PatchDEMUX: A Certifiably Robust Framework for Multi-label Classifiers Against Adversarial Patches

Supplementary Material

A. Certification Robustness Proofs

A.1. Baseline certification correctness

In this section, we provably demonstrate robustness for our baseline certification procedure. Specifically, we prove Theorem 1, which ensures correctness of the bounds returned by Algorithm 2. For convenience, we re-state the theorem.

Theorem 1 (Algorithm 2 Correctness)**.** *Suppose we have an image data point* $(\mathbf{x},\mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$*, a single-label CDPA* $SL\text{-}DEF$*, and a multi-label classification model* $\mathbb{F}: \mathcal{X} \to \mathcal{Y}$*. Then, under the patch threat model* $S_{\mathbf{x},\mathcal{R}}$ *the bounds returned by Algorithm 2 are correct.*

Proof. We first demonstrate that classes included in TP_{lower} will be guaranteed correctness. Consider an arbitrary class $i^* \in \{1, 2, \dots, c\}$ with label $\mathbf{y}[i^*] = 1$. If this class is included in TP_{lower}, then we must have $\boldsymbol{\kappa}[i^*] = 1$ (i.e., line 9 in Algorithm 2). This implies that on line 5 we must have $SL\text{-}CERT_{[\mathbb{F}[i^*],\sigma]}(\mathbf{x},\mathbf{y}[i^*],\mathcal{R}) = 1$. Now consider when Algorithm 1 reaches index $i^* \in \{1,2,...,c\}$ in the *for* loop on line 3. Because the datapoint $(\mathbf{x},\mathbf{y}[i^*])$ was certifiable, by Definition 2 we will have

$$SL\text{-}INFER_{[\mathbb{F}[i^*],\sigma]}(\mathbf{x}') = 1 \quad \forall \mathbf{x}' \in S_{\mathbf{x},\mathcal{R}}$$

This implies that every class accounted for in TP_{lower} will be successfully recovered by Algorithm 1 regardless of the attempted patch attack.

Next, we demonstrate that classes included in FN_{upper} will not be guaranteed correctness. Consider an arbitrary class $i^* \in \{1, 2, \dots, c\}$ with label $\mathbf{y}[i^*] = 1$. In this case we will have $\boldsymbol{\kappa}[i^*] = 0$, and thus classes included in FN_{upper} will have $SL\text{-}CERT_{[\mathbb{F}[i^*],\sigma]}(\mathbf{x},\mathbf{y}[i^*],\mathcal{R}) = 0$. Now consider when Algorithm 1 reaches index $i^* \in \{1,2,...,c\}$ in the *for* loop on line 3. By Definition 2 it is possible that

$$\exists \mathbf{x}' \in S_{\mathbf{x},\mathcal{R}} \quad | \quad SL\text{-}INFER_{[\mathbb{F}[i^*],\sigma]}(\mathbf{x}') = 0$$

Essentially, in the worst-case scenario these classes might be mispredicted and be false negatives. Thus, none of the classes included in FN_{upper} can be guaranteed correctness. Because every class with $\mathbf{y}[i^*] = 1$ will be accounted for by either TP_{lower} or FN_{upper} (mutually exclusive), we conclude that TP_{lower} will be the correct lower bound for objects recovered and FN_{upper} will be the correct upper bound for objects missed.

The correctness of the FP_{upper} bound can be shown in a similar fashion, albeit by considering classes with $\mathbf{y}[i^*] = 0$. □

A.2. Location-aware certification correctness

In this section we demonstrate the correctness of our novel location-based certification method. To do so, it is helpful to use the following lemma.

Lemma 1 (Algorithm 3 Tightness)**.** *Given that we have derived a bound on* FN *using the technique from Algorithm 2, Algorithm 3 will return a new bound* $FN_{new} \leq FN$*.*

Proof. We will show that FN_{new} provides a tighter bound (i.e., the inequality $FN_{new} \leq FN$ is true). To see this, we note as per lines 13 and 14 of Algorithm 3 that the worst-case sum will occur if some patch location is vulnerable for every false negative. Because summation is done over the set of false negatives, this implies the worst-case sum is FN. □

We also provide a formal definition for the concept of a *vulnerability status array*; recall that this array extends the certification procedure for a single-label CDPA (Sec. 3.4.1). We leverage similar notation as Eq. (2).

Definition 3 (Vulnerability status array)**.** *Suppose we have datapoint* $(\mathbf{x},y)$*, a single-label classifier* $\mathbb{F}_s: \mathcal{X} \to \{1,2,...,c\}$*, a certification procedure* $SL\text{-}CERT$ *with security parameters* σ *from a single-label CDPA, and patch locations* $\mathcal{R}$*. Then we define the* vulnerability status array $\boldsymbol{\lambda} := SL\text{-}CERT_{[\mathbb{F}_s,\sigma]}(\mathbf{x},y,\mathcal{R}) \in \{0,1\}^{|\mathcal{R}|}$ *such that if* $\boldsymbol{\lambda}[\mathbf{r}] = 1$ *for a patch location* $\mathbf{r} \in \mathcal{R}$ *then*[5]

$$SL\text{-}INFER_{[\mathbb{F}_s,\sigma]}(\mathbf{r} \circ \mathbf{x} + (\mathbf{1} - \mathbf{r}) \circ \mathbf{x}') = y \quad \forall \mathbf{x}' \in \mathcal{X}$$

Essentially, the vulnerability status array $\boldsymbol{\lambda}$ denotes the certification status of individual patch locations $\mathbf{r} \in \mathcal{R}$.

We can now prove Theorem 2. For convenience we re-state the theorem.

Theorem 2 (Algorithm 3 Correctness)**.** *Suppose we have an image data point* $(\mathbf{x},\mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$*, a single-label CDPA* $SL\text{-}DEF$*, and a multi-label classification model* $\mathbb{F}: \mathcal{X} \to \mathcal{Y}$*. If* $SL\text{-}CERT$ *returns the vulnerability status array* $\boldsymbol{\lambda}$ *associated with each* $\mathbf{r} \in \mathcal{R}$*, then under the patch threat model* $S_{\mathbf{x},\mathcal{R}}$ *the bounds from Algorithm 3 are correct and stronger than Algorithm 2.*

Proof. We will demonstrate the correctness and tightness of the new bound FN_{new} proposed in Algorithm 3. We first note as per Lemma 1 that $FN_{new} \leq FN$; this ensures that the new bound will be stronger than Algorithm 2. In the case with equality $FN_{new} = FN$, correctness is guaranteed by Theorem 1. We thus focus on the case with strict inequality $FN_{new} < FN$.

Define $\mathbf{r_{opt}} \in \mathcal{R}$ as the patch location which induces the maximum number of false negatives on line 14 of Algorithm 3.

[5] For the term $\boldsymbol{\lambda}[\mathbf{r}] = 1$ we slightly abuse notation and use $\mathbf{r}$ to refer to the index associated with the patch location

By assumption, a total of $FN - FN_{new} > 0$ false negatives will have contributed a value of 0 to the sum $fnTotal[\mathbf{r_{opt}}]$ on line 13. Consider an arbitrary such class $i^* \in \{1,2,...,c\}$. Because the $fnCertFails$ value for this class at patch location $\mathbf{r_{opt}}$ is 0, on line 10 we must have for $\boldsymbol{\lambda} := SL\text{-}CERT_{[\mathbb{F}[i^*],\sigma]}(\mathbf{x},\mathbf{y}[i^*],\mathcal{R})$

$$\boldsymbol{\lambda}[\mathbf{r_{opt}}] = 1$$

As per Definition 3, this means that we will have

$$SL\text{-}INFER_{[\mathbb{F}[i^*],\sigma]}(\mathbf{r_{opt}} \circ \mathbf{x} + (\mathbf{1} - \mathbf{r_{opt}}) \circ \mathbf{x}') = 1 \quad \forall \mathbf{x}' \in \mathcal{X}$$

In other words, $SL\text{-}INFER$ will be robust against any patch attack contained in location $\mathbf{r_{opt}} \in \mathcal{R}$. Because the patch must be placed at the optimal location $\mathbf{r_{opt}}$, this implies that Algorithm 1 will return the correct prediction for class i^* as desired. Overall, each of the $FN - FN_{new}$ classes will now be certified true positives instead of false negatives, and thus the new bounds from Algorithm 3 will be correct. □

B. Double-masking Algorithm from PatchCleanser

In this section, we provide a brief outline of the double-masking algorithm from the PatchCleanser defense and how it integrates into the PatchDEMUX framework; recall from Sec. 4.1 that PatchCleanser is the current SOTA single-label CDPA. For more details, we direct the reader to the original reference by Xiang et al. [33].

B.1. Double-masking overview

At a glance, the double-masking algorithm works by curating a specialized set of masks, $\mathcal{M} \subseteq \{0,1\}^{w \times h}$, to recover the output label $y \in \{1,2,...,c\}$ for certifiable input images $\mathbf{x} \in \mathcal{X}$ [33]. More specifically, these masks satisfy the following $\mathcal{R}$-*covering* property from Xiang et al. [33].

Definition 4 ($\mathcal{R}$-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.,*

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

Here $\mathcal{R}$ refers to the set of patch locations from Eq. (2), and $\mathcal{M}$ represents binary matrices where elements inside the mask are 0 and elements outside the mask are 1 [33]. Given an input image size $n_1 \times n_2$, an upper estimate on patch size[6] p, and number of desired masks $k_1 \times k_2$, a procedure from Xiang et al. [33] can readily create a mask set $\mathcal{M}$ with stride length $s_1 \times s_2$ and mask size $m_1 \times m_2$ which is $\mathcal{R}$-covering. The patch size p and mask number $k_1 \times k_2$ serve as security parameters, where the former corresponds to the threat level of $\mathcal{R}$ (i.e., larger patches will necessitate larger masks) and the latter represents a computational budget (i.e., more masks will require more checks to be performed) [33].

[6] PatchCleanser provides an option to specify the patch size for each axis; we simplify the notation here for convenience

Once the $\mathcal{R}$-covering mask set $\mathcal{M}$ is generated, the double-masking inference procedure removes the patch by selectively occluding the image $\mathbf{x} \in \mathcal{X}$ with mask pairs $\mathbf{m_0}, \mathbf{m_1} \in \mathcal{M} \times \mathcal{M}$. Correctness is verified through the associated certification procedure, which checks if predictions on $\mathbf{x}$ are preserved across all possible mask pairs [33].

B.2. Double-masking inference procedure

Algorithm 4 *The double-masking inference procedure from PatchCleanser [33]*

Input: Image $\mathbf{x} \in \mathcal{X}$, single-label classifier $\mathbb{F}_s : \mathcal{X} \to \{1,2,...,c\}$, $\mathcal{R}$-covering mask set $\mathcal{M}$
Output: Prediction $\hat{y} \in \{1,2,...,c\}$

1: **procedure** DOUBLEMASKINGINFER($\mathbf{x}, \mathbb{F}_s, \mathcal{M}$)
2: $\quad \hat{y}_{maj}, \mathcal{P}_{dis} \leftarrow$ MASKPRED($\mathbf{x}, \mathbb{F}_s, \mathcal{M}$) ▷ First-round
3: $\quad$ **if** $\mathcal{P}_{dis} = \emptyset$ **then**
4: $\quad\quad$ **return** $\hat{y}_{maj}$ ▷ Case I: agreed prediction
5: $\quad$ **end if**
6: $\quad$ **for** each $(\mathbf{m_{dis}}, \hat{y}_{dis}) \in \mathcal{P}_{dis}$ **do** ▷ Second round
7: $\quad\quad \hat{y}', \mathcal{P}' \leftarrow$ MASKPRED($\mathbf{x} \circ \mathbf{m_{dis}}, \mathbb{F}_s, \mathcal{M}$)
8: $\quad\quad$ **if** $\mathcal{P}' = \emptyset$ **then**
9: $\quad\quad\quad$ **return** $\hat{y}_{dis}$ ▷ Case II: disagreer pred.
10: $\quad\quad$ **end if**
11: $\quad$ **end for**
12: $\quad$ **return** $\hat{y}_{maj}$ ▷ Case III: majority prediction
13: **end procedure**

Input: Image $\mathbf{x} \in \mathcal{X}$, single-label classifier $\mathbb{F}_s : \mathcal{X} \to \{1,2,...,c\}$, $\mathcal{R}$-covering mask set $\mathcal{M}$
Output: Majority prediction $\hat{y}_{maj} \in \{1,2,...,c\}$, disagreer masks $\mathcal{P}_{dis}$

14: **procedure** MASKPRED($\mathbf{x}, \mathbb{F}_s, \mathcal{M}$)
15: $\quad \mathcal{P} \leftarrow \emptyset$ ▷ A set for mask-prediction pairs
16: $\quad$ **for** $\mathbf{m} \in \mathcal{M}$ **do** ▷ Enumerate every mask $\mathbf{m}$
17: $\quad\quad \hat{y} \leftarrow \mathbb{F}_s(\mathbf{x} \circ \mathbf{m})$ ▷ Evaluate masked prediction
18: $\quad\quad \mathcal{P} \leftarrow \mathcal{P} \bigcup \{(\mathbf{m}, \hat{y})\}$ ▷ Update set $\mathcal{P}$
19: $\quad$ **end for**
20: $\quad \hat{y}_{maj} \leftarrow \text{argmax}_{y^*} |\{(\mathbf{m}, \hat{y}) \in \mathcal{P} | \hat{y} = y^*\}|$ ▷ Majority
21: $\quad \mathcal{P}_{dis} \leftarrow \{(\mathbf{m}, \hat{y}) \in \mathcal{P} | \hat{y} \neq \hat{y}_{maj}\}$ ▷ Disagreers
22: $\quad$ **return** $\hat{y}_{maj}, \mathcal{P}_{dis}$
23: **end procedure**

The double-masking inference procedure from Xiang et al. [33] is outlined in Algorithm 4. It works by running up to two rounds of masking on the input image $\mathbf{x} \in \mathcal{X}$. In each round, the single-label classifier $\mathbb{F}_s : \mathcal{X} \to \{1,2,...,c\}$ is queried on copies of $\mathbf{x}$ which have been augmented by masks $\mathbf{m} \in \mathcal{M}$ [33].

- *First-round masking:* The classifier runs $\mathbb{F}_s(\mathbf{m} \circ \mathbf{x})$ for every mask $\mathbf{m} \in \mathcal{M}$ (line 2). If there is consensus, this is returned as the overall prediction (line 4); the intuition is that a clean image with no patch will be predicted correctly regardless of the mask present [33]. Otherwise, the minority/"disagreer" predictions trigger a second-round of masking (line 6). This

is done to determine whether to trust the majority prediction $\hat{y}_{maj}$ or one of the disagreers [33].

- *Second-round masking:* For each disagreer mask $\mathbf{m_{dis}}$, the classifier runs $\mathbb{F}_s(\mathbf{x} \circ \mathbf{m_{dis}} \circ \mathbf{m})$ for every mask $\mathbf{m} \in \mathcal{M}$ to form *double-mask predictions* [33]. If there is consensus, the disagreer label $\hat{y}_{dis}$ associated with $\mathbf{m_{dis}}$ is returned as the overall prediction (lines $6-10$). The intuition is that consensus is likely to occur if $\mathbf{m_{dis}}$ successfully covered the patch [33]. Otherwise, $\mathbf{m_{dis}}$ is ignored and the next available disagreer mask is considered; the assumption here is that $\mathbf{m_{dis}}$ failed to cover the patch [33]. Finally, if none of the disagreer masks feature consensus the majority label $\hat{y}_{maj}$ from the first-round is returned instead (line 12).

A key property of this method is that it is architecture agnostic and can be integrated with any single-label classifier [33].

B.3. Double-masking certification procedure

Algorithm 5 *The double-masking certification procedure from PatchCleanser [33]*

Input: Image $\mathbf{x} \in \mathcal{X}$, ground-truth $y \in \{1,2,...,c\}$, single-label classifier $\mathbb{F}_s : \mathcal{X} \to \{1,2,...,c\}$, patch locations $\mathcal{R}$, $\mathcal{R}$-covering mask set $\mathcal{M}$
Output: Overall certification status of $(\mathbf{x},y)$, vulnerability status array $\boldsymbol{\lambda} \in \{0,1\}^{|\mathcal{M}|}$

1: **procedure** DOUBLEMASKINGCERT($\mathbf{x},y,\mathbb{F}_s,\mathcal{R},\mathcal{M}$)
2: $certVal \leftarrow 1$
3: $\boldsymbol{\lambda} \leftarrow [1]^{|\mathcal{M}|}$
4: **if** $\mathcal{M}$ is not $\mathcal{R}$-covering **then** ▷ Insecure mask set
5: **return** 0, $[0]^{|\mathcal{M}|}$
6: **end if**
7: **for** every $(\mathbf{m_0},\mathbf{m_1}) \in \mathcal{M} \times \mathcal{M}$ **do**
8: $\hat{y}' \leftarrow \mathbb{F}_s(\mathbf{x} \circ \mathbf{m_0} \circ \mathbf{m_1})$ ▷ Two-mask prediction
9: **if** $\hat{y}' \neq y$ **then**
10: $certVal \leftarrow 0$ ▷ Input possibly vulnerable
11: $\boldsymbol{\lambda}[\mathbf{m_0}],\boldsymbol{\lambda}[\mathbf{m_1}] \leftarrow 0,0$ ▷ Vulnerable masks
12: **end if**
13: **end for**
14: **return** $certVal,\boldsymbol{\lambda}$
15: **end procedure**

The double-masking certification procedure from Xiang et al. [33] is outlined in Algorithm 5; we extend the original version to additionally return a vulnerability status array $\boldsymbol{\lambda}$. It works by first ensuring that the mask set $\mathcal{M}$ is $\mathcal{R}$-covering (line 4); otherwise, no guarantees on robustness can be made. Then, during the *for* loop on lines $7-13$ the procedure computes $\mathbb{F}_s(\mathbf{x} \circ \mathbf{m_0} \circ \mathbf{m_1})$ for every possible mask pair $\mathbf{m_0},\mathbf{m_1} \in \mathcal{M} \times \mathcal{M}$ [33]. If all of the predictions are the label y, then $(\mathbf{x},y)$ is certifiable and $certVal$ is set to 1; recall from Definition 2 that this implies that the inference procedure Algorithm 4 will be correct regardless of an attempted patch attack. Otherwise, $certVal$ is set to 0 and the $\boldsymbol{\lambda}$ array is updated to reflect vulnerable points.

The correctness of $certVal$ is guaranteed by the following theorem. Essentially, if predictions across all possible mask pairs are correct, it ensures that each of the three cases in Algorithm 4 will work as intended [33].

Theorem 3. *Suppose we have an image data point $(\mathbf{x},y)$, a single-label classification model $\mathbb{F}_s : \mathcal{X} \to \{1,2,...,c\}$, a patch threat model $S_{\mathbf{x},\mathcal{R}}$, and a $\mathcal{R}$-covering mask set $\mathcal{M}$. If $\mathbb{F}_s(\mathbf{x} \circ \mathbf{m_0} \circ \mathbf{m_1}) = y$ for all $\mathbf{m_0},\mathbf{m_1} \in \mathcal{M} \times \mathcal{M}$, then Algorithm 4 will always return a correct label.*

Proof. This theorem is proved in Xiang et al. [33]. □

We next consider the vulnerability status array $\boldsymbol{\lambda} \in \{0,1\}^{|\mathcal{M}|}$ returned by Algorithm 5. Notice that the length of the array is $|\mathcal{M}|$ rather than $|\mathcal{R}|$; this is a helpful consequence of the $\mathcal{R}$-covering property of the mask set $\mathcal{M}$, which ensures that every patch location $\mathbf{r} \in \mathcal{R}$ will be contained in at least one of the masks $\mathbf{m} \in \mathcal{M}$. As such, an implementation-level abstraction is possible for PatchCleanser where each element $\boldsymbol{\lambda}[\mathbf{m}]$ summarizes the vulnerability status for all patch locations contained within the mask $\mathbf{m} \in \mathcal{M}$. The correctness of this construction can be demonstrated through the following lemma.

Lemma 2. *Suppose we have an image data point $(\mathbf{x},y)$, a single-label classification model $\mathbb{F}_s : \mathcal{X} \to \{1,2,...,c\}$, a patch threat model $S_{\mathbf{x},\mathcal{R}}$, and a $\mathcal{R}$-covering mask set $\mathcal{M}$. Then the array $\boldsymbol{\lambda} \in \{0,1\}^{|\mathcal{M}|}$ returned by Algorithm 5 will be a valid vulnerability status array that satisfies Definition 3.*

Proof. Define $\mathcal{R}^* \subseteq \mathcal{R}$ as the set of patch locations contained in an arbitrary mask $\mathbf{m}^* \in \mathcal{M}$. To demonstrate the validity of $\boldsymbol{\lambda}$, we need to show that $\boldsymbol{\lambda}[\mathbf{m}^*] = 1$ implies Algorithm 4 will be protected from all attacks located in $\mathcal{R}^*$. To do so, we first note that we will only have $\boldsymbol{\lambda}[\mathbf{m}^*] = 1$ in Algorithm 5 if $\mathbb{F}_s(\mathbf{x} \circ \mathbf{m}^* \circ \mathbf{m}) = y$ for all $\mathbf{m} \in \mathcal{M}$; otherwise, $\boldsymbol{\lambda}[\mathbf{m}^*]$ would have been marked with 0 at some point.

We can use this robustness property to guarantee correctness in Algorithm 4. Suppose we have an arbitrary patch attack with a location in $\mathcal{R}^*$ and that $\boldsymbol{\lambda}[\mathbf{m}^*] = 1$. In the first-round masking stage the attack will be completely covered by the mask $\mathbf{m}^*$ (due to the $\mathcal{R}$-covering property) and form the masked image $\mathbf{x} \circ \mathbf{m}^* \in \mathcal{X}$. Note that this is the same as the image $\mathbf{x} \circ \mathbf{m}^* \circ \mathbf{m}^* \in \mathcal{X}$; therefore, the robustness property from above will guarantee that $\mathbb{F}_s(\mathbf{x} \circ \mathbf{m}^*) = y$. We have thus shown that the correct prediction will be represented at least once in the first-round, leaving three possible scenarios.

- *Scenario #1 (consensus):* In this scenario, the classifier returns the correct prediction y for every first-round mask. Then, line 4 of Algorithm 4 will ensure that y is correctly returned as the overall prediction.
- *Scenario #2 (majority of masks are correct):* In this scenario, the classifier returns the correct prediction y for the majority of first-round masks. The set of disagreer masks, $\mathcal{M}_{dis} \subseteq \mathcal{M}$, will thus run a second round of masking. This eventually requires computing $\mathbb{F}_s(\mathbf{x} \circ \mathbf{m_{dis}} \circ \mathbf{m}^*)$ for each $\mathbf{m_{dis}} \in \mathcal{M}_{dis}$. By leveraging symmetry and the robustness property from earlier, these are all guaranteed to return the correct prediction y. Therefore, none of the disagreer masks will have consensus in the second-round, and line 12 of Algorithm 4 will ensure that y is correctly returned as the overall prediction.

- *Scenario #3 (minority of masks are correct):* In this scenario, the classifier returns the correct prediction y for a minority of first-round masks. This implies that $\mathbf{m}^*$ will be a disagreer mask. During the second round of masking, the robustness property from earlier will ensure that $\mathbb{F}_s(\mathbf{x} \circ \mathbf{m}^* \circ \mathbf{m}) = y$ for each $\mathbf{m} \in \mathcal{M}$. Therefore, we will have consensus in the second round of $\mathbf{m}^*$. Because disagreers with incorrect predictions will fail to have consensus (i.e., using the logic from *Scenario #2*), line 9 of Algorithm 4 will ensure that y is correctly returned as the overall prediction.

Overall, we conclude that Algorithm 4 will return the correct prediction y. We have thus shown that $\boldsymbol{\lambda}[\mathbf{m}^*] = 1$ implies Algorithm 4 will be protected from any arbitrary patch attack located in $\mathcal{R}^*$, as desired. □

B.4. Integration with PatchDEMUX

To integrate PatchCleanser into the PatchDEMUX framework, we first generate a $\mathcal{R}$-covering set of masks $\mathcal{M}$; the mask set $\mathcal{M}$ essentially serves as a holistic representation of the security parameters σ. We then incorporate Algorithm 4 into the PatchDEMUX inference procedure (Algorithm 1) and Algorithm 5 into the PatchDEMUX certification procedure (Algorithm 2). Finally, we use the location-aware certification method (Algorithm 3) with the vulnerability status arrays expressed in terms of masks.

C. Further Details on Evaluation Metrics

In this section, we discuss the evaluation metrics from Sec. 4 and Sec. 5 in more detail.

C.1. Threshold analysis

We evaluate multi-label classifiers by computing precision and recall metrics over a variety of different thresholds; classes with output higher than the threshold are predicted 1, otherwise 0. This helps establish a large set of evaluation data from which to build precision-recall plots. We start by evaluating a set of *standard thresholds*:

$$T_{standard} := \{0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9\}$$

We then evaluate a set of *high-value thresholds*. This helps fill out the *low recall-high precision* region of a precision-recall curve:

$$T_{high} := \{0.91,0.92,0.93,0.94,0.95,0.96,0.97,0.98,0.99\}$$

We next evaluate a set of *very high-value thresholds*. These evaluations provide points at which recall is close to 0%:

$$T_{veryhigh} := \{0.999,0.9999,0.99999\}$$

Finally, we evaluate a set of mid-value thresholds. These help to smoothen out a precision-recall curve:

$$T_{mid} := T_{mid1} \cup T_{mid2} \cup T_{mid3} \cup T_{mid4}$$

where

$$T_{mid1} := \{0.5+0.02 \cdot t : t \in \{1,2,3,4\}\}$$
$$T_{mid2} := \{0.6+0.02 \cdot t : t \in \{1,2,3,4\}\}$$
$$T_{mid3} := \{0.7+0.02 \cdot t : t \in \{1,2,3,4\}\}$$
$$T_{mid4} := \{0.8+0.02 \cdot t : t \in \{1,2,3,4\}\}$$

For ViT-based models specifically, we found that the *low precision-high recall* region of a precision-recall curve does not readily appear if we limit evaluation to the thresholds outlined above. We thus further evaluate the following set of *low-value* thresholds for ViT-based models:

$$T_{low} := \{5 \cdot 10^{-5}, 10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 0.01, 0.05\}$$

In order to obtain precision values at key recall levels (i.e., 25%, 50%, 75%), we can perform linear interpolation between relevant recall bounds. However, recall values computed using the thresholds above are often not close enough to these target values. To this end, we use an iterative bisection scheme to find overestimated and underestimated bounds within 0.5 points of the target recalls. The precision values are then calculated by linearly interpolating between these bounds.

C.2. Computing average precision

In order to compute an approximation for average precision, we leverage the area-under-the-curve (AUC) of the associated precision-recall curves. However, in practice the threshold analysis from Appendix C.1 can result in different leftmost points for the precision-recall curves. In order to enforce consistency, we fix the leftmost points for each precision-recall plot at exactly 25% recall. Then, the AUC is computed using the trapezoid sum technique and normalized by a factor of 0.75 (i.e., the ideal precision-recall curve). Note that we pick 25% recall because a few evaluations under this value demonstrate floating-point precision errors (i.e., the required threshold is too high).

D. Resnet Architecture Analysis

In this section, we report results for PatchDEMUX while using the Resnet architecture [3]; we leverage the same defense fine-tuning routine as Sec. 4.2 to achieve stronger performance. Experiments are done on the MS-COCO 2014 validation dataset. The precision values associated with key recall levels are in Tab. 2. Fig. 5 features precision-recall plots, while AP values are present in Tab. 2.

We find that the Resnet and ViT architectures show similar qualitative trends. For instance, defended clean performance is close to undefended performance across a variety of thresholds (see Fig. 5a and Tab. 2a). The Resnet-based variant of PatchDEMUX also achieves non-trivial robustness, with a certified average precision of 37.544%. In general, the precision-recall curves for the two architectures are similar in shape across all four evaluation settings.

Despite these similarities, the ViT model consistently outperforms the Resnet model. More specifically, the ViT-based variant of PatchDEMUX provides a $\sim$4 point boost to clean

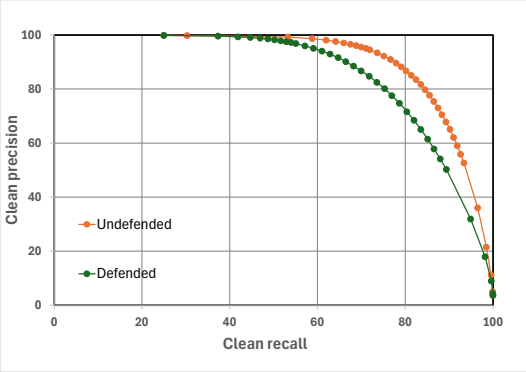
Table 2. *PatchDEMUX performance with Resnet architecture on the MS-COCO 2014 validation dataset. Precision values are evaluated at key recall levels along with the approximated average precision. We assume the patch attack is at most 2% of the image area and use a computational budget of 6×6 masks.*

(a) *Clean setting precision values*

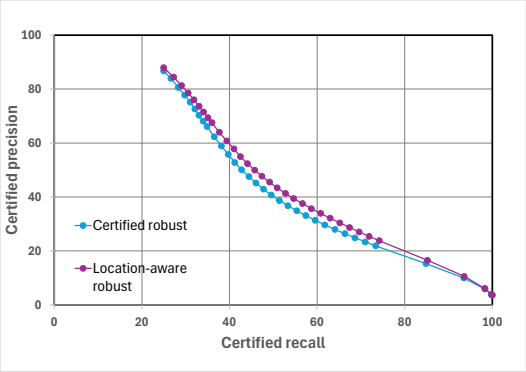
Architecture	Resnet			
Clean recall	25%	50%	75%	*AP*
Undefended	99.832	99.425	92.341	87.608
Defended	99.835	98.257	80.612	81.031

(b) *Certified robust setting precision values*

Architecture	Resnet			
Certified recall	25%	50%	75%	*AP*
Certified robust	86.696	40.190	20.959	34.859
Location-aware	87.950	44.373	23.202	37.544



(a) *Clean setting precision-recall curves*



(b) *Certified robust setting precision-recall curves*

Figure 5. *PatchDEMUX precision-recall curves with Resnet architecture over the MS-COCO 2014 validation dataset. We consider the clean and certified robust evaluation settings. We assume the patch attack is at most 2% of the image area and use a computational budget of 6×6 masks.*

AP in the defended clean setting and a ∼7 point boost to certified AP in the two certified robust settings (see Tab. 1). This improvement might be attributable to the training procedure of vision transformers, which involves a masking process that is similar in concept to PatchCleanser's double-masking algorithm [9, 33].

E. Location-aware Certification Analysis

Table 3. *ViT-based PatchDEMUX performance with different location-aware attackers. Experiments performed on the MS-COCO 2014 validation dataset. Precision values are evaluated at key recall levels along with the approximated average precision. We assume the patch attack is at most 2% of the image area and use a computational budget of 6×6 masks.*

Architecture	ViT			
Certified recall	25%	50%	75%	*AP*
FP attacker	95.724	62.132	33.112	49.474
FN attacker	95.971	58.158	27.199	45.951
Location-aware robust	95.670	56.038	26.375	44.902
Certified robust	95.369	50.950	22.662	41.763

In this section we investigate the location-aware certification approach from Sec. 3.4 in more detail.

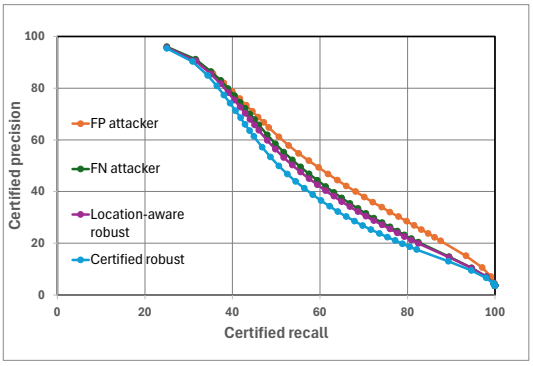


Figure 6. *ViT-based PatchDEMUX precision-recall curves with different location-aware attackers. Experiments performed on the MS-COCO 2014 validation dataset. The baseline certified robust evaluation setting is included for comparison. We assume the patch attack is at most 2% of the image area and use a computational budget of 6×6 masks.*

E.1. Attack vectors

Based on Sec. 3.4.2, there are a couple different ways to evaluate the robustness provided by the location-aware method.

- *FN attacker:* Here, we only track vulnerability status arrays $\boldsymbol{\lambda}$ for false negatives. Intuitively, this corresponds to the optimal attacker from Sec. 3.4.2 constructing a patch with the sole intent of increasing false negatives (i.e., a FN attack). In

Algorithm 3, tie-breakers are decided by picking the location which induces more false positives.

- *FP attacker:* In this scenario we only track vulnerability status arrays $\boldsymbol{\lambda}$ for false positives. This corresponds to the optimal attacker from Sec. 3.4.2 constructing a patch with the sole intent of increasing false positives (i.e., a FP attack). In the FP version of Algorithm 3, tie-breakers are decided by picking the location which induces more false negatives.

We also consider "worst case" performance where we simultaneously determine the worst patch location for both false negatives and false positives. Note that these two locations do not have to be identical, and as a result this "worst case" performance is not necessarily realizable. However, we evaluate this approach because it represents the theoretical lower bound on robustness for Algorithm 3 given an arbitrarily motivated attacker.

E.2. Experiment results

We now empirically compare the different attack vectors possible under location-aware certification. We consider the ViT architecture alone as it provides better performance compared to Resnet. In addition, we leverage the same pre-trained model checkpoints used in Sec. 4.2 for consistency. Experiments are done on the MS-COCO 2014 validation dataset. Precision values corresponding to different attackers are present in Tab. 3, while precision-recall plots are in Fig. 6.

Provable robustness improvements. Regardless of the attack strategy employed, location-aware certification provides improved robustness compared to the baseline certified robust setting; this is expected due to Theorem 2. Improvement is most notable in both the *mid recall-mid precision* and *high recall-low precision* sections of the precision-recall curve. Overall, the most favorable evaluation approach provides an $\sim$8 point increase in certified AP compared to the baseline. Despite these improvements, location-aware certification does not fundamentally change the shape of the robust precision-recall curve under any of the three attack settings.

Asymmetric attack performance. Interestingly, location-aware certification provides the strongest robustness guarantees under the FP attack strategy. This is likely due to the asymmetric dependence of precision and recall metrics on false positives. Specifically, both metrics depend on false negatives[7], but the recall metric does not depend on false positives. This makes FP attacks "weaker" relative to other methods.

F. Defense Fine-tuning for PatchDEMUX

Single-label CDPAs often leverage defense fine-tuning routines to improve the robustness of underlying single-label classifiers; these work by training the model on specially augmented data [8, 28, 33]. In this section, we investigate whether some of these routines can extend to multi-label classifiers and improve PatchDEMUX performance. We specifically consider fine-tuning strategies used by PatchCleanser, as PatchCleanser is the certifiable backbone for PatchDEMUX in this work.

[7]Precision indirectly depends on false negatives via the true positive count

F.1. Defense fine-tuning techniques for PatchCleanser

Two different defense fine-tuning techniques have been used to improve the performance of PatchCleanser: *Random Cutout* [8] and *Greedy Cutout* [28]. The former works by placing two square masks at random locations on training images, with each mask covering at most 25% of the image area [8, 33]. Xiang et al. [33] found that Random Cutout fine-tuning provides significant boosts to the robustness of PatchCleanser; intuitively, using cutout masks for defense fine-tuning helps the underlying model become more tolerant to occlusion effects from double-masking procedures. Later, Saha et al. [28] proposed the Greedy Cutout fine-tuning procedure and demonstrated superior performance to Random Cutout for PatchCleanser. This approach works by augmenting each training image with the pair of certification masks that greedily induce the highest loss.

F.2. Defense fine-tuning methodology

In our experiments we compare the following three defense fine-tuning methods, which are representative of settings used in prior work [8, 28, 33].

- Random Cutout fine-tuning with two square 25% masks
- Greedy Cutout fine-tuning with 6×6 certification masks
- Greedy Cutout fine-tuning with 3×3 certification masks

For Greedy Cutout, we compute the loss for masks while models are in evaluation mode; this approach helps avoid consistency issues associated with batch normalization. We do not consider the more complex multi-size greedy cutout approach from Saha et al. [28] due to difficulties with mask decompositions.

To train the model with these methods, we first obtain existing checkpoints for the MS-COCO 2014 classification task [3, 20]. We then follow the training methodology for multi-label classifiers outlined by Ben-Baruch et al. [3]. Specifically, we use asymmetric loss (ASL) as the loss function, a 1cycle learning rate policy with max learning rate $\alpha_{max} = 5.0 \cdot 10^{-5}$, automatic mixed precision (AMP) for faster training, and exponential moving average (EMA) of model checkpoints for improved inference [3]. Models are fine-tuned on copies of the MS-COCO 2014 training dataset augmented by Random Cutout and Greedy Cutout. We use the Adam optimizer for 5 epochs, and best checkpoints are picked according to the loss on held out data[8]. A cluster of NVIDIA A100 40GB GPUs are used to perform the fine-tuning.

F.3. Experiment results

Results for the different defense fine-tuning routines are in Tab. 4. In addition, precision-recall plots comparing the defense fine-tuning routines for each of the four PatchDEMUX evaluation settings are present in Fig. 7. We consider the ViT architecture alone as it provides better performance compared to Resnet. Experiments are done on the MS-COCO 2014 validation dataset.

Defense fine-tuning boosts performance. In general, we find that using a defense fine-tuning routine of any kind leads

[8]We find that fine-tuning for longer leads to overfitting.

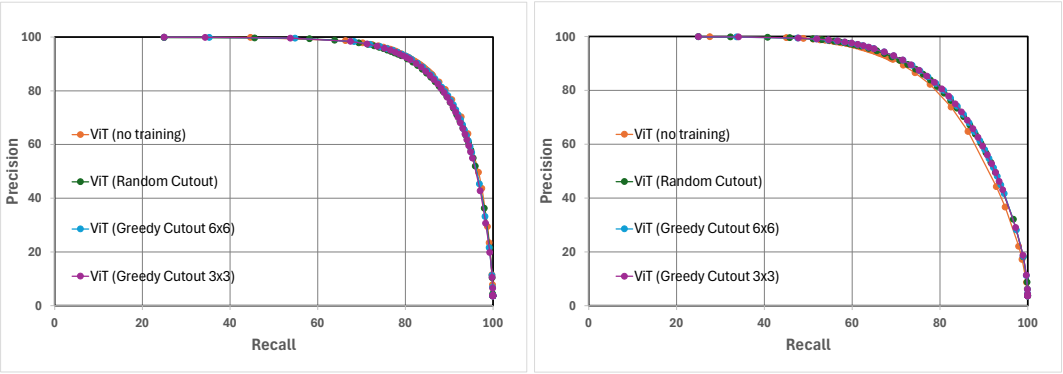
Table 4. *PatchDEMUX performance with ViT architecture on the MS-COCO 2014 validation dataset when using different defense fine-tuning techniques. Precision values are evaluated at key recall levels along with the approximated average precision. We assume the patch attack is at most* 2% *of the image area and use a computational budget of* 6×6 *masks.*

(a) *Clean setting precision values*

Architecture	ViT (vanilla/no fine-tuning)				ViT (Random Cutout)				ViT (Greedy Cutout 6×6)				ViT (Greedy Cutout 3×3)			
Recall	25%	50%	75%	AP	25%	50%	75%	AP	25%	50%	75%	AP	25%	50%	75%	AP
Undefended	99.940	99.749	96.265	91.449	99.770	99.642	95.951	90.900	99.930	99.704	96.141	91.146	99.930	99.736	95.973	90.903
Defended	99.930	99.138	85.757	83.776	99.858	99.224	87.273	85.028	99.894	99.223	87.764	85.276	99.900	99.230	87.741	85.271

(b) *Certified robust setting precision values*

Architecture	ViT (vanilla/no fine-tuning)				ViT (Random Cutout)				ViT (Greedy Cutout 6×6)				ViT (Greedy Cutout 3×3)			
Certified recall	25%	50%	75%	AP	25%	50%	75%	AP	25%	50%	75%	AP	25%	50%	75%	AP
Certified robust	90.767	38.490	20.846	35.003	94.192	47.548	24.603	40.975	95.369	51.580	22.662	41.763	95.574	51.095	23.454	42.077
Location-aware robust	91.665	43.736	23.163	38.001	94.642	52.491	27.526	43.908	95.670	56.038	26.375	44.902	95.959	55.958	27.105	45.122



(a) *Undefended clean precision-recall curves*

(b) *Defended clean precision-recall curves*

(c) *Certified robust precision-recall curves*

(d) *Location-aware robust precision-recall curves*

Figure 7. *PatchDEMUX precision-recall curves with ViT architecture over the MS-COCO 2014 validation dataset when using different defense fine-tuning techniques. We consider each of the four evaluation settings in separate plots. We assume the patch attack is at most* 2% *of the image area and use a computational budget of* 6×6 *masks.*

to performance boosts for PatchDEMUX. For instance, fine-tuning helps the two certified robust evaluation settings achieve a $6-7$ point improvement in certified AP compared to the vanilla checkpoints, while the defended clean setting demonstrates a $\sim$2 point improvement in clean AP compared to the baseline. Greedy Cutout also provides additional robustness boosts compared to Random Cutout, with certified AP metrics being almost a full point higher; this corroborates with findings from Saha et al. [28]. Note that in general defense fine-tuning strategies are less effective in the clean settings. This is likely because the clean settings already demonstrate (relatively) strong performance, and thus potential gains from fine-tuning are more marginal. Nevertheless, we prioritize the defended clean setting overall as it is most representative of typical performance.

The Greedy Cutout 6×6 fine-tuning strategy, which achieves the highest defended clean AP value, is therefore featured in Sec. 4.2.

Location-aware certification provides consistent improvements. An interesting observation from Tab. 4 is that the location-aware robust setting provides a consistent 3 point boost to certified AP regardless of the presence/absence of defense fine-tuning. This suggests that our location-aware certification technique has general utility across a variety of scenarios and that it "stacks" with other sources of robustness improvements.

G. Runtime Analysis of PatchDEMUX

Table 5. *Runtime experiments on PatchDEMUX. We report median per-sample inference time (in milliseconds) across a random sample of 2000 datapoints from the MS-COCO 2014 validation dataset. We assume the patch attack is at most 2% of the image area.*

Architecture	ViT (2×2 masks)	ViT (4×4 masks)	ViT (6×6 masks)
Undefended	31.130	31.130	31.130
Defended (single-label)	200.61	674.98	1451.1
Defended (multi-label)	317.30	1892.3	5668.7

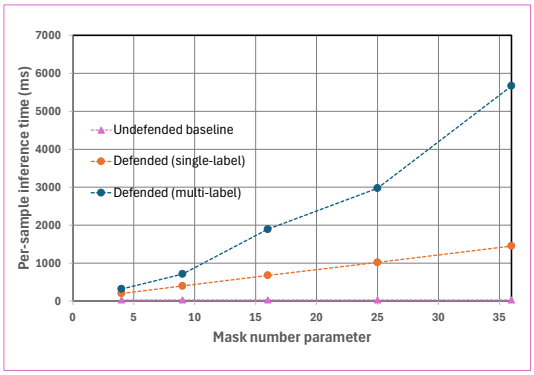


Figure 8. *Plot of PatchDEMUX runtime as a function of mask number. We report median per-sample inference time (in milliseconds) across a random sample of 2000 datapoints from the MS-COCO 2014 validation dataset. We assume the patch attack is at most 2% of the image area.*

In this section, we analyze the runtime of the PatchDEMUX inference procedure. To determine the impact of class number, we create a restricted version of our inference procedure that operates only on the first class (i.e., it ignores the remainder of the label $\mathbf{y}\in\{0,1\}^c$); this is essentially an instance of PatchCleanser isolated to a single class. We then track the runtime for 2000 random datapoints from the MS-COCO 2014 validation dataset. We use the ViT checkpoints from Sec. 4.2 and use a batch size of 1 to directly obtain per-sample inference time. The median per-sample inference times for different mask numbers are present in Tab. 5 and Fig. 8.

We note that for most mask numbers the full multi-label inference procedure takes roughly $3\times$ longer than the single-label implementation. Given that MS-COCO has $c=80$ classes, this is significantly faster than the expected runtime for the naive method from Algorithm 1. The reason for this improvement is an implementation-level optimization that takes advantage of relatively negligible defense post-processing. Specifically, the primary bottleneck for single-label inference procedures is often model query time; the associated defense post-processing is negligible in comparison. This means that for each feedforward through the multi-label classifier we can apply single-label defense post-processing to every class and re-use individual class outputs as needed for multi-label inference. As an example, with PatchCleanser this is done by saving intermediate outputs that correspond to double-masked images. Overall, this technique helps prevent computation cost from increasing drastically with the number of classes.

Despite this optimization, we note that the multi-label inference implementation is still not as fast as the single-label inference implementation. This is because many single-label inference procedures have worst-case scenarios which take significantly longer than typical cases. Increasing the number of classes increases the possibility that at least one class will trigger a worst-case scenario, leading to longer overall runtime.

H. Performance on PASCAL VOC

In this section we report evaluation results for PatchDEMUX on PASCAL VOC. Because model checkpoints for PASCAL VOC are not readily available, we first create a multi-label classifier for the PASCAL VOC task. To do so, we use model checkpoints pre-trained on the MS-COCO dataset and fine-tune it for the PASCAL VOC dataset. We use asymmetric loss (ASL) as the loss function, a 1cycle learning rate policy with max learning rate $\alpha_{max}=2.0\cdot10^{-3}$, automatic mixed precision (AMP) for faster training, and exponential moving average (EMA) of model checkpoints for improved inference [3]. Models are fine-tuned on the PASCAL VOC 2007 training split. We use the Adam optimizer for 15 epochs and select the best checkpoint according to average loss on the PASCAL VOC 2007 validation split. A cluster of NVIDIA A100 40GB GPUs are used to perform the fine-tuning. We omit additional security fine-tuning to focus on baseline performance.

We evaluate the fine-tuned model on the PASCAL VOC 2007 test dataset. We summarize the precision values associated with key recall levels in Tab. 6. Fig. 9 features precision-recall plots, while AP values are present in Tab. 6. We consider the ViT architecture alone as it provides better performance compared to Resnet.

Strong all-around performance. As shown in Tab. 6 and Fig. 9, PatchDEMUX achieves strong performance in all evaluation settings. In fact, PatchDEMUX's performance on PASCAL VOC is significantly higher than its performance on MS-COCO, with a ~7 point increase in defended clean performance and ~12 point increase in certified robustness metrics (see Sec. 4.2). Overall, these stronger results are expected given that the PASCAL VOC benchmark has fewer classes than MS-COCO, making it an easier benchmark for classifiers to predict.

Concave robustness curves. An interesting observation is that both of the PASCAL VOC robustness curves are concave

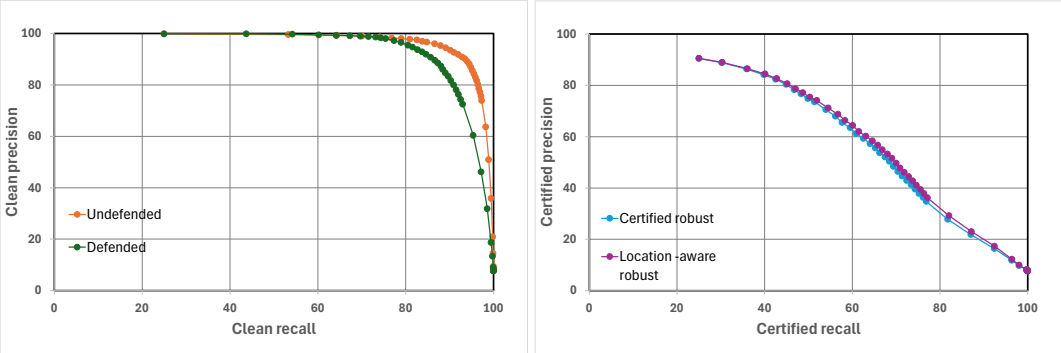
Table 6. *PatchDEMUX performance with ViT architecture on the PASCAL VOC 2007 validation dataset. Precision values are evaluated at key recall levels along with the approximated average precision. We assume the patch attack is at most 2% of the image area and use a computational budget of 6×6 masks.*

(a) *Clean setting precision values*

Architecture	ViT			
Clean recall	25%	50%	75%	*AP*
Undefended	99.790	99.710	98.506	96.140
Defended	99.894	99.870	98.167	92.593

(b) *Certified robust setting precision values*

Architecture	ViT			
Certified recall	25%	50%	75%	*AP*
Certified robust	90.520	74.675	38.100	54.904
Location-aware	90.591	75.672	40.320	56.030



(a) *Clean setting precision-recall curves* (b) *Certified robust setting precision-recall curves*

Figure 9. *PatchDEMUX precision-recall curves with ViT architecture over the PASCAL VOC 2007 test dataset. We consider the clean and certified robust evaluation settings. We assume the patch attack is at most 2% of the image area and use a computational budget of 6×6 masks.*

in Fig. 9b. This is in contrast to MS-COCO experiments, where even after applying security fine-tuning methods the robustness curves remained convex (see Fig. 7). An important takeaway from this is that PatchDEMUX performance is dataset dependent, and robustness bounds will ultimately depend on the nature of image datapoints and/or labels. Additionally, we note that location-aware certification only provides a ~ 1 point boost to certified AP; this suggests that location-aware certification is most beneficial when baseline robustness bounds are weak.

I. Tables for Security Parameter Experiments

In this section we provide the tables associated with the security parameter experiments in Sec. 5. In Tab. 7 we list metrics associated with the mask number experiments from Sec. 5.1. In Tab. 8 we list metrics associated with the patch size experiments from Sec. 5.2.

Table 7. *PatchDEMUX performance with ViT architecture on the MS-COCO 2014 validation dataset. We vary the mask number security parameter associated with the underlying single-label CDPA PatchCleanser and fix the estimated patch size at 2% of the image area. We list even mask number values for brevity. Precision values are evaluated at key recall levels along with the approximated average precision.*

(a) *Clean setting precision values*

Architecture	ViT (2×2 masks)				ViT (4×4 masks)				ViT (6×6 masks)			
Recall	25%	50%	75%	*AP*	25%	50%	75%	*AP*	25%	50%	75%	*AP*
Undefended	99.940	99.749	96.265	91.449	99.940	99.749	96.265	91.449	99.940	99.749	96.265	91.449
Defended	99.910	96.999	75.393	78.727	99.930	98.845	83.388	82.529	99.930	99.138	85.757	83.776

(b) *Certified robust setting precision values*

Architecture	ViT (2×2 masks)				ViT (4×4 masks)				ViT (6×6 masks)			
Certified recall	25%	50%	75%	*AP*	25%	50%	75%	*AP*	25%	50%	75%	*AP*
Certified robust	41.577	17.924	9.909	15.735	87.976	37.163	19.798	33.231	90.767	38.490	20.846	35.003
Location-aware robust	46.553	20.624	10.798	17.690	89.259	41.490	21.763	35.953	91.665	43.736	23.163	38.001

Table 8. *PatchDEMUX performance with ViT architecture on the MS-COCO 2014 validation dataset. We vary the patch size security parameter associated with the underlying single-label CDPA PatchCleanser and fix the mask number parameter at 6×6. Precision values are evaluated at key recall levels along with the approximated average precision.*

(a) *Clean setting precision values*

Architecture	ViT (0.5% patch)				ViT (2% patch)				ViT (8% patch)				ViT (32% patch)			
Recall	25%	50%	75%	*AP*	25%	50%	75%	*AP*	25%	50%	75%	*AP*	25%	50%	75%	*AP*
Undefended	99.940	99.749	96.265	91.449	99.940	99.749	96.265	91.449	99.940	99.749	96.265	91.449	99.940	99.749	96.265	91.449
Defended	99.947	99.470	89.150	85.731	99.930	99.138	85.757	83.776	99.907	97.798	78.712	80.093	99.529	89.813	60.543	69.952

(b) *Certified robust setting precision values*

Architecture	ViT (0.5% patch)				ViT (2% patch)				ViT (8% patch)				ViT (32% patch)			
Certified recall	25%	50%	75%	*AP*	25%	50%	75%	*AP*	25%	50%	75%	*AP*	25%	50%	75%	*AP*
Certified robust	97.670	61.867	30.239	48.820	90.767	38.490	20.846	35.003	44.666	19.249	11.832	16.961	6.933	5.827	4.854	5.297
Location-aware robust	97.769	66.350	32.850	51.158	91.665	43.736	23.163	38.001	50.263	22.965	13.363	19.713	9.169	6.997	5.307	6.195