e., $F(\mathbf{x}) = \text{argmax}_{l=1,2,\cdots,c} H_l(\mathbf{x})$. Suppose the base classifier predicts label y for $\mathbf{x}$ when there is no adversarial perturbation, i.e., $F(\mathbf{x}) = y$. A base classifier based certification method derives a certified radius R such that $F(\mathbf{x} + \delta) = y$ for all $\|\delta\|_2 < R$, where δ is adversarial perturbation. Next, we discuss how to derive the certified radius R for CROWN [9], a state-of-the-art base classifier based certification method.

The key idea of CROWN is to bound each entry of the output vector, i.e., $H_l(\mathbf{x})$. In particular, when the output of the activation function (e.g., ReLU) can be bounded by two linear functions, CROWN shows that each entry $H_l(\mathbf{x})$ can be bounded by two linear functions $H_l^L(\mathbf{x})$ and $H_l^U(\mathbf{x})$. Formally, we have $H_l^L(\mathbf{x}) \leq H_l(\mathbf{x}) \leq H_l^U(\mathbf{x})$, where $l \in \{1, 2, \cdots, c\}$. Suppose an adversarial perturbation δ that satisfies $\|\delta\|_2 < r$ is added to $\mathbf{x}$. Then, we have the following:

$$\min_{\|\delta\|_2 < r} H_l^L(\mathbf{x} + \delta) \leq H_l(\mathbf{x} + \delta) \leq \max_{\|\delta\|_2 < r} H_l^U(\mathbf{x} + \delta), \tag{21}$$

where $l = 1, 2, \cdots, c$. The base classifier F still predicts label y for $\mathbf{x} + \delta$ when the lower bound of the yth entry (i.e., $\min_{\|\delta\|_2 < r} H_y^L(\mathbf{x} + \delta)$) is larger than the upper bounds of all other entries (i.e., $\max_{l \neq y} \max_{\|\delta\|_2 < r} H_l^U(\mathbf{x} + \delta)$). Therefore, CROWN derives the certified radius R as follows:

$$R = \max_{r} r \ s.t. \ \min_{\|\delta\|_2 < r} H_y^L(\mathbf{x} + \delta) > \max_{l \neq y} \max_{\|\delta\|_2 < r} H_l^U(\mathbf{x} + \delta).$$

Note that CROWN shows that $\min_{\|\delta\|_2 < r} H_l^L(\mathbf{x} + \delta)$ and $\max_{\|\delta\|_2 < r} H_l^U(\mathbf{x} + \delta)$ have closed-form solutions for $l = 1, 2, \cdots, c$.

Appendix C
Technical Details of SC based Certification [13]

Given a base classifier F (e.g., $g \circ f$ in SEaaS and g in REaaS), randomized smoothing builds a smoothed classifier h via adding random Gaussian noise to the testing input of the base classifier. The smoothed classifier h provably predicts the same label for the testing input when the ℓ_2-norm of the perturbation add to the testing input is less than a threshold (i.e., certified radius). Next, we respectively discuss how to build a smoothed classifier, how to derive the certified radius for a testing input, as well as how to train the base classifier to improve the certified radius.

Building a smoothed classifier: For simplicity, we use $\mathcal{N}(0, \sigma^2\mathbf{I})$ to denote a zero-mean isotropic Gaussian noise, where σ is the standard deviation and $\mathbf{I}$ is an identity matrix. Given an input $\mathbf{x}$, a base classifier F, and a zero-mean Gaussian noise $\mathcal{N}(0, \sigma^2\mathbf{I})$, we define the *label probability* $p_l, l \in \{1, 2, \cdots, c\}$, as follows:

$$p_l = \Pr(F(\mathbf{x} + \mathcal{N}(0, \sigma^2\mathbf{I})) = l). \tag{22}$$

Roughly speaking, p_l is the probability that the base classifier F predicts label l when taking $\mathbf{x} + \mathcal{N}(0, \sigma^2\mathbf{I})$ as input. The smoothed classifier h predicts the label with the largest label probability for the input $\mathbf{x}$:

$$h(\mathbf{x}) = \underset{l \in \{1,2,\cdots,c\}}{\text{argmax}} \ p_l. \tag{23}$$

In practice, it is usually computationally intractable to compute the exact label probabilities due to the complexity of the base classifier. In response, we sample N noise $\mathbf{n}_1, \mathbf{n}_2, \cdots, \mathbf{n}_N$ from $\mathcal{N}(0, \sigma^2\mathbf{I})$. For each $l \in \{1, 2, \cdots, c\}$, we denote $N_l = \sum_{j=1}^{N} \mathbb{I}(F(\mathbf{x} + \mathbf{n}_j) = l)$ (called *label frequency*), where $\mathbb{I}$ is an indicator function. The label (denoted as y) with the largest label frequency is predicted as the label of $\mathbf{x}$.

Computing certified radius for an input: Cohen et al. [13] proved that the smoothed classifier predicts the label y for the input $\mathbf{x}$ when the ℓ_2-norm of the adversarial perturbation is less than $\frac{\sigma}{2} \cdot (\Phi^{-1}(\underline{p_y}) - \Phi^{-1}(\overline{p_u}))$, where $\underline{p_y}$ is a lower bound of p_y and $\overline{p_u}$ is an upper bound of $\max_{l \neq y} p_l$. In other words, we have $h(\mathbf{x}+\delta) = y$ when $\|\delta\|_2 < R = \frac{\sigma}{2} \cdot (\Phi^{-1}(\underline{p_y}) - \Phi^{-1}(\overline{p_u}))$, where R is the certified radius.

To compute the certified radius, we need to estimate a lower bound of p_y and an upper bound of $\max_{l \neq y} p_l$. Cohen et al. proposed to estimate such lower or upper bounds using the standard one-sided Clopper-Pearson method [46] based on label frequencies. In particular, N_y follows a binomial distribution with parameters N and p_y. Based on the Clopper-Pearson method, we have the following lower bound for p_y:

$$\underline{p_y} = Beta(\alpha; N_y, N - N_y + 1), \tag{24}$$

where $1 - \alpha$ is the confidence level and $Beta(\alpha; \varsigma, \vartheta)$ is the αth quantile of the Beta distribution with shape parameters ς and ϑ. Intuitively, with probability at least $1 - \alpha$ over the randomness of the sampling of Gaussian noise, we have $p_y \geq \underline{p_y}$. Given the lower bound $\underline{p_y}$ and the fact that the summation of label probabilities is 1 (i.e., $\sum_{l=1}^{c} p_l = 1$), we can derive $1 - \underline{p_y}$ as an upper bound for $\max_{l \neq y} p_l$, i.e., $\max_{l \neq y} p_l \leq 1 - \underline{p_y}$. Given the lower bound of p_y and the upper bound of $\max_{l \neq y} p_l$, we have the certified radius $R = \sigma \cdot \Phi^{-1}(\underline{p_y})$. In other words, with probability at least $1 - \alpha$ over the randomness of the sampling of Gaussian noise, we have $g(\mathbf{x} + \delta) = y$ when $\|\delta\|_2 < \Phi^{-1}(\underline{p_y})$.

Training a base classifier with Gaussian noise: The certified radius for an input depends on whether the base classifier F can correctly classify the input with Gaussian noise. However, when the base classifier F is trained using the normal training examples, it is very likely that the base classifier cannot correctly predict the label for an input with Gaussian noise due to the difference between training and testing data distributions. Therefore, Cohen et al. [13] proposed to add Gaussian noise to the training inputs when training the base classifier. In particular, for each mini-batch of training examples, we add random Gaussian noise to each training input and then use them to update the base classifier. Such Gaussian noise based training method can significantly improve the certified radius [13].

APPENDIX D
ON THE TIGHTNESS OF EQUATION 5

We show that the upper and lower bounds in (5) are tight when the encoder consists of one linear layer. Suppose $f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$, where $\mathbf{W}$ and $\mathbf{b}$ are the weight matrix and bias vector of the linear encoder f. For simplicity, we respectively use $\mathbf{W}[i,:]$ and $\mathbf{b}[i]$ to denote the ith row of $\mathbf{W}$ and ith element of $\mathbf{b}$. Based on Theorem 3.2 in CROWN [9], we have $f_i^U(\mathbf{x}) = \mathbf{W}[i,:]\mathbf{x} + \mathbf{b}[i]$ and $f_i^L(\mathbf{x}) = \mathbf{W}[i,:]\mathbf{x} + \mathbf{b}[i]$ when $f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$ (we omit the details for simplicity). Next, we prove the right-hand side of Equation 5 is tight. To reach the goal, our idea is to show there exists δ' satisfying $\|\delta'\|_2 = \rho_k$ such that $f_i(\mathbf{x} + \delta') > \max_{\|\delta\|_2 < \rho_k} f_i^U(\mathbf{x} + \delta)$. We first derive an upper bound of $\max_{\|\delta\|_2 < \rho_k} f_i^U(\mathbf{x} + \delta)$.

$$\max_{\|\delta\|_2 < \rho_k} f_i^U(\mathbf{x} + \delta) \tag{25}$$

$$= \max_{\|\delta\|_2 < \rho_k} (\mathbf{W}[i,:]\mathbf{x} + \mathbf{W}[i,:]\delta + \mathbf{b}[i]) \tag{26}$$

$$= \mathbf{W}[i,:]\mathbf{x} + \mathbf{b}[i] + \max_{\|\delta\|_2 < \rho_k} \mathbf{W}[i,:]\delta \tag{27}$$

$$\leq \mathbf{W}[i,:]\mathbf{x} + \mathbf{b}[i] + \max_{\|\delta\|_2 < \rho_k} \|\mathbf{W}[i,:]\|_2 \|\delta\|_2 \tag{28}$$

$$< \mathbf{W}[i,:]\mathbf{x} + \mathbf{b}[i] + \|\mathbf{W}[i,:]\|_2 \rho_k \tag{29}$$

We have Equation 28 from 27 based on Cauchy-Schwartz inequality. Our derived upper bound is the same as the one obtained from the closed-form solution of f_i^U based on Corollary 3.3 in [9]. We let $\delta' = \frac{\rho_k \mathbf{W}[i,:]^T}{\|\mathbf{W}[i,:]\|_2}$, where T represents the transpose operation. We can verify that $\|\delta'\| = \rho_k$. Then, we have:

$$f_i(\mathbf{x} + \delta') \tag{30}$$

$$= \mathbf{W}[i,:]\mathbf{x} + \mathbf{b}[i] + \mathbf{W}[i,:] \frac{\rho_k \mathbf{W}[i,:]^T}{\|\mathbf{W}[i,:]\|_2} \tag{31}$$

$$= \mathbf{W}[i,:]\mathbf{x} + \mathbf{b}[i] + \|\mathbf{W}[i,:]\|_2 \rho_k. \tag{32}$$

Combining the above equations with Equation 25 and 29, we know that there exist δ' satisfying $\|\delta'\|_2 = \rho_k$ such that $f_i(\mathbf{x} + \delta') > \max_{\|\delta\|_2 < \rho_k} f_i^U(\mathbf{x} + \delta)$. Thus, we prove the tightness of the right-hand side of Equation 5. Similarly, we can prove the tightness of the left-hand side of Equation 5.

APPENDIX E
APPLYING REAAS TO NLP DOMAIN

Our REaaS provides two APIs: Feature-API and F2IPerturb-API. The Feature-API can be directly applied to the NLP domain. In particular, given a text as input, the Feature-API returns the feature vector produced by a text encoder for it. Next, we discuss how to adapt our F2IPerturb-API to the NLP domain.

Adapting our F2IPerturb-API to NLP domain: Given a text input and a feature-space certified radius, we can adapt our F2IPerturb-API to provide the input-space certified radius for the input text. In particular, we consider an attacker can replace a small number of words in a text with their synonyms such that the downstream classifier of a client makes incorrect prediction for it. For simplicity, given a text input $\mathbf{x}$, we use $\mathbf{x}[i]$ to denote the ith word in $\mathbf{x}$. Moreover, we use δ to denote an adversarial text perturbation whose length is the same as $\mathbf{x}$, where $\delta[i]$ is an empty string if we don't perturb $\mathbf{x}[i]$; we use $\mathbf{x} \oplus \delta$ to denote the perturbed text obtained by replacing $\mathbf{x}[i]$ as $\delta[i]$ if $\delta[i]$ is not an empty string; we use $\|\delta\|_0$ to denote the number of words in δ that are not empty string.

The input-space certified radius returned by our adapted F2IPerturb-API is the maximum number of words that can be replaced by their synonyms such that the predicted label by the downstream classifier is unchanged. Formally, given a text input $\mathbf{x}$ and a feature-space certified radius R_F, our adapted F2IPerturb-API aims to solve the following optimization problem:

$$R = \max_r r \tag{33}$$

$$s.t. \max_{\|\delta\|_0 < r} \|f(\mathbf{x} \oplus \delta) - f(\mathbf{x})\|_2 < R_F. \tag{34}$$

Similar to the image domain, we can use binary search to solve the above optimization problem. The key difference with the image domain is how to estimate the lower or upper bounds of $f_i(\mathbf{x} \oplus \delta)$. We note that Xu et al. [47] extend CROWN [9] for general perturbations, which can be applied to derive lower and

Algorithm 2: *F2IPerturb-API (NLP)*

```
Input from client: text $\mathbf{x}$ and feature-space certified radius $R_F$
Output for client: text-space certified radius $R$
$\rho^L, \rho^U \leftarrow 0, 10$
while $\rho^U - \rho^L > 1$ do
    $\rho_k = \lceil \frac{\rho^L + \rho^U}{2} \rceil$
    for $i = 1, 2, \cdots, d$ do
        $f_i^L, f_i^U \leftarrow$ ExtendedCrown$(i, \mathbf{x}, f)$
        $L_i = \min_{\|\delta\|_0 \le \rho_k} f_i^L(\mathbf{x} + \delta) - f_i(\mathbf{x})$
        $U_i = \max_{\|\delta\|_0 \le \rho_k} f_i^U(\mathbf{x} + \delta) - f_i(\mathbf{x})$
    end for
    $R'_F = \sqrt{\sum_{i=1}^d \max(L_i^2, U_i^2)}$
    if $R'_F < R_F$ then
        $\rho^L = \rho_k$
    else
        $\rho^U = \rho_k$
    end if
end while
return $\rho^L$
```

upper bounds (denoted as $f_i^L(\mathbf{x}\oplus\delta)$ and $f_i^U(\mathbf{x}\oplus\delta)$) of $f_i(\mathbf{x}\oplus\delta)$. Moreover, they also provide a computation-efficient method to compute $\min_{\|\delta\|_0<r} f_i^L(\mathbf{x} \oplus \delta)$ and $\max_{\|\delta\|_0<r} f_i^U(\mathbf{x} \oplus \delta)$ (please see Theorem 2 in [47] for details). Given those bounds, our adapted F2IPerturb-API can be used to obtain the certified radius. Algorithm 2 shows our complete algorithm. The function ExtendedCrown uses the extended CROWN [47] to obtain lower or upper bounds.

Experimental setup: We use SST-2 dataset [48] as the pre-training dataset and train a text encoder using supervised learning. Following [47], we use LSTM as the encoder architecture. Moreover, we use public implementation from [47] in our experiments. We consider the same word substitutions as previous work [33], [49] when deriving the input-space certified radius for our F2IPerturb-API. We use IMDB dataset [45] as the downstream dataset, which is a benchmark dataset used for sentiment analysis in the NLP domain. We train a logistic regression classifier as the downstream classifier. Moreover, we use CROWN [9] as BC based certification to derive the certified radius of the downstream classifier.

Appendix F
Extending StolenEncoder [34] to Steal Encoders with our new F2IPerturb-API

We first introduce StolenEncoder [34] and then discuss how to extend it to steal encoders with our F2IPerturb-API.

StolenEncoder: Suppose an attacker has an unlabeled surrogate dataset $\mathcal{D}$. Given black-box access to an encoder (i.e., an attacker can query the Feature-API in our terminology), the attacker can use StolenEncoder to steal a target encoder f. Suppose f_s is the stolen encoder. StolenEncoder aims to solve the following optimization problem:

$$\min_{f_s} \mathcal{L}(\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}}[d(f(\mathbf{x}), f_s(\mathbf{x})) + \kappa d(f(\mathbf{x}), f_s(\mathcal{A}(\mathbf{x})))], \tag{35}$$

where κ is a hyperparameter, d is a distance metric (e.g., ℓ_2-distance), and $\mathcal{A}$ is the data augmentation component (e.g., RandomHorizontalFlip, ColorJitter, and RandomGrayScale). The total number of queries to the Feature-API is $|\mathcal{D}|$ as a single query is made for each input in $\mathcal{D}$. StolenEncoder uses standard stochastic gradient descent (SGD) to solve the above optimization problem.

Extending StolenEncoder with our new F2IPerturb-API: An attacker can also query F2IPerturb-API in our REaaS. We extend StolenEncoder to leverage our F2IPerturb-API to steal the target encoder f. In particular, given an input image $\mathbf{x}$ and a feature-space certified radius R_F for $\mathbf{x}$, our F2IPerturb-API returns an input-space certified radius R. Therefore, we know the following based on the output of F2IPerturb-API:

$$\|f(\mathbf{x}) - f(\mathbf{x}+\delta)\|_2 < R_F \text{ for } \forall\delta, \|\delta\|_2 < R. \tag{36}$$

We can train the stolen encoder f_s such that it also satisfies Equation 36. Suppose we have a surrogate dataset $\mathcal{D}$. Given an input $\mathbf{x} \in \mathcal{D}$, we first randomly sample a radius from $[R_F^{min}, R_F^{max}]$ and view it as the feature-space certified radius $R_F^{\mathbf{x}}$. Then, we query F2IPerturb-API to obtain the input-space certified radius $R^{\mathbf{x}}$. To make the stolen encoder f_s satisfy Equation 36 for $\mathbf{x}$, we randomly sample an input-space perturbation $\delta^{\mathbf{x}}$ that satisfies $\|\delta^{\mathbf{x}}\|_2 = R^{\mathbf{x}}$. Then, we define the following loss:

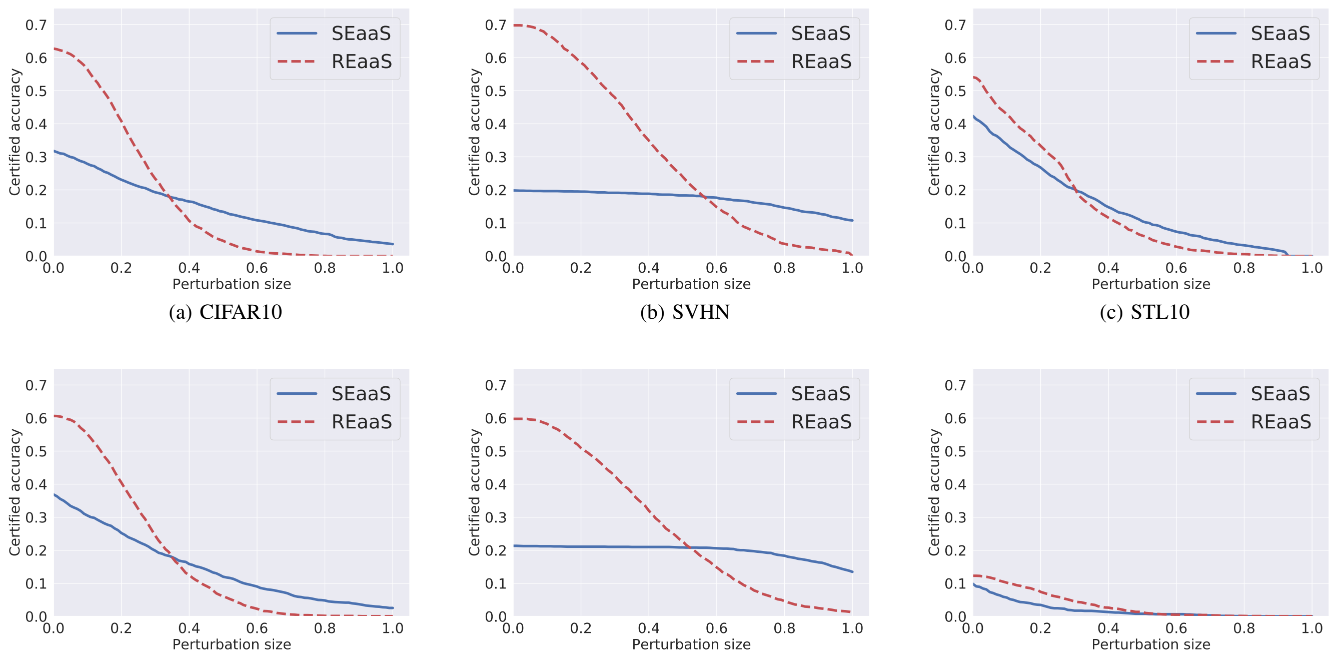
$$\mathcal{L}'(\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} d(f_s(\mathbf{x}), f_s(\mathbf{x}+\delta^{\mathbf{x}}))\cdot \mathbb{I}(d(f_s(\mathbf{x}), f_s(\mathbf{x}+\delta^{\mathbf{x}})) > R_F^{\mathbf{x}}), \tag{37}$$

where $\mathbb{I}$ is an indicator function whose output is 1 if the condition is satisfied and 0 otherwise. Intuitively, the loss term in $\mathcal{L}'(\mathcal{D})$ for an input $\mathbf{x}$ is 0 if $d(f_s(\mathbf{x}), f_s(\mathbf{x}+\delta^{\mathbf{x}})) \le R_F^{\mathbf{x}}$ and is non-zero otherwise. Combining with the loss of StolenEncoder, our final optimization problem is as follows:

$$\min_{f_s} \mathcal{L}_1(\mathcal{D}) = \mathcal{L}(\mathcal{D}) + \gamma\mathcal{L}'(\mathcal{D}), \tag{38}$$

where γ is a hyperparameter and $\mathcal{L}$ is defined in Equation 35. The total number of queries is $2|\mathcal{D}|$ as we respectively make one query to the Feature-API and F2IPerturb-API for each input in $\mathcal{D}$. Similarly, we use stochastic gradient descent to solve the optimization problem. Note that our extended StolenEncoder reduces to StolenEncoder when $\gamma = 0$.

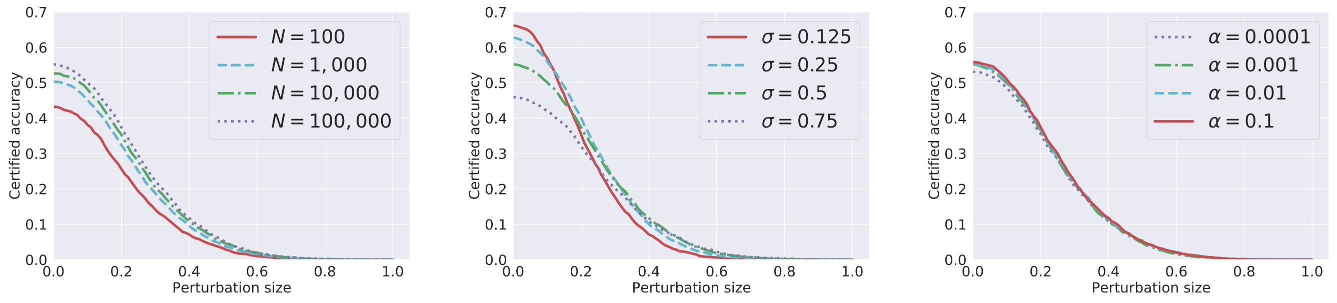
Experimental setup: We use Tiny-ImageNet as the pre-training dataset and use CIFAR10 as the downstream task. Moreover, we randomly sample 10,000 images from STL10 dataset as the surrogate dataset $\mathcal{D}$. We adopt the same κ, d, and $\mathcal{A}$ as StolenEncoder. We set R_F^{min} and R_F^{max} to be 0.01 and 0.5, respectively. As we use the same surrogate dataset, we give advantages to the extended StolenEncoder since its total number of queries is twice of StolenEncoder. Following [34], we use *Stolen Accuracy* as evaluation metrics. In particular, given a stolen encoder and a downstream task, Stolen Accuracy is the classification accuracy of the downstream classifier built upon the stolen encoder for the downstream task.



(a) CIFAR10 (b) SVHN (c) STL10

(d) CIFAR10 (e) SVHN (f) Tiny-ImageNet

Fig. 7: Comparing certified accuracy vs. perturbation size of SC in SEaaS and REaaS for different downstream datasets, where the pre-training dataset is Tiny-ImageNet (first row) and STL10 (second row).



(a) N (b) σ (c) α

Fig. 8: Impact of N, σ, and α on certified accuracy of SC in REaaS. The pre-training dataset is Tiny-ImageNet and the downstream dataset is CIFAR10.