PatchCURE: Improving Certifiable Robustness, Model Utility, and Computation Efficiency of Adversarial Patch Defenses

Chong Xiang, Tong Wu, and Sihui Dai, *Princeton University;* Jonathan Petit, *Qualcomm Technologies, Inc.;* Suman Jana, *Columbia University;* Prateek Mittal, *Princeton University*

https://www.usenix.org/conference/usenixsecurity24/presentation/xiang-chong

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

PATCHCURE: Improving Certifiable Robustness, Model Utility, and Computation Efficiency of Adversarial Patch Defenses

Chong Xiang[1], Tong Wu[1], Sihui Dai[1], Jonathan Petit[2], Suman Jana[3], Prateek Mittal[1]
[1]*Princeton University,* [2]*Qualcomm Technologies, Inc.,* [3]*Columbia University*

Abstract

State-of-the-art defenses against adversarial patch attacks can now achieve strong certifiable robustness with a marginal drop in model utility. However, this impressive performance typically comes at the cost of 10-100× more inference-time computation compared to undefended models – the research community has witnessed an intense three-way trade-off between certifiable robustness, model utility, and computation efficiency. In this paper, we propose a defense framework named PATCHCURE to approach this trade-off problem. PATCHCURE provides sufficient "knobs" for tuning defense performance and allows us to build a family of defenses: the most robust PATCHCURE instance can match the performance of any existing state-of-the-art defense (without efficiency considerations); the most efficient PATCHCURE instance has similar inference efficiency as undefended models. Notably, PATCHCURE achieves state-of-the-art robustness and utility performance across all different efficiency levels, e.g., 16-23% absolute clean accuracy and certified robust accuracy advantages over prior defenses when requiring computation efficiency to be close to undefended models. The family of PATCHCURE defenses enables us to flexibly choose appropriate defenses to satisfy given computation and/or utility constraints in practice.[1]

1 Introduction

The adversarial patch attack [4] against computer vision models overlays a malicious pixel patch onto an image to induce incorrect model predictions. Notably, this attack can be realized in the physical world by printing and attaching the patch to real-world objects: images/photos captured from that physical scene would become malicious. The physically realizable nature of patch attacks raises significant concerns in security and safety-critical applications like autonomous vehicles [14], face authentication [49], security surveillance [62], and thus, motivates the design of defense mechanisms.

Among all the efforts in mitigating adversarial patch attacks, research on certifiably robust defenses stands out with remarkable progress [7, 8, 25, 26, 44, 54, 55]. These defenses aim to provide provable/certifiable robustness guarantees that hold for any attack strategy within a given threat model, including attackers with full knowledge of the defense algorithms and model details. The property of certifiable robustness provides a pathway toward ending the arms race between attackers and defenders. Notably, the state-of-the-art certifiably robust defenses [55] now can achieve high certifiable robustness while only marginally affecting the model utility (e.g., 1% accuracy drop on ImageNet [12]).

However, the impressive robustness and utility performance comes at the cost of overwhelming computation overheads. Many defenses [7, 26, 44, 55] require 10-100× more inference-time computation compared to undefended models, and this makes them computationally prohibitive for real-world deployment. On the other hand, there exist efficient defenses [34, 54, 67] with small computation overheads, but they all suffer from a significant drop in certifiable robustness and model utility (e.g., 30+% absolute drop from the state-of-the-art). *The research community has witnessed an intense three-way trade-off between certifiable robustness, model utility, and computation efficiency [59].*

Unfortunately, there is no existing method to systematically study this three-way trade-off problem. State-of-the-art (inefficient) defenses, e.g., PatchCleanser [55], do not have a design point to achieve similar computation efficiency as undefended models. Efficient defenses, e.g., PatchGuard [54], lack approaches to trade part of their efficiency for better utility and robustness performance. In this paper, we make a *first* attempt to address these challenges: we propose a defense framework named PATCHCURE to consolidate **C**ertifiable **R**obustness, Model **U**tility, and Computation **E**fficiency.

Contributions. The main contribution of PATCHCURE is identifying a key factor of the three-way trade-off problem: the model *receptive field*, i.e., the image region each

[1]Our source code is available at https://github.com/inspire-group/PatchCURE.

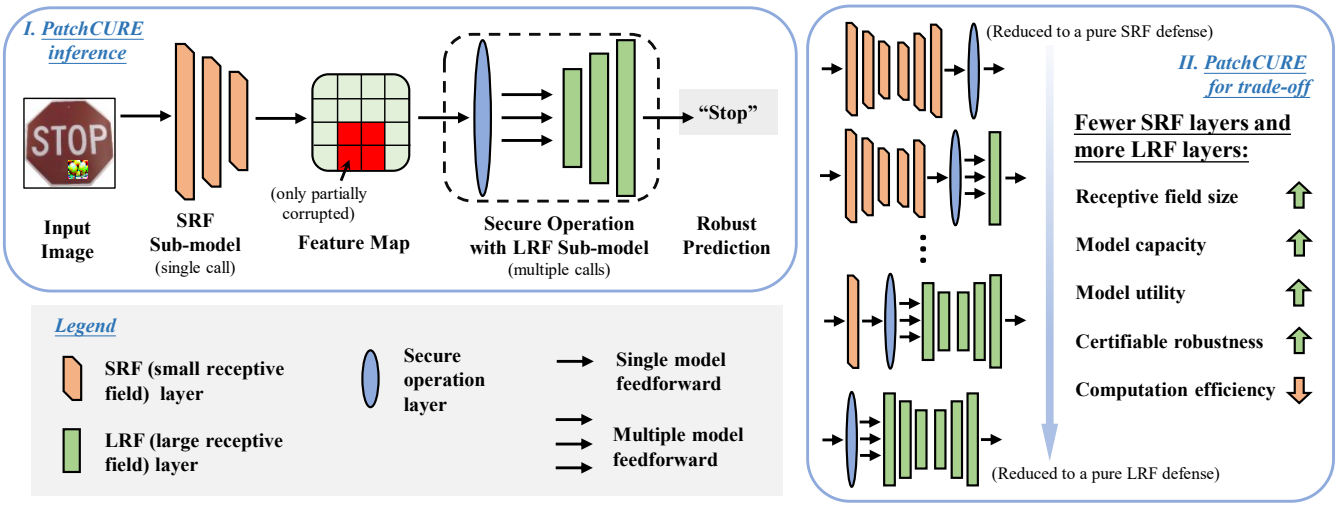

Figure 1: **PatchCURE overview. I. PatchCURE inference (Section 3.2):** Given an input image, we first call an SRF (small receptive field) sub-model *once* to extract an intermediate feature map. The use of SRF ensures that only part of the features is corrupted. Next, we leverage secure operation, which typically involves *multiple* calls to an LRF (large receptive field) sub-model, for final predictions. **II. PatchCURE for the trade-off problem (Section 3.5).** We can adjust the combination of SRF and LRF layers to balance the three-way trade-off. As we use fewer SRF layers and more LRF layers, the defense model (with a fixed number of total layers) normally has larger receptive fields, larger model capacity, better model utility, and higher certifiable robustness, but poorer computation efficiency.

extracted feature is looking at. We observe that most existing defenses can be categorized as either small receptive field (SRF) defenses or large receptive field (LRF) defenses. SRF defenses [7, 25, 44, 54] use a model with SRF for feature extraction so that there is only a limited number of corrupted features that marginally interfere with the prediction. In contrast, LRF defenses [32, 55] aim to directly mask out the entire patch from the input images and then use a high-performance model with LRF for final predictions. Interestingly, SRF techniques criticize LRF defenses for excessive computation overheads (LRF techniques require multiple model predictions on different masked images) while LRF techniques criticize SRF defenses for poor model utility (SRF limits the information received by each feature and hurts model capacity). In PatchCURE, we propose a *unified* generalization of SRF and LRF techniques.

PatchCURE: a framework that unifies SRF and LRF techniques. We provide an overview of PatchCURE in Figure 1 (see the figure caption for more details). A PatchCURE defense has three modules: an SRF sub-model, an LRF sub-model, and a secure operation "layer" (an abstract layer representing a robust prediction algorithm/procedure, e.g., the double-masking procedure from PatchCleanser [55]). To make a prediction on an input image (upper left of the figure), PatchCURE first uses the SRF sub-model to extract intermediate features and then leverages the secure operation layer, together with multiple calls to the LRF sub-model, for a final robust prediction. To balance the three-way trade-off (right of the figure), PatchCURE adjusts the portion/combination of SRF and LRF layers within the end-to-end defense model.

State-of-the-art efficient defense instances. Using our PatchCURE framework, we can build state-of-the-art efficient defenses (with similar inference speed as undefended models). In doing so, we first design a ViT-SRF architecture (a ViT [13] variant with SRF) to instantiate the SRF sub-model. Next, to minimize computation overheads, we instantiate a lightweight LRF sub-model, which is composed of a linear feature aggregation followed by a classification head. Finally, we use the double-masking algorithm [55] as the secure operation. In Figure 2, we demonstrate that the most efficient instances of PatchCURE (top stars) have similar inference speed as undefended ViT (cross), while significantly outperforming prior efficient defenses (top triangles) in terms of certifiable robustness (more than 18% absolute improvement). Moreover, these efficient defense instances even outperform all but one existing inefficient defense (bottom squares/triangles).[2]

A systematic way to balance the three-way trade-off. Furthermore, we demonstrate that the PatchCURE framework provides sufficient knobs to balance the three-way trade-off. We systematically explore different defense parameters and plot a family of PatchCURE defenses in Figure 2. As shown in the figure, PatchCURE instances

[2]For simplicity, we did not plot the model utility performance in Figure 2; we will demonstrate PatchCURE's superiority in model utility in Section 4.

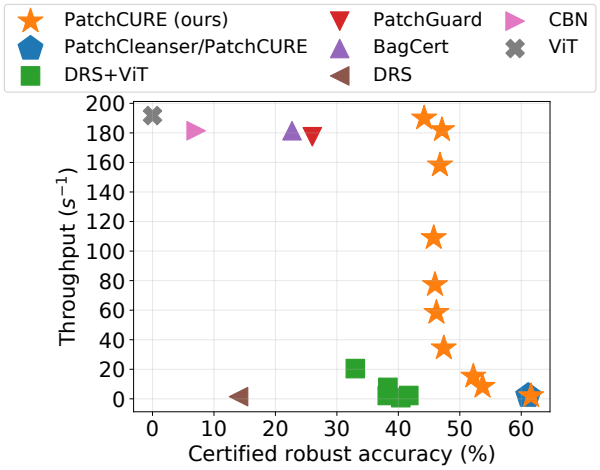

Figure 2: Certified robust accuracy and inference throughput (img/s) for different defenses on ImageNet-1k [12]: (i) our PATCHCURE instances with different settings; (ii) PatchCleanser [55] – also a special instance of PATCHCURE; (iii) DRS+ViT, including Smoothed ViT [44], ECViT [7], and ViP [26]; (iv) PatchGuard [54]; (v) BagCert [34]; (vi) De-Randomized Smoothing (DRS) [25]; (vii) Clipped BagNet (CBN) [67]; (viii) undefended ViT [13]. Certified robustness considers one 2%-pixel square patch anywhere on the image.

(stars) can easily bridge the robustness gap between the most efficient PATCHCURE (top stars) and the state-of-the-art PatchCleanser [55] (pentagon; also a special instance of PATCHCURE as discussed in Section 3). Moreover, we note that PATCHCURE also achieves the best robustness performance (and utility performance) across all different efficiency levels. With PATCHCURE, we can flexibly build the optimal defense that satisfies certain computation efficiency or model utility requirements.

We summarize our contributions as follows.

1. We propose a PATCHCURE defense framework that unifies disparate SRF and LRF techniques for approaching the three-way trade-off problem.
2. We designed a ViT-SRF architecture to instantiate efficient PATCHCURE defenses and achieve 18+% absolute robustness improvements from prior efficient defenses.
3. We experimentally demonstrate that PATCHCURE provides sufficient knobs to balance robustness, utility, and efficiency, and also achieves state-of-the-art robustness and utility performance across different efficiency levels.

2 Preliminaries

In this section, we formulate the research problem, discuss the important concept of model receptive fields, and present an overview of existing SRF and LRF defense techniques.

2.1 Problem Formulation

In this subsection, we detail image classification models, adversarial patch attacks, certifiable robustness, and three performance dimensions studied in the trade-off problem.

Image classification models. In this paper, we study image classification models. We let $\mathcal{X} \subset [0,1]^{H\times W\times C}$ denote the input image space: H, W, C correspond to the height, width, and number of channels of the image; $[0,1]$ is the range of normalized pixel values. We next denote the label space as $\mathcal{Y} = \{0, 1, \cdots, N-1\}$, where N is the total number of classes. Finally, we denote an image classification model as $\mathbb{M} : \mathcal{X} \to \mathcal{Y}$, which maps an input image $\mathbf{x} \in \mathcal{X}$ to its prediction label $y \in \mathcal{Y}$.

We further use $\mathcal{F} \in \mathbb{R}^{H'\times W'\times C'}$ to denote the space of the intermediate feature map. We will overload the notation $\mathbb{M}$ as (1) a feature extractor that maps an input image to intermediate features $\mathbb{M}_0 : \mathcal{X} \to \mathcal{F}$, or (2) a classifier that makes predictions based on the intermediate feature map $\mathbb{M}_1 : \mathcal{F} \to \mathcal{Y}$.

Adversarial patch attack. The adversarial patch attack [4] is a type of test-time evasion attack. Given a model $\mathbb{M}$, an image $\mathbf{x}$, and its correct label y, the attacker aims to generate an adversarial image $\mathbf{x}' \in \mathcal{A}(\mathbf{x}) \subset \mathcal{X}$ satisfying certain constraints $\mathcal{A}$ to induce incorrect predictions $\mathbb{M}(\mathbf{x}') \neq y$. The patch attack constraint $\mathcal{A}$ allows the attacker to introduce *arbitrary* pixel patterns within *one* restricted image region (i.e., the patch region); the patch region can be at *any location* chosen by the attacker. This constraint threat model is widely used in prior certifiably robust defenses [7, 25, 26, 34, 44, 54, 55, 67].

Formally, we use a binary pixel tensor $\mathbf{r} \in \{0,1\}^{H\times W}$ to represent each restricted patch region: pixels within the region are set to zeros, and others are ones. We next let $\mathcal{R}$ denote a set of all possible patch regions $\mathbf{r}$ (e.g., patches at all different image locations). Then, we can formalize the constraint as $\mathcal{A}_{\mathcal{R}}(\mathbf{x})$ as $\{\mathbf{r}\odot\mathbf{x} + (\mathbf{1}-\mathbf{r})\odot\mathbf{x}' \mid \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \mathbf{r} \in \mathcal{R}\}$, where $\odot$ is the element-wise multiplication operator, and $\mathbf{x}'$ contains malicious pixels. When clear from the context, we drop $\mathcal{R}$ and simplify $\mathcal{A}_{\mathcal{R}}$ as $\mathcal{A}$.

Certifiable robustness. We study defense algorithms whose robustness can be formally proved or certified. That is, given an image $\mathbf{x}$, its correct label y, and a patch attack constraint $\mathcal{A}_{\mathcal{R}}$, we aim to build a defense model $\mathbb{M}$ such that we can certify that

$$\forall \mathbf{x}' \in \mathcal{A}_{\mathcal{R}}(\mathbf{x}), \mathbb{M}(\mathbf{x}') = \mathbb{M}(\mathbf{x}) = y \tag{1}$$

In addition to the defense model $\mathbb{M}$, we will also develop a special certification procedure $\mathbb{C} : \mathcal{X} \times \mathcal{Y} \times \mathbb{P}(\mathcal{A}_{\mathcal{R}}) \to \{\texttt{True}, \texttt{False}\}$ to determine whether Equation 1 holds for an image $\mathbf{x}$, its label y, and threat model $\mathcal{A}_{\mathcal{R}}$. Note that the universal quantifier $\forall$ requires the certification procedure $\mathbb{C}$ to account for all possible attackers within the threat model $\mathcal{A}_{\mathcal{R}}$, who could have full knowledge of the defense algorithm and setup. This certification ensures that any robustness we claim will not be compromised by adaptive at-

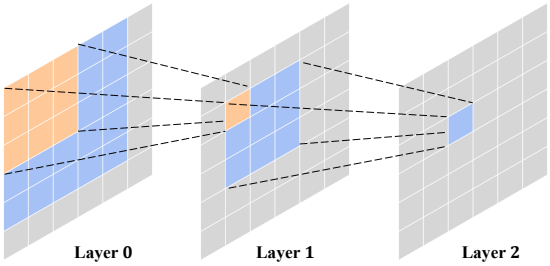

Figure 3: **Illustration of model receptive field.** For a convolutional network with a kernel size of 3 and stride size of 1, the blue cell in Layer 2 is affected by 3×3 cells in Layer 1 and 5×5 cells in Layer 0 (the model input).

tackers, which is a significant advantage over empirical defenses [11, 18, 36, 37, 41, 52] without formal robustness guarantees.

Three performance dimensions. We will study the trade-off problem between three performance dimensions. The first is *certifiable robustness*, as discussed above. We will run the certification procedure over all images of a labeled test dataset and report certified robust accuracy as the fraction of images for which the procedure $\mathbb{C}$ returns `True`. The second is *model utility* – model accuracy on clean images without adversarial patches, also termed as clean accuracy.The third is *computation efficiency*; we measure it via inference throughput – the number of images a model can process within every second.

2.2 Receptive Fields of Vision Models

The receptive field of a vision model is the input image region that each extracted feature is looking at, or is affected by; we provide a visual example in Figure 3. A conventional vision model first extracts features from different image regions (with different focuses), and then aggregates all extracted features and makes an informed prediction.

There has been a line of research studying the effect of receptive field size on model performance [1, 3, 24, 30]. Normally, larger receptive fields lead to larger model capacity and thus better model performance (as long as the model is well-trained with enough data). For example, a convolutional network like ResNet [20] usually has better performance when it has deeper layers with larger receptive fields; the emerging powerful Vision Transformer (ViT) [13] architecture allows features in every layer to have a large/global receptive field of the entire image. In this paper, we will demonstrate that the receptive field size also plays an important role in the three-way trade-off problem for certifiably robust patch defenses.

2.3 Overview of SRF and LRF Defenses

In this subsection, we provide an overview of existing small receptive field (SRF) and large receptive field (LRF) defense techniques.

SRF defenses. The use of SRF was first explicitly discussed in PatchGuard [54]. Its key insight is that: using models with SRF for feature extraction can limit the number of features that see (and hence, are affected by) the adversarial patch. The maximum number of corrupted features p^f can be computed as

$$p^f = \lceil (p + r - 1)/s \rceil \tag{2}$$

where p is the patch size in the image space, r is the receptive field size of the SRF model, s is the effective stride of the model receptive field (i.e., the distance between the centers of receptive fields of two adjacent features). The correctness of this formula was proven in [54]. Next, an SRF defense performs lightweight secure feature aggregation (e.g., clipping, majority voting) on the partially corrupted feature map for robust predictions. The idea of SRF has been shown effective and adopted by many defenses [7, 25, 26, 34, 44, 67].

Strength: high efficiency. Since the secure aggregation operates on the final feature map, its computation complexity can be as low as a linear transformation layer. Therefore, the computation of SRF defenses is dominated by the feedforward pass of the SRF model, which can be made as efficient as standard CNN or ViT (see Section 3.3 for more details).

Weakness: poor utility. However, though the use of SRF bounds the number of corrupted features, it also limits the information received by each feature. As a result, the model utility is affected. For example, the clean accuracy on ImageNet-1k [12] reported in the original PatchGuard paper [54] is only 54.6% while standard ResNet-50 [20] and ViT-B [13] can achieve 80+% accuracy without additional training data [19, 50].

LRF defenses. The key idea of LRF defenses is to remove the patch from the input image and then use a high-performance LRF model to recover robust predictions. Its most representative defense, PatchCleanser [55], proposed a *double-masking* algorithm that applies different pixel masks to the input image and analyzes model predictions (with LRF) on different masked images to recover the correct prediction. The intuition behind the masking defense is that: model predictions on images with different masks usually have a unanimous agreement on clean image (predictions are robust to partial occlusions), but disagree when there is an adversarial patch (when the patch is completely masked, the model prediction changes to benign). We additionally provide pseudocode for the PatchCleanser [55] in Appendix B.

Strength: high utility&robustness. Since LRF defenses operate on the input image, they are compatible with any high-performance image classifiers (which usually have LRF). This allows LRF defenses like PatchCleanser [55] to maintain a very high model utility (e.g., 1% drops on ImageNet-1k from vanilla ResNet and ViT) while achieving state-of-the-art certifiable robustness.

Weakness: low computation efficiency. The downside of image-space operations of LRF defenses is that it normally

Table 1: Comparison for the SRF defense, the LRF defense, and PATCHCURE

Technique	Secure Operation Loc.	Utility&Robustness	Efficiency
SRF (e.g., [54])	Final (feature) layer	Poor/Fair	Good
LRF (e.g., [55])	Input (image) layer	Good	Poor
PATCHCURE (ours)	Flexible	Tunable	Tunable

requires performing model feedforward *multiple times* on different masked images. As a result, LRF defenses can easily incur 10+ times more computation compared to undefended models, making it impractical for real-world deployment.

In summary, SRF and LRF techniques are widely viewed as two distinct approaches to building defenses with different strengths and weaknesses [59]. In the next section, we will discuss how PATCHCURE unifies these disparate SRF and LRF techniques to approach the three-way trade-off problem.

3 PATCHCURE Framework

In this section, we discuss our PATCHCURE design. We start with our defense insights and the full PATCHCURE algorithm. We then elaborate on approaches for building SRF models, discuss robustness certification, and conclude with PATCHCURE's instantiation strategy.

3.1 PATCHCURE Insights

Section 2.3 demonstrated a tension between certifiable robustness, model utility, and computation efficiency: existing defenses with SRF or LRF techniques struggle to perform well in all three dimensions. In Table 1, we summarize and compare different properties of SRF and LRF techniques. This table helps us identify a key factor of the three-way trade-off problem, which inspires our PATCHCURE design.

Key factors in the trade-off problem. From Table 1, we find that different *receptive field sizes* lead to different secure operation locations (i.e., where the defense logic is applied to) and eventually different defense properties. SRF defenses (e.g., PatchGuard [54]) can operate on the final feature layer. This design only requires SRF defenses to perform *one* expensive model feedforward; that is, their secure operation can reuse the extracted feature map to achieve high computation efficiency. However, the use of SRF hurts model utility and robustness. On the other hand, LRF defenses (e.g., PatchCleanser [55]) operate on the input image. This makes them compatible with high-performance LRF models to achieve high model utility (and robustness). However, image-space defenses suffer from low efficiency because they need to perform *multiple* expensive end-to-end model feedforward passes on different modified images (e.g., masked images).

PATCHCURE as a unified defense. To benefit from the strengths of SRF and LRF defenses, we propose a unified PATCHCURE defense that leverages both SRF and LRF techniques (bottom row of Table 1). Recall our defense overview in Figure 1. A PATCHCURE defense has three modules: an SRF sub-model, an LRF sub-model, and a secure operation layer/procedure that leverages the LRF sub-model. At the inference time, we call the SRF sub-model *once* to extract intermediate features and then activate the secure operation with *multiple* calls to the LRF sub-models to remove corrupted features for a robust final prediction. Notably, the SRF component allows us to reuse the extracted intermediate feature map to **save computation**. In the meanwhile, the use of SRF also ensures that only a limited number of features are corrupted, so we can still use secure operation techniques like double-masking algorithm [55] to remove corrupted features for **certifiable robustness** (note that all intermediate features are likely to be corrupted without the use of SRF [54]). Furthermore, the use of LRF makes the end-to-end model have large receptive fields so that we can retain **high model utility**.

3.2 PATCHCURE Algorithm

In this subsection, we discuss details of PATCHCURE algorithm, which includes model construction, i.e., how to build SRF and LRF sub-models, model inference, and robustness certification procedures. We provide their pseudocode in Algorithm 1.

Model construction (Lines 1-6). We build SRF and LRF sub-models based on existing state-of-the-art models, which we call *base models*. First, we select an off-the-shelf base model architecture $\mathbb{M}_{\text{b}}$ (e.g., ViT [13]) and pick its k^{th} backbone layer (e.g., the second self-attention layer in ViT) as the *splitting layer*. Second, we split the model at this layer into two sub-models using the $\text{SPLIT}(\cdot)$ procedure. We have $\mathbb{M}_0, \mathbb{M}_1 = \text{SPLIT}(\mathbb{M}_{\text{b}}, k)$ s.t. $\mathbb{M}(\mathbf{x}) = \mathbb{M}_1(\mathbb{M}_0(\mathbf{x}))$; $\mathbb{M}_0$ contains layers with indices from 0 to $k-1$ while $\mathbb{M}_1$ has the remaining layers. Third, we keep the second sub-model $\mathbb{M}_1$ unchanged as the LRF sub-model $\mathbb{M}_{\text{lrf}} = \mathbb{M}_1$ (note that vanilla models normally have LRF for high utility) and convert the first sub-model $\mathbb{M}_0$ into an SRF sub-model $\mathbb{M}_{\text{srf}} = \text{TOSRF}(\mathbb{M}_0, \mathbf{rf})$ with a receptive field size of **rf**. We will discuss the details of $\text{TOSRF}(\cdot)$ in Section 3.3 – how to build attention-based (e.g., ViT) and convolution-based (e.g., ResNet) SRF models with high computation efficiency. Finally, we *conceptually* insert a secure operation "layer" between the SRF sub-model and the LRF sub-model (recall Figure 1). The secure operation layer represents a robust prediction algorithm/procedure $\text{SO}(\cdot)$; its design choices include the double-masking algorithm proposed in PatchCleanser [55] and the Minority Reports algorithm [32]. We will focus on the double-masking algorithm in this paper since it is the *state-of-the-art* certifiably robust algorithm to *recover* correct predictions. We provide details of the double-masking algorithm in Appendix B. We also discuss alternative secure operation choices in Section 5.

Algorithm 1 PATCHCURE algorithm

Require: Base model $\mathbb{M}_b$, splitting layer index k, SRF size **rf**, secure operation algorithm $\mathrm{SO}(\cdot)$ and its certification procedure $\text{SO-CERT}(\cdot)$, secure operation parameters $\mathcal{M}$, patch threat model $\mathcal{A}_{\mathcal{R}}$

1: **procedure** PCURE-CONSTRUCT($\mathbb{M}_b, k$, **rf**)
2: $\mathbb{M}_0, \mathbb{M}_1 \leftarrow \text{SPLIT}(\mathbb{M}_b, k)$ ▷ Split model at k^{th} layer
3: $\mathbb{M}_{\text{srf}} \leftarrow \text{TOSRF}(\mathbb{M}_0, \textbf{rf})$ ▷ Convert $\mathbb{M}_0$ to SRF
4: $\mathbb{M}_{\text{lrf}} \leftarrow \mathbb{M}_1$ ▷ Keep $\mathbb{M}_1$ as LRF
5: **return** $\mathbb{M}_{\text{srf}}, \mathbb{M}_{\text{lrf}}$
6: **end procedure**

7: **procedure** PCURE-INFER($\mathbf{x}, \mathbb{M}_{\text{srf}}, \mathbb{M}_{\text{lrf}}, \mathcal{M}$)
8: $\mathbf{f} \leftarrow \mathbb{M}_{\text{srf}}(\mathbf{x})$ ▷ Extract SRF features
9: $\hat{y} \leftarrow \mathrm{SO}(\mathbf{f}, \mathbb{M}_{\text{lrf}}, \mathcal{M})$ ▷ Secure operation on $\mathbf{f}$
10: **return** $\hat{y}$
11: **end procedure**

12: **procedure** PCURE-CERTIFY($\mathbf{x}, y, \mathbb{M}_{\text{srf}}, \mathbb{M}_{\text{lrf}}, \mathcal{M}, \mathcal{A}_{\mathcal{R}}$)
13: $\mathbf{f} \leftarrow \mathbb{M}_{\text{srf}}(\mathbf{x})$ ▷ Extract SRF features
14: $\mathcal{A}^f_{\mathcal{R}} \leftarrow \text{MAP}(\mathbb{M}_{\text{srf}}, \mathcal{A}_{\mathcal{R}})$ ▷ To feature-space adversary
15: $c \leftarrow \text{SO-CERT}(\mathbf{f}, \mathbb{M}_{\text{lrf}}, \mathcal{M}, \mathcal{A}^f_{\mathcal{R}}, y)$ ▷ Certification
16: **return** c
17: **end procedure**

Model inference (Lines 7-11). Once we build the SRF and LRF sub-models and decide on the secure operation algorithm, the inference is straightforward. Given an input image $\mathbf{x}$, we first use the SRF model $\mathbb{M}_{\text{srf}}(\mathbf{x})$ to extract features $\mathbf{f} = \mathbb{M}_{\text{srf}}(\mathbf{x})$. Next, we activate the secure operation $\mathrm{SO}(\cdot)$ on the feature tensor for the final prediction: $\hat{y} = \mathrm{SO}(\mathbf{f}, \mathbb{M}_{\text{lrf}}, \mathcal{M})$. The procedure $\mathrm{SO}(\cdot)$ takes as inputs a feature/image tensor $\mathbf{f}$, an LRF model $\mathbb{M}_{\text{lrf}}$, and any parameters $\mathcal{M}$ of the secure operation algorithm, and outputs robust prediction labels $\hat{y} \in \mathcal{Y}$.

Robustness certification (Lines 12-17). Given a *clean* image $\mathbf{x}$, we first extract features using the SRF model $\mathbf{f} = \mathbb{M}_{\text{srf}}(\mathbf{x})$. Next, we map the image-space adversary to the feature-space adversary: the procedure $\text{MAP}(\cdot)$ uses Equation 2 to calculate the number of corrupted features p^f based on the adversarial patch size p in the input image, convert each image-space patch region $\mathbf{r}$ to feature-space corruption region $\mathbf{r}^f$ accordingly, and then generate the feature-space adversary threat model $\mathcal{A}^f_{\mathcal{R}}$ for further robustness analysis. Finally, we call the robustness certification procedure of the secure operation algorithm on the feature tensor (with the feature-space threat model): $\hat{y} = \text{SO-CERT}(\mathbf{f}, \mathbb{M}_{\text{lrf}}, \mathcal{M}, \mathcal{A}^f_{\mathcal{R}}, y)$. The certification procedure $\text{SO-CERT}(\cdot)$ takes as inputs a feature/image tensor $\mathbf{f}$, the LRF model $\mathbb{M}_{\text{lrf}}$, the secure operation parameters $\mathcal{M}$ used for the model inference, the feature-space threat model $\mathcal{A}^f_{\mathcal{R}}$, and the ground-truth label $y \in \mathcal{Y}$, and outputs a boolean variable $b \in \{\texttt{True}, \texttt{False}\}$ indicating if the certification succeeds, i.e., $\hat{y} = \mathbb{M}_{\text{lrf}}(\mathbf{f}') = y, \forall \mathbf{f}' \in \mathcal{A}^f_{\mathcal{R}}(\mathbf{f})$. We will state and prove the correctness of our certification procedure $\text{PCURE-CERTIFY}(\cdot)$ in Section 3.4.

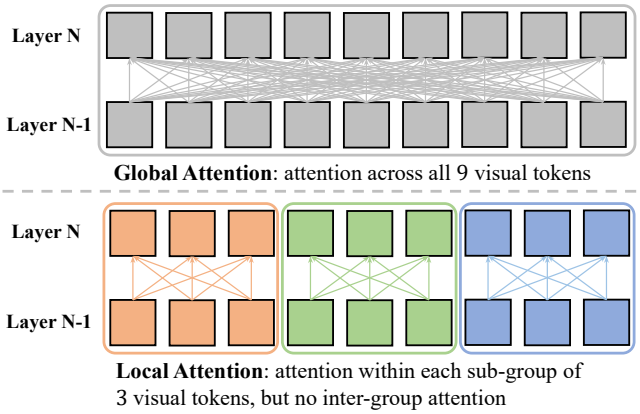

Figure 4: Local attention: each square is a visual token.

3.3 Building SRF Models

In this subsection, we discuss the details of the $\text{TOSRF}(\cdot)$ procedure – how to build SRF models upon off-the-shelf LRF architectures. Since we take computation efficiency as a major performance metric in this paper, we aim to build SRF models with similar computation efficiency as vanilla LRF models. We will discuss approaches for both attention-based (e.g., ViT [13]) and convolution-based (e.g., ResNet [20]) architectures.

ViT-SRF – a ViT [13] variant. To build an attention-based SRF model, we design a ViT variant named ViT-SRF. A vanilla ViT [13] model leverages *global attention* across all visual tokens to fine-tune visual features in each attention layer: the top of Figure 4 illustrates a global attention example where the attention operates across all 9 visual tokens. The global attention makes each ViT feature have a large (global) receptive field. As a result, even if the attacker only corrupts one token, all tokens might be maliciously affected and corrupted. To enforce SRF and avoid complete feature corruption, we propose to use *local attention* over a subset of visual tokens. At the bottom of Figure 4, we provide an example where the attention operation is applied locally to sub-groups of visual tokens (each sub-group has 3 tokens). With this local attention operation, one corrupted token can only affect tokens within the sub-group instead of all tokens.

Computation efficiency. The ViT-SRF model can achieve similar computation efficiency as the vanilla ViT model. First, the number of attention operations can be reduced by our local attention operation. For example, the global attention example in Figure 4 requires $\binom{3\cdot 3}{2} = 36$ attention pairs where the local attention only requires $3 \cdot \binom{3}{2} = 9$. On the other hand, the local attention requires additional operations like reor-

ganizing tensor memory layout and adding/removing `[CLS]` tokens. We will show that our implementation of ViT-SRF has a similar overall inference speed as vanilla ViT in Section 4.

BagNet [3] – a ResNet [20] variant. To build convolution-based SRF models, we leverage an off-the-shelf SRF architecture named BagNet [3]. BagNet is based on the ResNet-50 architecture [20]; it achieves SRF by reducing the kernel sizes and strides of certain convolution and max-pooling layers to ones. BagNet was originally proposed for interpretable machine learning and later adopted as a building block for patch defenses (e.g., PatchGuard [54], BagCert [34]).

Computation efficiency. Small kernel sizes and strides increase the size of the feature map as well as the overall computation costs. As a result, BagNet requires more computation than ResNet-50: we find that BagNet can be 1.5× slower than ResNet in our empirical evaluation (Section 4). Nevertheless, we still categorize BagNet as a computationally efficient SRF model, since other convolution-based SRF models usually incur more than 10× computation overheads.

Notes: other inefficient SRF models. We note that there exist other *inefficient* SRF architectures [7, 25, 26, 44]. For example, a popular SRF strategy is to use an ensemble of vanilla classifiers: each classifier makes a prediction on a cropped small image region; the final prediction is generated via majority voting. This strategy was first proposed in De-randomized Smoothing [25] for ResNet, which incurs a 200+× slowdown. Later, this ensemble idea was adapted for ViT [13] in Smoothed ViT [44], ECViT [7], and ViP [26] with different tricks to improve computation efficiency; however, these SRF models still require more than 10× more computation than vanilla ViT. In contrast, our ViT-SRF has similar computation efficiency as vanilla ViT and BagNet only incurs a 1.5× slowdown (Table 2 in Section 4).

3.4 Robustness Certification

In this subsection, we discuss the robustness certification algorithm (PCURE-CERTIFY$(\cdot)$ in Algorithm 1). We first discuss the correctness of two sub-procedures used in PCURE-CERTIFY$(\cdot)$, i.e., MAP$(\cdot)$ and SO-CERT$(\cdot)$, and then formally state and prove the correctness of our certification algorithm in Theorem 1.

Sub-procedure MAP$(\cdot)$. Recall that MAP$(\cdot)$ uses Equation 2 (from Section 2.3) to calculate the maximum number of corrupted features and convert an image-space threat model $\mathcal{A}_{\mathcal{R}}$ to a feature-space threat model $\mathcal{A}^f_{\mathcal{R}}$. A correctly implemented MAP$(\cdot)$ procedure provides the following proposition.

Proposition 1 (Correctness of MAP$(\cdot)$)**.** *Given a correctly implemented* MAP$(\cdot)$*, an image-space threat model $\mathcal{A}_{\mathcal{R}}$, an SRF sub-model $\mathbb{M}_{srf}$, an input image $\mathbf{x}$, and the converted feature-space threat model $\mathcal{A}^f_{\mathcal{R}} = \text{MAP}(\mathbb{M}_{srf}, \mathcal{A}_{\mathcal{R}})$, we have: for any adversarial image $\mathbf{x}' \in \mathcal{A}_{\mathcal{R}}(\mathbf{x})$, its corresponding adversarial feature map $\mathbf{f}' = \mathbb{M}_{srf}(\mathbf{x}')$ is covered by the feature-space threat model $\mathcal{A}^f_{\mathcal{R}}$. Formally, we have $\forall \mathbf{x}' \in \mathcal{A}_{\mathcal{R}}(\mathbf{x}), \mathbf{f}' = \mathbb{M}_{srf}(\mathbf{x}') : \mathbf{f}' \in \mathcal{A}^f_{\mathcal{R}}(\mathbf{f}), \mathbf{f} = \mathbb{M}_{srf}(\mathbf{x})$.*

Sub-procedure SO-CERT$(\cdot)$. A correctly implemented SO-CERT$(\cdot)$ ensures that the following proposition is true.

Proposition 2 (Correctness of SO-CERT$(\cdot)$)**.** *Given a correctly implemented* SO-CERT$(\cdot)$*, an input tensor $\mathbf{f}$, the model $\mathbb{M}_{lrf}$, the secure operation parameter $\mathcal{M}$, the threat model $\mathcal{A}^f_{\mathcal{R}}$, and the ground-truth label y, if* SO-CERT$(\mathbf{f}, \mathbb{M}_{lrf}, \mathcal{M}, \mathcal{A}^f_{\mathcal{R}}, y)$ *returns* `True`*, we have certifiable robustness. Formally, we have* SO-CERT$(\mathbf{f}, \mathbb{M}_{lrf}, \mathcal{M}, \mathcal{A}^f_{\mathcal{R}}, y) =$ `True` $\implies \forall \mathbf{f}' \in \mathcal{A}^f_{\mathcal{R}}(\mathbf{f}) : \text{SO}(\mathbf{f}', \mathbb{M}_{lrf}, \mathcal{M}) = y$.

Certification procedure PCURE-CERTIFY$(\cdot)$. With the two propositions discussed above, we can state and prove the correctness of the certification procedure below.

Theorem 1. *Given a clean image $\mathbf{x}$, its ground-truth label y, the* PATCHCURE *defense setting $\mathbb{M}_{srf}, \mathbb{M}_{lrf}, \mathcal{M}$, the image-space patch threat model $\mathcal{A}_{\mathcal{R}}$, and correctly implemented* MAP$(\cdot)$ *and* SO-CERT$(\cdot)$ *procedures (Propositions 1 and 2), if the certification procedure* PCURE-CERTIFY$(\cdot)$ *returns* `True`*, we have certifiable robustness for this clean image. Formally, we have* PCURE-CERTIFY$(\mathbf{x}, y, \mathbb{M}_{srf}, \mathbb{M}_{lrf}, \mathcal{M}, \mathcal{A}_{\mathcal{R}}) =$ `True` $\implies \forall \mathbf{x}' \in \mathcal{A}_{\mathcal{R}}(\mathbf{x}) : \text{PCURE-INFER}(\mathbf{x}', \mathbb{M}_{srf}, \mathbb{M}_{lrf}, \mathcal{M}) = y$

Proof. The correctness of two sub-procedures MAP$(\cdot)$ and SO-CERT$(\cdot)$ gives us two useful propositions. Proposition 1 demonstrates that $\forall \mathbf{x}' \in \mathcal{A}_{\mathcal{R}}(\mathbf{x}), \mathbf{f}' = \mathbb{M}_{\text{srf}}(\mathbf{x}')$, we have $\mathbf{f}' \in \mathcal{A}^f_{\mathcal{R}}(\mathbf{f})$; Proposition 2 demonstrates that $\forall \mathbf{f}' \in \mathcal{A}^f_{\mathcal{R}}(\mathbf{f})$, we have $\text{SO}(\mathbf{f}', \cdot) = y$, as long as SO-CERT$(\cdot)$ returns `True`. Combining two propositions together, we can derive that, $\forall \mathbf{x}' \in \mathcal{A}_{\mathcal{R}}(\mathbf{x}), \mathbf{f}' = \mathbb{M}_{\text{srf}}(\mathbf{x}')$ we have $\text{SO}(\mathbf{f}', \cdot) = y$, as long as SO-CERT$(\cdot)$ returns `True`.

According to Lines 12-17 of Algorithm 1, PCURE-CERTIFY$(\cdot)$ returns `True` iff SO-CERT$(\cdot)$ returns `True`. Moreover, Lines 7-11 show that PATCHCURE-INFER$(\mathbf{x}', \cdot) =$ SO$(\mathbf{f}', \cdot)$. Therefore, we have proved that $\forall \mathbf{x}' \in \mathcal{A}_{\mathcal{R}}(\mathbf{x})$, PCURE-INFER$(\mathbf{x}', \cdot) = y$, as long as PCURE-CERTIFY$(\cdot)$ returns `True`. □

Remark 1: certifiable robustness evaluation. In our evaluation (Section 4), we will apply the PCURE-CERTIFY$(\cdot)$ procedure to labeled datasets and report *certified robust accuracy* as the fraction of test images for which PCURE-CERTIFY$(\cdot)$ returns `True`. This certified robust accuracy is our robustness evaluation metric.

Remark 2: adaptive attacks vs. certifiable robustness. Theorem 1 ensures that the certified robust accuracy discussed above covers all possible attackers within a given threat model, i.e., $\forall \mathbf{x}' \in \mathcal{A}_{\mathcal{R}}(\mathbf{x})$. The threat model $\mathcal{A}_{\mathcal{R}}$ can capture an adaptive attacker who has full knowledge of the defense and uses a patch of a certain shape and size at all possible locations

and with all possible patch content. Therefore, we can view certified robust accuracy as a lower bound of model accuracy against any (adaptive) attack within the threat model $\mathcal{A}_{\mathcal{R}}$. For example, if we consider an attacker who can use a 2%-pixel square patch with any patch content at any image location, a certified robust accuracy of 61.6% means that no adaptive attacker using the same patch shape and size can reduce the model accuracy below 61.6% (guaranteed by Theorem 1). With this theoretical guarantee, our evaluation focuses on certified robustness instead of empirical robustness against *concrete* adaptive attack algorithms.

Remark 3: certification and inference procedures. We note that our certification procedure PCURE-CERTIFY$(\cdot)$ requires ground-truth labels to check the correctness of the model prediction for robustness evaluation. In contrast, our inference procedure PCURE-INFER$(\cdot)$ does not require a ground-truth label and thus can be deployed in the wild. The certifiable robustness evaluated on a labeled dataset (using the certification procedure) provides a robustness estimation for the model (inference procedure) deployed in the wild.

3.5 PATCHCURE Instantiation

In this subsection, we discuss how to instantiate PATCHCURE with different parameter settings to approach the three-way trade-off problem.

PATCHCURE parameters. There are four major parameters for the PATCHCURE defense. The *first* is the splitting layer index k used in model splitting SPLIT$(\cdot)$. As we choose a larger k, we will split the model at a deeper layer, which could improve inference efficiency but could also make the LRF sub-model too shallow to achieve high utility for the end-to-end defense model. The *second* parameter is the base model architecture $\mathbb{M}_{\mathfrak{b}}$ used to construct SRF and LRF models. Different architectures usually have different properties. For example, ViT is shown more robust to occlusion [19], which can be viewed as a non-adversarial pixel patch. The *third* parameter is the receptive field size **rf**, larger receptive fields can improve model utility but might hurt the certifiable robustness [54]. The fourth parameter is the parameter(s) of the secure operation layer; it controls the properties of the secure operation algorithm.

Next, we focus on the model splitting parameter k, which is the unique parameter introduced by our PATCHCURE framework, and discuss three different choices of k for PATCHCURE instantiation. Recall that $k \in \{0, 1, \cdots, L\}$, where L is the number of all model backbone layers (excluding feature aggregation and classification layers), specifies the layer where we split the model and insert the secure operation.

Case 1: Optimizing for computation efficiency. ($k = L$). First, we aim to build efficient defense instances by setting k to its largest value L. In this case, we convert the entire model backbone into an SRF model and instantiate the LRF model as a combination of global average pooling plus a linear classification head. The computation overhead of the LRF model is much smaller than the SRF model (e.g., 500× difference). Even dozens of LRF calls generated from the secure operation layer will not have a significant impact on the model inference efficiency.

Remark. In this case, we can consider PATCHCURE reduced to a pure SRF defense like PatchGuard [54]. Nevertheless, we note that PATCHCURE's flexibility of combining different modules naturally leads to the new idea of applying an LRF-based secure operation, e.g., double-masking [55], to an SRF-based feature map. In Figure 2, we have seen the benefits of this new combination: we build efficient PATCHCURE instances that outperform all but one existing (inefficient) defense in terms of robustness.

Case 2: Optimizing for model utility and robustness ($k = 0$). In the second case, we set $k = 0$ and reduce PATCHCURE to a pure LRF defense like PatchCleanser [55]. PATCHCURE with $k = 0$ can match the utility and robustness of any existing defense, at the cost of relatively large computation overheads.

Case 3: Interpolation between efficient defenses with high-robustness defenses ($0 < k < L$). Our final case study aims to leverage PATCHCURE to systematically adjust model performance in terms of robustness, utility, and efficiency. Recall that we can build state-of-the-art efficient defenses using $k = L$ and state-of-the-art defenses (without efficiency considerations) using $k = 0$. Now we want to further build defenses whose robustness-utility performance can fill the gap between efficient PATCHCURE (top stars in Figure 2) and the state-of-the-art PatchCleanser [55] (the pentagon in Figure 2; a special PATCHCURE instance with $k = 0$). We note that this task used to be hard: existing LRF defenses like PatchCleanser [55] do not have a design point that has similar inference efficiency as undefended models; existing efficient SRF defenses like PatchGuard [54] cannot sacrifice part of their efficiency in trade for better utility and robustness. However, with our PATCHCURE design, we can easily reach different performance points by varying k between 0 and L, as demonstrated in Figure 2 in Section 1 and more analyses in Section 4. This flexibility of tuning defense performance is useful in practice when we want to find the optimal defense instances given utility and efficiency constraints.

4 Evaluation

In this section, we present our evaluation results. We will demonstrate that PATCHCURE (1) guides us to build efficient defenses that outperform all but one prior (mostly inefficient) defense, and (2) allows us to flexibly adjust the robustness, utility, and efficiency of defense models.

4.1 Setup

Dataset. We focus on the ImageNet-1k dataset [12]. It contains 1.3M training images and 50k validation images from

1000 classes. ImageNet-1k is a challenging dataset that witnesses an intense three-way trade-off between robustness, utility, and efficiency [59]. It was also widely used for evaluation in prior works [7, 25, 26, 34, 44, 54, 67] and thus enables an easy and fair comparison between different defenses. We consider an image resolution of 224×224 for a fair inference efficiency comparison. We also include additional evaluation results for CIFAR-10 [23] in Section 4.3, and CIFAR-100 [23] and SVHN [38] in our technical report [61].

Model architectures. We consider ViT-B [13] and ResNet-50 [20] as our non-robust base models, and build ViT-SRF and BagNet as their SRF variants.

ViT-B and ViT-SRF. ViT-B [13] has 12 attention layers; we split between different attention layers, i.e., we select $k \in \{0, 1, \cdots, 12\}$. Moreover, we consider the ViT-B architecture with an input image size of 224×224: each visual token accounts for 16×16 image pixels, and there are 14×14 visual tokens in total. We consider local attention with a sub-group of 2×2, 14×1, and 14×2 visual tokens, which correspond to a receptive field size of 32×32, 224×16, 224×32 pixels, respectively. We name a ViT-SRF instance that uses local attention over $m \times n$ visual tokens as ViT{m}x{n}. We name a PATCHCURE instance that splits at k^{th} layer and instantiates SRF model with ViT{m}x{n} as PCURE-ViT{m}x{n}-k{k}. For example, PCURE-ViT14x1-k9 stands for a PATCHCURE instance that splits the vanilla ViT-B at the 9$^{\text{th}}$ attention layer and uses ViT-SRF with 14x1 local attention.

ResNet-50 and BagNet. ResNet-50 [20] has 50 layers in total, and we consider $k \in \{0, 1, \cdots, 50\}$. Moreover, we consider BagNet with different receptive field sizes of 17×17, 33×33, and 45×45 pixels. We name a PATCHCURE instance that uses BagNet with a receptive field size of $m \times m$ and splits at the k^{th} layer as PCURE-BagNet{m}-k{k}. For example, PCURE-BagNet33-k50 stands for a PATCHCURE instance that splits the vanilla ResNet-50 at the 50$^{\text{th}}$ layer and uses BagNet33 as the SRF sub-model.

Model training. We note that PATCHCURE models without the secure operation layer are standard feedforward networks. Therefore, we can leverage off-the-shelf training techniques and recipes to train our PATCHCURE models. For ViT-based models (ViT or ViT-SRF), we take the pretrained weight from MAE [19] and follow the recipe of MAE finetuning to tune a model for the classification task on ImageNet-1k. For ResNet-based models (ResNet or BagNet), we follow Schedule B of the ResNet-strikes-back paper [50] and train the model from scratch. We note that both ViT-based and ResNet-based models only use ImageNet-1k without additional data for fair comparison. During the training, we add random masks to the feature map of the k^{th} model backbone layer; this mimics the masking operation used in our LRF secure operation layer (e.g., double-masking [55]). After we train our PATCHCURE models, we add the secure operation layer back for robust inference.

Attack threat model. Our main evaluation results focus on *one* square patch that takes up to 2% of the image pixels and can be placed anywhere on the image, i.e., a 32×32 patch anywhere on the 224×224 image. This is a popular evaluation setting used in most existing works [7,25,26,34,44,54,67]. We will additionally analyze model performance for large patches (up to 70% of the image pixels) in Section 4.3. As discussed in Section 3.4 (Remark 2), our robustness certification procedure has accounted for *any* adaptive attack strategy within the constraint $\mathcal{A}_{\mathcal{R}}$ defined in Section 2.1. Therefore, we only need to specify the threat model $\mathcal{A}_{\mathcal{R}}$ (but not concrete attack algorithms) for robustness evaluation.

Table 2: Performance of undefended models

Models	Accuracy (%) clean	robust	Throughput (img/s)
ViT-B [13, 19]	83.7	0	191.7
ResNet-50 [20]	80.1	0	295.5
BagNet-33 [3]	73.0	0	192.2
ViT14x2	79.4	0	195.2
ViT14x1	77.4	0	190.4
ViT2x2	72.7	0	166.7

Evaluation metrics. We focus on the three-way trade-off problem between certifiable robustness, model utility, and computation efficiency; therefore, we have three major evaluation metrics. For certifiable robustness, we report *certified robust accuracy*, defined as the fraction of test images that the certification procedure (PCURE-CERTIFY$(\cdot)$ of Algorithm 1) returns `True`. This is a lower bound of model accuracy against any adaptive attack within the given threat model (recall Remark 2 in Section 3.4). For model utility, we report *clean accuracy*, which is the standard model accuracy on clean test images without adversarial patches. For computation efficiency, we report empirical *inference throughput* on clean images, which is defined as the number of images a model can make predictions within every second (including data loading and model feedforward). We use a batch size of 4 for the throughput evaluation – we find large batch sizes do not significantly affect throughput in our experiment setting (Appendix A). We also report latency, estimated FLOPs, and memory footprint of different defense models in Appendix A. We conduct our experiments using PyTorch [40] on one NVIDIA RTX A4000 GPU. We also discuss the implication of different hardware settings and application scenarios in Section 5.

Comparison with prior defenses. We compare our results with all prior certifiably robust defenses that can recover correct predictions (without abstention) and are scalable to the ImageNet-1k dataset: Clipped BagNet (CBN) [67], De-Randomized Smoothing (DRS) [25], PatchGuard [54], BagCert [34], PatchCleanser [55], Smoothed ViT (S-ViT) [44], ECViT [7], and ViP [26]. We note that S-ViT,

Table 3: Performance of different defenses against one 2%-pixel patch anywhere on the ImageNet images (parentheses contain the relative performance compared to state-of-the-art PatchCleanser [55])

Defenses†	Accuracy (%) clean	Accuracy (%) robust	Throughput (img/s)
PCURE-ViT14x2-k12	78.3 (0.95)	44.2 (0.72)	**189.9 (90.4)**
PCURE-ViT14x1-k12	76.3 (0.92)	47.1 (0.77)	**182.0 (86.7)**
PCURE-ViT2x2-k12	71.3 (0.86)	46.8 (0.77)	**158.1 (75.3)**
PCURE-BagNet17-k50	65.4 (0.79)	42.2 (0.69)	**115.1 (54.8)**
PCURE-BagNet33-k50	70.8 (0.85)	44.1 (0.72)	**136.8 (65.1)**
PCURE-BagNet45-k50	72.4 (0.88)	34.8 (0.57)	**132.5 (63.1)**
PCURE-ViT14x2-k11	78.3 (0.95)	45.8 (0.75)	109.0 (51.9)
PCURE-ViT14x2-k10	79.8 (0.97)	46.0 (0.75)	77.2 (36.8)
PCURE-ViT14x2-k9	80.0 (0.97)	46.2 (0.75)	58.4 (27.8)
PCURE-ViT14x2-k6	81.8 (0.99)	47.4 (0.78)	34.4 (16.4)
PCURE-ViT14x2-k3	**82.1 (0.99)**	47.8 (0.78)	23.9 (11.4)
PCURE-ViT14x2-k0	**82.2 (1.00)**	47.8 (0.78)	19.7 (9.4)
PCURE-ViT14x1-k0	**82.5 (1.00)**	53.7 (0.88)	8.3 (4.0)
PCURE-ViT2x2-k0	**82.6 (1.00)**	**61.6 (1.00)**	2.0 (1.0)
PatchCleanser [55] (robust)§	**82.5 (1.00)**	**61.1 (1.00)**	2.1 (1.0)
PatchCleanser [55] (efficient)	82.0 (0.99)	55.1 (0.90)	12.5 (5.9)
ECViT [7]	78.6 (0.95)	41.7 (68.2)	2.25 (1.1)
ViP [26] (robust)	75.8 (0.92)	40.4 (66.1)	0.8 (0.4)
ViP [26] (efficient)	75.3 (0.91)	38.3 (0.63)	7.7 (3.7)
S-ViT [44] (robust)	73.2 (0.89)	38.2 (0.63)	0.8 (0.4)
S-ViT [44] (efficient)	67.3 (0.82)	33.0 (0.54)	20.5 (9.7)
PatchGuard [54]	54.6 (0.66)	26.0 (0.43)	**177.4 (84.5)**
BagCert [34]	45.3 (0.55)	22.7 (0.37)	**181.4 (86.4)**
DRS [25]	44.4 (0.54)	14.0 (0.23)	1.5 (0.7)
CBN [67]	49.5 (0.60)	7.1 (0.11)	**181.5 (86.4)**

† For each prior defense without multiple instances, we report its most robust and most efficient instances, denoted with "(robust)" and "(efficient)".
§ The PatchCleanser numbers in its original paper are slightly higher because they use additional ImageNet-21k [12] data for pretraining.

ECViT, and ViP propose similar algorithms by applying DRS to ViT, we sometimes group them into "DRS+ViT" to simplify the figure legend.

4.2 Main results

In Table 2 and Table 3, we report the main evaluation results for undefended models and different defenses on the ImageNet-1k dataset against one 2%-pixel square patch anywhere on the image. We note that many defenses have multiple defense instances; we report the performance of their most *robust* and most *efficient* instances if efficiency greatly affects their robustness performance. In Table 3, we additionally report the relative performance compared with the most robust prior defense, i.e., PatchCleanser [55].

PatchCURE builds defense instances that have similar efficiency as undefended models. First, we analyze efficient defense instances. As discussed in Section 3.5, we will set $k = L$. We report performance for PCURE-ViT-14x2-k12, PCURE-ViT-14x1-k12, PCURE-ViT-2x2-k12, PCURE-BagNet33-k50, PCURE-BagNet17-k50, and PCURE-BagNet45-k50 in Table 3. For ViT-based defenses, we can see that the inference throughput is similar to that of vanilla ViT-B (Table 2). For ResNet/BagNet-based defenses, we find that our defenses are 2× slower than ResNet-50. Nevertheless, our BagNet-based defenses are still faster than most prior works, and we still categorize them as efficient defenses. In addition to the high inference efficiency, PatchCURE also achieves decent model utility and robustness. For example, PCURE-ViT14x2-12 has a 78.3% clean accuracy for the challenging 1000-class ImageNet classification task and 44.2% certified robust accuracy against one 2%-pixel square patch anywhere on the image.

Table 4: Additional results for PatchCleanser [55] and PatchGuard [54] with different backbones

Defenses	Accuracy (%) clean	Accuracy (%) robust	Throughput (img/s)
PatchCleanser-ViT-B	82.5 (1.00)	61.1 (1.00)	2.1 (1.0)
PatchGuard-ViT14x2	76.9 (0.93)	23.1 (0.38)	193.4 (92.1)
PatchGuard-ViT14x1	75.2 (0.91)	29.9 (0.49)	188.7 (89.8)
PatchGuard-ViT2x2	69.6 (0.84)	33.0 (0.54)	164.7 (78.4)
PatchCleanser-ResNet50	78.3 (0.95)	53.1 (0.87)	10.6 (5.8)
PatchGuard-BagNet17	60.4 (0.73)	27.6 (0.45)	181.2 (86.3)
PatchGuard-BagNet33	67.2 (0.81)	24.0 (0.39)	183.4 (87.3)
PatchGuard-BagNet45	67.9 (0.82)	16.0 (0.26)	175.8 (83.7)

PatchCURE provides a systematic way to balance the three-way trade-off between robustness, utility, and efficiency. Second, we aim to demonstrate PatchCURE's flexibility to balance defense performance by adjusting the parameters k. In Table 3 and Figure 5, we report the performance of PCURE-ViT14x2 instantiated with different k. As we split the model at a deeper layer (larger k), the inference throughput greatly improves while the clean accuracy and the certified robust accuracy gradually decrease. This is because a deeper splitting point leads to a shallower LRF sub-model and thus reduces the cost of the secure operation layer (which calls the LRF sub-model multiple times). Meanwhile, a deeper splitting point slightly decreases the model capacity and leads to drops in clean accuracy and certified robustness accuracy.

PatchCURE builds efficient defenses that outperform all but one (mostly inefficient) prior defenses and can achieve the best robustness/utility by sacrificing part of the computation efficiency. Now, we compare PatchCURE with prior defenses. In addition to Table 3, we visualize the performance of different PatchCURE instances and prior defenses in Figure 6.[3] First, we can see that efficient PatchCURE instances (e.g., the first three rows in Table 3 and the red stars in Figure 6) achieve state-of-the-art utility and robustness performance among defenses with small defense overheads. For example, compared to the best efficient

[3] There are a large number of different PatchCURE instances. Table 3 and Figure 6 only report two partially overlapped subsets of PatchCURE instances for simplicity and better visual appearance.

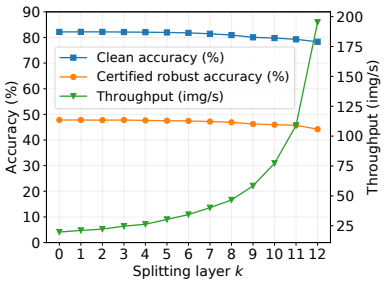

Figure 5: Effect of the splitting layer k on ViT14x2-based PATCHCURE

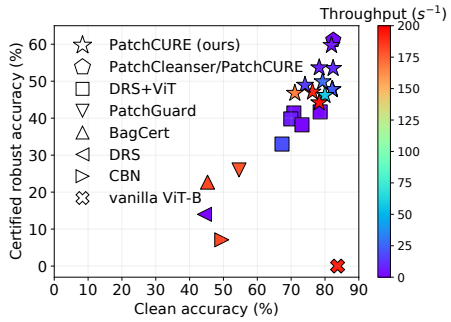

Figure 6: Comparison between PATCHCURE and prior defenses

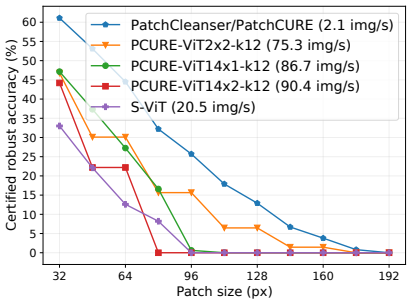

Figure 7: PATCHCURE robustness against larger patches

defense PatchGuard [54], our ViT-based defenses have 16.7-23.7% absolute improvements in clean accuracy and 18.2-21.1% absolute improvement in certified robust accuracy; our BagNet-based defenses have 10.8-17.8% absolute clean accuracy improvements and 8.8-18.1% absolute certified robust accuracy improvements. Moreover, we note that the robustness and utility performance of our efficient defenses is also comparable to, or even surpasses, many inefficient defenses. For example, our ViT-based defenses (red stars in Figure 6) outperform defenses from the DRS-ViT family [7, 26, 44] (purple squares in Figure 6).

The only existing defense that significantly outperforms our efficient instances is PatchCleanser [55]. However, we note that PatchCleanser is simply a design point of PATCHCURE with $k = 0$; PATCHCURE can easily reach the performance of PatchCleanser by sacrificing part of its computation efficiency. Figure 6 demonstrates that we can build a family of PATCHCURE instances (with different k and SRF submodels) that fill in the gap between PatchCleanser (purple pentagon) and efficient PATCHCURE instances (red stars). Moreover, we note that the PATCHCURE instances also achieve the best robustness and utility performance across all different efficiency levels (denoted by different colors in Figure 6). This PATCHCURE defense family allows us to easily find the optimal defenses that satisfy certain utility or computation constraints in practice.

PATCHCURE works well across different model backbones compared to PatchCleanser [55] and PatchGuard [54]. In Table 4, we report additionally results for PatchCleanser [55] and PatchGuard [54] with the same SRF/LRF backbones used by PATCHCURE for a targeted comparison. For ViT-based backbones, PATCHCURE instances like PCURE-ViT14x2-k12 (in Table 3) achieve significant speedup compared to PatchCleaner-ViT (in Table 4). Efficient PATCHCURE instances PCURE-k12 in Table 3 have similarly high inference throughput compared with PatchGuard in Table 4, but achieve a much high certified robust accuracy. We can also see similar observations for ResNet/BagNet backbones. These comparisons demonstrate that PATCHCURE works well across different backbone models with different receptive field sizes.

4.3 Detailed Analyses

In this section, we provide additional analyses of our PATCHCURE defenses. We will discuss the properties of SRF models used in PATCHCURE, PATCHCURE performance when facing larger patches, PATCHCURE performance on additional datasets, and the effects of different PATCHCURE parameters.

Our undefended SRF models have similar inference efficiency as off-the-shelf models with a small utility drop. In Section 3.3, we discussed how to build SRF models using off-the-shelf CNN and ViT. In Table 2, we report the performance of vanilla undefended SRF models. First, we can see that ViT-based models have similarly high computation efficiency as vanilla ViT models. For PCURE-ViT14x2, the inference throughput is even slightly higher than vanilla ViT. Moreover, we note that as we use a smaller receptive field (from 14x2 to 14x1 to 2x2 visual tokens), the clean accuracy of ViT-SRF decreases as expected; meanwhile, the inference throughput is also slightly affected because a smaller receptive field leads to more sub-groups of visual tokens, and each sub-group incurs additional computation costs (conversion between to the sub-group-style token layout and the original-style visual token layout). Second, for BagNet – the ResNet-based SRF model – we find that both clean accuracy and inference throughput are moderately affected. This is because smaller convolution kernels and strides result in a larger feature map and thus more computation. This is also why we find that the ViT-based PATCHCURE achieves better performance than the BagNet-based PATCHCURE in Section 4.2. For the rest of this subsection, we will focus on analyzing ViT-based defenses.

Different PATCHCURE instances exhibit different robustness against larger patches. In Section 4.2, we report certified robust accuracy for one 2%-pixel square patch anywhere on the ImageNet image, because this is a popular evalu-

Table 5: Performance of ViT-based defenses on CIFAR-10 (one 2%-pixel patch on the resized 224×224 images)

Dataset	Accuracy (%) clean	Accuracy (%) robust	Throughput (img/s)
PCURE-ViT14x2-k12	95.3	67.6	190.0
PCURE-ViT14x2-k6	96.8	70.3	44.4
PCURE-ViT2x2-k3	97.8	80.5	5.0
ViT-B [13]	98.1	0	188.3
PatchCleanser [55]/PCURE	98.0	86.5	3.8
S-ViT [44]	90.8	67.6	0.7
ECViT [7]	93.5	76.4	2.3

Table 6: PATCHCURE for *attack detection* on ImageNet against one 2%-pixel square patch

Models	Accuracy (%) clean	Accuracy (%) robust	Throughput (img/s)
PCURE-ViT14x2-k12	64.7	64.7	196.8
PCURE-ViT14x1-k12	63.5	63.5	192.3
PCURE-ViT2x2-k12	59.5	59.5	166.5
ViT + MR [32, 55]	74.3	74.3	2.5
ViP [26]	74.6	74.6	2.5
ScaleCert [17]	58.5	55.4	NA†
PatchGuard++ [57]	49.8	49.8	151.1

† We did not evaluate ScaleCert [17] due to the lack of open-source implementation.

ation setup for benchmark comparison [7,25,26,34,44,54,67]. In Figure 7, we report the certified robust accuracy of different PATCHCURE instances for different patch sizes (up to 192×192 on the 224×224 image). As shown in Figure 7, the certified robust accuracy decreases as the patch size increases. Notably, different defense instances have different sensitivity to larger patches. The most efficient defense, PCURE-ViT14x2-k12, is most sensitive to larger patches due to the relatively large receptive field size of its SRF model (ViT14x2 has a receptive field of 224×32). In contrast, ViT2x2 has a smaller receptive field of 32×32, and its PATCHCURE defense is more robust to larger patches. Moreover, we plot the results for PatchCleanser [55] and S-ViT [44], the two best-performing open-source defenses. First, we note that PatchCleanser can be viewed as a special PATCHCURE instance that uses an identical mapping layer (with a receptive field size of 1×1) as the SRF sub-model. It has the best robustness to larger patches but smaller throughput. Second, we can also see that our efficient PATCHCURE instances beat the robustness of S-ViT in almost all cases, at a much larger inference throughput.

PATCHCURE performs well on other datasets like CIFAR-10 [23]. In Section 4.2, we discuss PATCHCURE's promising performance on the challenging ImageNet dataset. In this analysis, we demonstrate that our PATCHCURE also works well for other benchmark datasets. In Table 5, we report the performance of different ViT-based models on the CIFAR-10 [23] dataset. As shown in the table, PATCHCURE has the following advantages compared to other defenses: efficient instances like PCURE-ViT14x2-k12 have significantly higher throughput and comparable robustness and utility; robust instances like PCURE-ViT-2x2-k3 match the robustness and utility performance of state-of-the-art defenses. In our technical report [61], we include more datasets like CIFAR-100 [23] and SVHN [38].

PATCHCURE parameter selection. As discussed in Section 3.5, PATCHCURE has four parameters: k (splitting layer), model architecture, SRF size, and the secure operation parameter. We need to understand their effect to properly select parameters in practice. The effect of the first parameter k has been already discussed in Figure 5 and Section 4.2: it is the most important knob in PATCHCURE to balance the three-way trade-off. In practice, we can set k to the largest value that meets the computation efficiency requirement. The effects of the model architecture and receptive field size have been implicitly discussed in our earlier experiments. For model architecture, Table 2 demonstrates that ViT has an advantage over ResNet in their SRF model's utility and efficiency; this advantage is also reflected in the defense performance of ViT and ResNet-based defenses (Table 3). We attribute this difference to the recent findings that ViT is more robust to input masking [19]. The effect of different receptive field sizes is also demonstrated in Table 3 and Figure 7. Table 3 shows that ViT-SRF with a larger receptive field has higher clean accuracy and slightly better inference efficiency. However, larger receptive fields make the defenses more vulnerable to larger patches (Figure 7). Therefore, we need to carefully choose the defense parameters based on the application scenarios. In practice, we can choose the smallest receptive field size that meets the model utility requirement to get optimal robustness. The effect of the secure operation parameter depends on the chosen secure operation algorithm; we provide a quantitative analysis in our technical report [61].

5 Discussion

In this section, we discuss PATCHCURE's compatibility with different secure operation algorithms, different model sizes, application scenarios, limitations, and future work directions.

Compatibility with different secure operation algorithms. We propose PATCHCURE as a general defense framework with three modules: the SRF model, the LRF model, and the secure operation. In Section 4, we extensively analyze different design choices of SRF and LRF models. Nevertheless, we keep secure operation the same as double-masking [55] because it is the state-of-the-art secure operation algorithm

Table 7: PATCHCURE with ViT-Base and ViT-Large

Defense	Clean	Certified	Throughput	Memory (MB)
ViT-B-14x2-k12	78.3	44.2	189.9	362.0
ViT-B-14x2-k9	80.0	46.2	58.4	515.6
ViT-L-14x2-k24	80.3	47.7	58.0	1203.2

for prediction recovery. In this discussion, we aim to showcase PATCHCURE's compatibility with a different secure operation algorithm, namely the Minority Reports (MR) defense [32]. MR is an LRF defense that aims to detect an ongoing patch attack: given an input image, MR either returns a robust prediction label or issues an attack alert.[4] In Table 6, we report the performance of PATCHCURE with MR-style secure operation and compare it with other attack-detection-based defenses. The table demonstrates that our efficient PATCHCURE instances achieve decent robustness and utility performance with high inference speed.

PATCHCURE with different model sizes. In Section 4, we demonstrate that PATCHCURE can flexibly tune the model performance without significantly altering the model architecture and size. We note that PATCHCURE technique is also compatible with (and orthogonal to) strategies like using models with different sizes. In Table 7, we report an example using PATCHCURE with ViT-Base and ViT-Large. We can see that PCURE-ViT-B-14x2-k9 can match the robustness, utility, and throughput of PCURE-ViT-L-14x2-k24 (ViT-L has 24 attention layers in total), but with smaller memory consumption. This also demonstrates the benefits of introducing PATCHCURE's knobs. In practice, developers can combine PATCHCURE with orthogonal techniques like model compression and pruning.

PATCHCURE application scenarios. In this paper, we focus on the general problem of adversarial patch attacks/defenses, which can be applied to many different applications like autonomous driving [14], face authentication [49], surveillance [62], and cashierless self-checkout [21], which have different hardware settings and also different levels of sensitivity to latency. Our evaluation setting in Section 4 is more suited for cloud deployments since we use A4000, a common workstation GPU, for experiments. We demonstrate that PATCHCURE can have a similar inference speed as vanilla models under the same hardware setting (191.7 img/s for ViT-B vs. 189.9 img/s for PCURE-ViT14x2-k12). This implies that, in time-sensitive applications, PATCHCURE can meet the throughput/latency requirement as the vanilla model does. An interesting direction of future works is to concretely implement PATCHCURE with other model compression techniques for embedded edge devices.

Limitations and future directions. The certifiable robustness of PATCHCURE relies on its secure operation layer. As a result, PATCHCURE instances will inherit limitations from the secure operation algorithm. For example, if we use the Minority Reports algorithm [32], we can only achieve robustness for attack detection. If we use the double-masking algorithm [55], our defense requires additional patch information such as an estimation of patch size and shape. How to relax these assumptions is an important future work direction. Nevertheless, we note that PATCHCURE provides a systematic way to build defense instances with different properties. The compatibility of the framework implies its potential for future improvements: given any progress in the design of SRF models, LRF models, or secure operations, we can leverage these advancements to build stronger PATCHCURE defense instances. Moreover, another interesting direction is to incorporate PATCHCURE, a test-time defense, with training-time certified defenses [16, 35] to further improve robustness.

[4] We note that this attack-detection-based defense has a weaker notion than prediction-recovery-based defense like PATCHCURE and PatchCleanser [55].

6 Related Work

Adversarial Patch Attacks. Brown et al. [4] introduced the first adversarial patch attack; they demonstrated that an attacker can use a physically printed adversarial patch to induce targeted misclassification. The physical realizability of patch attacks has drawn great attention from the machine learning security community. In a concurrent work, Karmon et al. [22] explored a similar concept called the Localized and Visible Adversarial Noise (LaVAN) attack, which focuses on the digital-domain patch attack. Many more patch attacks have been proposed for different attack scenarios [6, 28, 29, 45, 46, 53, 62, 65]. In this paper, we focus on attacks against image classification models.

Adversarial Patch Defenses. To defend against adversarial patch attacks, both empirically robust and certifiably robust defenses have been proposed. Empirical defenses [9, 18, 27, 36, 37, 41, 42, 48, 52, 63] are usually based on heuristics and lack formal robustness guarantees; in contrast, certifiably robust defenses [7, 8, 25, 26, 32, 34, 44, 54, 55, 67] can claim robustness in a provable manner. Chiang et al. [8] proposed the first certifiably robust defense for patch attacks via Interval Bound Propagation (IBP) [16, 35]. This defense is too computationally intense to scale to the ImageNet dataset, so we did not include it in Section 4 for comparison. Zhang et al. [67] proposed Clipped BagNet (CBN) that clips BagNet [3] features for certifiable robustness. Levine et al. [25] proposed De-Randomized Smoothing (DRS) that performs majority voting on model predictions on different small cropped images. We note that DRS was further improved by Smoothed ViT [44], ECViT [7], and ViP [26] using the Vision Transformer (ViT) [13] architecture. Xiang et al. [54] proposed PatchGuard shortly after DRS as a general defense framework that uses a model with small receptive fields (SRF) for feature extraction and performs secure feature aggregation for robust predictions. The idea of SRF is widely used in

many defenses [7, 17, 25, 26, 34, 44]. Our PATCHCURE also leverages the SRF idea to improve computation efficiency. After PatchGuard, Metzen et al. [34] proposed BagCert, in which they modified vanilla BagNet [3] for robust predictions. Xiang et al. [55] later proposed PatchCleanser with a double-masking algorithm that reliably removes the patch from the input image. In contrast to other works, PatchCleanser does not rely on SRF models and achieves state-of-the-art certifiable robustness while maintaining high model utility (e.g., 1% drops from undefended models). In a concurrent work of this paper, PatchCleanser is further improved with a new model training technique [43]. However, we note that PatchCleanser requires expensive model predictions on multiple masked images and incurs a large computation overhead. In this paper, we propose PATCHCURE to approach the three-way trade-off between robustness, utility, and efficiency. Moreover, we note that our general PATCHCURE design subsumes PatchCleanser by setting $k = 0$.

In addition to the defenses discussed above, there are other certifiably robust defenses for attack detection [17, 32, 57]. These defenses alert and abstain from making predictions when they detect an ongoing attack, which achieves a weaker robustness property compared to defenses that can recover correct predictions without any abstention. In Section 5, we demonstrated PATCHCURE's compatibility with one of the LRF-based attack-detection algorithms, Minority Reports [32]. Finally, there is another line of certifiably robust defenses that study harder vision tasks such as object detection [56, 60] and semantic segmentation [66]. We refer the readers to a survey paper [59] for more discussions on certified patch defenses.

Other Adversarial Example Attacks and Defenses. There are many existing paper studies adversarial example attacks defenses for different threat models [2, 5, 10, 15, 31, 33, 39, 47, 51, 58, 64]. We focus on the adversarial patch attacks because they can be realized in the physical world and impose a threat to the cyber-physical world.

7 Conclusion

In this paper, we proposed PATCHCURE as a general defense framework to build certifiably robust defenses against adversarial patch attacks. PATCHCURE is the first to explicitly approach the three-way trade-off problem between certifiable robustness, model utility, and computation efficiency. We demonstrated that PATCHCURE enables us to build state-of-the-art efficient defense instances, and provides sufficient knobs to adjust the defense performance across the three dimensions. We note that PATCHCURE is a general defense framework compatible with any SRF and LRF model architectures as well as secure operation algorithms – advancements in SRF/LRF model architectures and secure operations will also lead to stronger PATCHCURE defense instances.

Acknowledgements

We are grateful to the anonymous shepherd and reviewers at USENIX Security 2024 for their valuable feedback. This work was supported in part by the National Science Foundation under grant CNS-2131938 and the Princeton SEAS Innovation Grant.

References

[1] Andre Araujo, Wade Norris, and Jack Sim. Computing receptive fields of convolutional neural networks. *Distill*, 2019.

[2] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *ECML PKDD*. Springer, 2013.

[3] Wieland Brendel and Matthias Bethge. Approximating CNNs with bag-of-local-features models works surprisingly well on ImageNet. In *ICLR*, 2019.

[4] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. In *NeurIPS Workshops*, 2017.

[5] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *S&P*, 2017.

[6] Zhaoyu Chen, Bo Li, Shuang Wu, Jianghe Xu, Shouhong Ding, and Wenqiang Zhang. Shape matters: deformable patch attack. In *ECCV*, 2022.

[7] Zhaoyu Chen, Bo Li, Jianghe Xu, Shuang Wu, Shouhong Ding, and Wenqiang Zhang. Towards practical certifiable patch defense with vision transformer. In *CVPR*, 2022.

[8] Ping-Yeh Chiang, Renkun Ni, Ahmed Abdelkader, Chen Zhu, Christoph Studor, and Tom Goldstein. Certified defenses for adversarial patches. In *ICLR*, 2020.

[9] Edward Chou, Florian Tramer, and Giancarlo Pellegrino. Sentinet: Detecting localized universal attacks against deep learning systems. In *S&P Workshops)*, 2020.

[10] Jeremy M. Cohen, Elan Rosenfeld, and J. Zico Kolter. Certified adversarial robustness via randomized smoothing. In *ICML*, 2019.

[11] Christian Cosgrove, Adam Kortylewski, Chenglin Yang, and Alan L. Yuille. Robustness out of the box: Compositional representations naturally defend against black-box patch attacks. *arXiv preprint arXiv:2012.00558*, 2020.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[14] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *CVPR*, 2018.

[15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[16] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Arthur Mann, and Pushmeet Kohli. Scalable verified training for provably robust image classification. In *ICCV*, 2019.

[17] Husheng Han, Kaidi Xu, Xing Hu, Xiaobing Chen, Ling Liang, Zidong Du, Qi Guo, Yanzhi Wang, and Yunji Chen. ScaleCert: Scalable certified defense against adversarial patches with sparse superficial layers. In *NeurIPS*, 2021.

[18] Jamie Hayes. On visible adversarial perturbations & digital watermarking. In *CVPR Workshops*, 2018.

[19] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, 2022.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[21] Omer Hofman, Amit Giloni, Yarin Hayun, Ikuya Morikawa, Toshiya Shimizu, Yuval Elovici, and Asaf Shabtai. X-detect: Explainable adversarial patch detection for object detectors in retail. *arXiv preprint arXiv:2306.08422*, 2023.

[22] Danny Karmon, Daniel Zoran, and Yoav Goldberg. LaVAN: Localized and visible adversarial noise. In *ICML*, 2018.

[23] Alex Krizhevsky. Learning multiple layers of features from tiny images. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf, 2009.

[24] Hung Le and Ali Borji. What are the receptive, effective receptive, and projective fields of neurons in convolutional neural networks? *arXiv preprint arXiv:1705.07049*, 2017.

[25] Alexander Levine and Soheil Feizi. (De)randomized smoothing for certifiable defense against patch attacks. In *NeurIPS*, 2020.

[26] Junbo Li, Huan Zhang, and Cihang Xie. Vip: Unified certified detection and recovery for patch attack with vision transformers. In *ECCV*, 2022.

[27] Jiang Liu, Alexander Levine, Chun Pong Lau, Rama Chellappa, and Soheil Feizi. Segment and complete: Defending object detectors against adversarial patch attacks with robust patch detection. In *CVPR*, 2022.

[28] Xin Liu, Huanrui Yang, Ziwei Liu, Linghao Song, Yiran Chen, and Hai Li. DPATCH: an adversarial patch attack on object detectors. In *AAAI Workshops*, 2019.

[29] Giulio Lovisotto, Nicole Finnie, Mauricio Munoz, Chaithanya Kumar Mummadi, and Jan Hendrik Metzen. Give me your attention: Dot-product attention considered harmful for adversarial patch robustness. In *CVPR*, 2022.

[30] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard S. Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *NeurIPS*, 2016.

[31] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

[32] Michael McCoyd, Won Park, Steven Chen, Neil Shah, Ryan Roggenkemper, Minjune Hwang, Jason Xinyu Liu, and David A. Wagner. Minority reports defense: Defending against adversarial patches. In *ACNS Workshops*, 2020.

[33] Dongyu Meng and Hao Chen. Magnet: A two-pronged defense against adversarial examples. In *CCS*, 2017.

[34] Jan Hendrik Metzen and Maksym Yatsura. Efficient certified defenses against patch attacks on image classifiers. In *ICLR*, 2021.

[35] Matthew Mirman, Timon Gehr, and Martin T. Vechev. Differentiable abstract interpretation for provably robust neural networks. In *ICML*, 2018.

[36] Norman Mu and David Wagner. Defending against adversarial patches with robust self-attention. In *ICML Workshops*, 2021.

[37] Muzammal Naseer, Salman Khan, and Fatih Porikli. Local gradients smoothing: Defense against localized adversarial attacks. In *WACV*, 2019.

[38] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NeurIPS Workshops*, 2011.

[39] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *EuroS&P*, 2016.

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[41] Sukrut Rao, David Stutz, and Bernt Schiele. Adversarial training against location-optimized adversarial patches. In *ECCV Workshops*, 2020.

[42] Aniruddha Saha, Akshayvarun Subramanya, Koninika Patil, and Hamed Pirsiavash. Role of spatial context in adversarial robustness for object detection. In *CVPR Workshops*, 2020.

[43] Aniruddha Saha, Shuhua Yu, Arash Norouzzadeh, Wan-Yi Lin, and Chaithanya Kumar Mummadi. Revisiting image classifier training for improved certified robust defense against adversarial patches. *TMLR*, 2023.

[44] Hadi Salman, Saachi Jain, Eric Wong, and Aleksander Madry. Certified patch robustness via smoothed vision transformers. In *CVPR*, 2022.

[45] Vikash Sehwag, Chawin Sitawarin, Arjun Nitin Bhagoji, Arsalan Mosenia, Mung Chiang, and Prateek Mittal. Not all pixels are born equal: An analysis of evasion attacks under locality constraints. In *CCS*, 2018.

[46] Shawn Shan, Arjun Nitin Bhagoji, Haitao Zheng, and Ben Y Zhao. A real-time defense against website fingerprinting attacks. In *AISec@CCS*, 2021.

[47] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

[48] Bilel Tarchoun, Anouar Ben Khalifa, Mohamed Ali Mahjoub, Nael Abu-Ghazaleh, and Ihsen Alouani. Jedi: Entropy-based localization and removal of adversarial patches. In *CVPR*, 2023.

[49] Xingxing Wei, Ying Guo, and Jie Yu. Adversarial sticker: A stealthy attack method in the physical world. *TPAMI*, 2022.

[50] Ross Wightman, Hugo Touvron, and Hervé Jégou. Resnet strikes back: An improved training procedure in timm. In *NeurIPS Workshops*, 2021.

[51] Eric Wong and J. Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*, 2018.

[52] Tong Wu, Liang Tong, and Yevgeniy Vorobeychik. Defending against physically realizable attacks on image classification. In *ICLR*, 2020.

[53] Zuxuan Wu, Ser-Nam Lim, Larry S. Davis, and Tom Goldstein. Making an invisibility cloak: Real world adversarial attacks on object detectors. In *ECCV*, 2020.

[54] Chong Xiang, Arjun Nitin Bhagoji, Vikash Sehwag, and Prateek Mittal. Patchguard: A provably robust defense against adversarial patches via small receptive fields and masking. In *USENIX Security*, 2021.

[55] Chong Xiang, Saeed Mahloujifar, and Prateek Mittal. Patchcleanser: Certifiably robust defense against adversarial patches for any image classifier. In *USENIX Security*, 2022.

[56] Chong Xiang and Prateek Mittal. DetectorGuard: Provably securing object detectors against localized patch hiding attacks. In *CCS*, 2021.

[57] Chong Xiang and Prateek Mittal. Patchguard++: Efficient provable attack detection against adversarial patches. In *ICLR Workshop*, 2021.

[58] Chong Xiang, Charles R Qi, and Bo Li. Generating 3d adversarial point clouds. In *CVPR*, 2019.

[59] Chong Xiang, Chawin Sitawarin, Tong Wu, and Prateek Mittal. Short: Certifiably robust perception against adversarial patch attacks: A survey. In *VehicleSec*, 2023.

[60] Chong Xiang, Alexander Valtchanov, Saeed Mahloujifar, and Prateek Mittal. Objectseeker: Certifiably robust object detection against patch hiding attacks via patch-agnostic masking. In *S&P*, 2023.

[61] Chong Xiang, Tong Wu, Sihui Dai, Jonathan Petit, Suman Jana, and Prateek Mittal. Patchcure: Improving certifiable robustness, model utility, and computation efficiency of adversarial patch defenses. *arXiv preprint arXiv:2310.13076*, 2023.

[62] Kaidi Xu, Gaoyuan Zhang, Sijia Liu, Quanfu Fan, Mengshu Sun, Hongge Chen, Pin-Yu Chen, Yanzhi Wang, and Xue Lin. Adversarial t-shirt! evading person detectors in a physical world. In *ECCV*, 2020.

Table 8: Throughput results (img/s) with different batch sizes

Batch size	1	4	8	32
ViT-B	113.4	190.6	189.3	199.9
ViT14x2	111.5	192.9	193.8	203.9
PCURE-ViT14x2-k12	110.7	186.3	188.9	200.9
PatchCleanser	2.0	2.1	2.1	2.1
S-ViT	0.6	0.8	0.8	0.8

[63] Ke Xu, Yao Xiao, Zhaoheng Zheng, Kaijie Cai, and Ram Nevatia. Patchzero: Defending against adversarial patch attacks by detecting and zeroing the patch. In *WACV*, 2023.

[64] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *NDSS*, 2018.

[65] Chenglin Yang, Adam Kortylewski, Cihang Xie, Yinzhi Cao, and Alan Yuille. Patchattack: A black-box texture-based attack with reinforcement learning. In *ECCV*, 2020.

[66] Maksym Yatsura, Kaspar Sakmann, N Grace Hua, Matthias Hein, and Jan Hendrik Metzen. Certified defences against adversarial patch attacks on semantic segmentation. In *ICLR*, 2023.

[67] Zhanyuan Zhang, Benson Yuan, Michael McCoyd, and David Wagner. Clipped bagnet: Defending against sticker attacks with clipped bag-of-features. In *Deep Learning and Security Workshop (DLS)*, 2020.

A Additional Efficiency Experiments

Throughput with different batch sizes. In Section 4, we use a batch size of 4 for throughput evaluation. In Table 8, we report additional throughput results with different batch sizes. We can see that the throughput does not significantly change with a larger batch in our experiment setting.

FLOP and memory analyses. In Table 9, we augment our evaluation results (discussed in Table 2 and Table 3) and include latency, FLOP counts, and GPU memory consumption. We use a batch size of one for per-image inference latency. We estimate FLOPs using the `fvcore` libarary.[5] For GPU memory, we report the average of the maximum allocated GPU memory for each data batch. First, we can see that throughput and latency are (inversely) correlated since we use one GPU for both two experiments. Second, the FLOP count loosely correlates to the inference throughput: models with a smaller FLOP count usually have higher inference throughput. However, theoretically, it is possible that an algorithm that has fewer FLOPs but requires sequential execution can have a much larger latency than a parallelizable algorithm with more FLOPs when running on a GPU. Third, we can see a weaker connection between GPU memory consumption and inference throughput because memory consumption heavily depends on the implementation: for example, making inferences on masked images one by one consumes less memory than batched masked images. We can see that PCURE-ViT14x2-k12 and PCURE-ViT14x2-k11 have significantly different memory footprints because we used an optimized PATCHCURE implementation for pure SRF defenses; ECViT, ViP, and S-ViT have a low inference speed but similar memory consumption as vanilla ViT due to their special implementation.

[5] https://github.com/facebookresearch/fvcore/blob/main/docs/flop_count.md

Table 9: Performance for different models and defenses

Models	Accuracy (%) clean	Accuracy (%) robust	Throughput (img/s)	Latency (ms)	FLOP ($\times 10^9$)	Memory (MB)
ViT-B [13]	83.7	0	191.7	7.5	17.6	360.6
ResNet-50 [20]	80.1	0	295.5	11.8	4.1	254.8
BagNet-33 [3]	73.0	0	192.2	11.0	16.4	320.2
ViT14x2	79.4	0	195.2	9.2	17.6	362.0
ViT14x1	77.4	0	190.4	8.2	18.1	362.4
ViT2x2	72.7	0	166.7	8.8	21.1	368.1
PCURE-ViT14x2-k12	78.3	44.2	189.9	10.3	17.6	362.0
PCURE-ViT14x1-k12	76.3	47.1	182.0	9.8	18.1	362.4
PCURE-ViT2x2-k12	71.3	46.8	158.1	9.7	21.1	368.2
PCURE-BagNet17-k4	65.4	42.2	115.1	14.5	17.1	723.2
PCURE-BagNet33-k4	70.8	44.1	136.8	13.1	17.5	517.4
PCURE-BagNet45-k4	72.4	34.8	132.5	14.8	17.2	465.9
PCURE-ViT14x2-k11	78.3	45.8	109.0	13.8	27.4	513.9
PCURE-ViT14x2-k10	79.8	46.0	77.2	16.5	38.6	511.7
PCURE-ViT14x2-k9	80.0	46.2	58.4	21.1	48.8	515.6
PCURE-ViT14x2-k6	81.8	47.4	34.4	32.4	76.3	517.9
PCURE-ViT14x2-k3	82.1	47.8	23.9	44.1	101.8	515.8
PCURE-ViT14x2-k0	82.2	47.8	19.7	54.3	92.0	515.8
PCURE-ViT14x1-k0	82.5	53.7	8.3	117.2	217.5	764.9
PCURE-ViT2x2-k0	82.6	61.6	2.0	433.3	1084.0	2181.6
PatchCleanser [55]	82.5	61.1	2.1	460.6	1205.0	2302.8
PatchCleanser [55]	82.0	55.1	12.5	82.7	216.0	646.9
ECViT [7]	78.6	41.7	2.25	1593.9	920.8	344.5
ViP [26] (robust)	75.8	40.4	0.8	1637.2	3938.4	363.7
ViP [26] (efficient)	75.3	38.3	7.7	174.2	404.4	363.7
S-ViT [44] (robust)	73.2	38.2	0.8	1577.1	920.8	344.5
S-ViT [44] (efficient)	67.3	33.0	20.5	177.3	63.3	342.5
PatchGuard [54]	54.6	26.0	162.9	11.5	17.5	311.8
BagCert [34]	45.3	22.7	164.2	11.6	17.5	310.8
DRS [25]	44.4	14.0	1.5	2476.8	920.9	258.6
CBN [67]	49.5	7.1	166.3	11.6	17.5	311.8

B PatchCleanser and Double-masking

In this section, we discuss PatchCleanser [55] and its double-masking algorithm, which can be used as our secure operation $\mathrm{SO}(\cdot)$ in Algorithm 1. We generally follow the original presentation of the PatchCleanser paper [55], with small modifications to keep its notations consistent with this paper.

$\mathcal{R}$-covering mask set. The most important parameter of the double-masking algorithm is an $\mathcal{R}$-covering mask set. We copied the definition of $\mathcal{R}$-covering from the original

paper [55]. The mask $\mathbf{m} \in \mathcal{M}$ has a similar definition as the patch region set $\mathbf{r}$ discussed in Section 2.1: it is a binary tensor with the same shape as the input image $H \times W$; elements that correspond to the mask region take the value of zeros and otherwise ones.

Definition 1 ($\mathcal{R}$-covering)**.** *A mask set $\mathcal{M}$ is $\mathcal{R}$-covering if, for any patch in the patch region set $\mathcal{R}$, at least one mask from the mask set $\mathcal{M}$ can cover the entire patch, i.e.,*

$$\forall\, \mathbf{r} \in \mathcal{R},\ \exists\, \mathbf{m} \in \mathcal{M}\ \text{ s.t. }\ \mathbf{m}[i,j] \le \mathbf{r}[i,j],\ \forall (i,j)$$

Mask set generation. The PatchCleanser [55] discussed a systematic way to generate a mask set $\mathcal{M}$ and adjust the mask set size $|\mathcal{M}|$. For simplicity, we only discuss the procedure using a 1-D "image" example. To generate a mask set, PatchCleanser moves a mask over the input image. Formally, let us consider using a mask of width m over an image of size n. The first mask is placed at the image coordinate of 0 and thus covers indices from 0 to $m-1$. Next, the mask is moved with a stride of s across different image locations $\{0, s, 2s, \cdots, \lfloor \frac{n-m}{s} \rfloor s\}$. Finally, the last mask is placed at the index of $n-m$ in case the mask at $\lfloor \frac{n-m}{s} \rfloor s$ cannot cover the last m pixels. PatchCleanser then define a mask set $\mathcal{M}_{m,s,n}$ as:

$$\begin{aligned}\mathcal{M}_{m,s,n} = \{\mathbf{m} \in \{0,1\}^n \mid\ & \mathbf{m}[u] = 0, u \in [i, i+m);\\ & \mathbf{m}[u] = 1, u \notin [i, i+m);\ i \in I\}\\ I = \{0, s, 2s, \cdots, \lfloor \frac{n-m}{s} \rfloor s\} & \bigcup \{n-m\} \qquad (3)\end{aligned}$$

The mask set size can be computed as:

$$|\mathcal{M}_{m,s,n}| = |I| = \lceil \frac{n-m}{s} \rceil + 1 \qquad (4)$$

To adjust the mask set size, PatchCleanser proposed to change the mask stride s. To ensure the $\mathcal{R}$-covering property, PatchCleanser proved that the mask size needs to be at least $m = p + s - 1$ to cover patches no larger than the size of p. In our PATCHCURE implementation, we set $s = 1$ by default. We analyze the effect of stride s in our technical report [61].

Inference procedure. We present the inference procedure of the double-masking algorithm $\mathrm{SO}(\cdot)$ in Algorithm 2. Compared to the original double-masking algorithm (Algorithm 1 of [55]), we generalize its input image $\mathbf{x}$ as input image or intermediate feature map $\mathbf{f}$ and replace its vanilla model as our LRF model $\mathbb{M}_{\text{lrf}}$.

Certification procedure. We present the certification procedure CERT-$\mathrm{SO}(\cdot)$ in Algorithm 3. Compared to the original double-masking algorithm (Algorithm 2 of [55]), we generalize its input image $\mathbf{x}$ as input image or intermediate feature map $\mathbf{f}$, replace its vanilla model as our LRF model $\mathbb{M}_{\text{lrf}}$, and consider the feature-space threat model $\mathcal{A}^f_{\mathcal{R}}$.

Algorithm 2 Inference procedure of double-masking [55]

Input: Image or intermediate feature map $\mathbf{f}$, LRF model $\mathbb{M}_{\text{lrf}}$, $\mathcal{R}$-covering mask set $\mathcal{M}$
Output: Robust prediction $\bar{y}$

procedure SO($\mathbf{f}, \mathbb{M}_{\text{lrf}}, \mathcal{M}$)
  $\bar{y}_{\text{maj}}, \mathcal{P}_{\text{dis}} \leftarrow$ MASKPRED($\mathbf{f}, \mathbb{M}_{\text{lrf}}, \mathcal{M}$)
  if $\mathcal{P}_{\text{dis}} = \varnothing$ **then**
    return $\bar{y}_{\text{maj}}$ ▷ *Case I: agreed prediction*
  end if
  for each $(\mathbf{m}_{\text{dis}}, \bar{y}_{\text{dis}}) \in \mathcal{P}_{\text{dis}}$ **do** ▷ Second-rnd. mask
    $\bar{y}', \mathcal{P}' \leftarrow$ MASKPRED($\mathbf{f} \odot \mathbf{m}_{\text{dis}}, \mathbb{M}_{\text{lrf}}, \mathcal{M}$)
    if $\mathcal{P}' = \varnothing$ **then**
      return $\bar{y}_{\text{dis}}$ ▷ *Case II: disagreer prediction*
    end if
  end for
  return $\bar{y}_{\text{maj}}$ ▷ *Case III: majority prediction*
end procedure

procedure MASKPRED($\mathbf{f}, \mathbb{M}_{\text{lrf}}, \mathcal{M}$)
  $\mathcal{P} \leftarrow \varnothing$ ▷ A set for mask-prediction pairs
  for $\mathbf{m} \in \mathcal{M}$ **do** ▷ Enumerate every mask $\mathbf{m}$
    $\bar{y} \leftarrow \mathbb{M}_{\text{lrf}}(\mathbf{f} \odot \mathbf{m})$ ▷ Evaluate masked prediction
    $\mathcal{P} \leftarrow \mathcal{P} \bigcup \{(\mathbf{m}, \bar{y})\}$ ▷ Update set $\mathcal{P}$
  end for
  $\bar{y}_{\text{maj}} \leftarrow \arg\max_{y^*} |\{(\mathbf{m}, \bar{y}) \in \mathcal{P} \mid \bar{y} = y^*\}|$ ▷ Majority
  $\mathcal{P}_{\text{dis}} \leftarrow \{(\mathbf{m}, \bar{y}) \in \mathcal{P} \mid \bar{y} \neq \bar{y}_{\text{maj}}\}$ ▷ Disagreers
  return $\bar{y}_{\text{maj}}, \mathcal{P}_{\text{dis}}$
end procedure

Algorithm 3 Certification procedure of double-masking [55]

Input: Image or intermediate feature map $\mathbf{f}$, ground-truth label y, LRF model $\mathbb{M}_{\text{lrf}}$, mask set $\mathcal{M}$, feature-space threat model $\mathcal{A}^f_{\mathcal{R}}$
Output: Whether $\mathbf{f}$ has certified robustness

procedure CERT-SO($\mathbf{f}, \mathbb{M}_{\text{lrf}}, \mathcal{M}, \mathcal{A}^f_{\mathcal{R}}, y$)
  if $\mathcal{M}$ is not $\mathcal{R}$-covering **then** ▷ Insecure mask set
    return `False`
  end if
  for every $(\mathbf{m}_0, \mathbf{m}_1) \in \mathcal{M} \times \mathcal{M}$ **do**
    $\bar{y}' \leftarrow \mathbb{M}_{\text{lrf}}(\mathbf{f} \odot \mathbf{m}_0 \odot \mathbf{m}_1)$ ▷ Two-mask pred.
    if $\bar{y}' \neq y$ **then**
      return `False` ▷ Possibly vulnerable
    end if
  end for
  return `True` ▷ Certified robustness!
end procedure