Boosting Few-Pixel Robustness Verification via Covering Verification Designs

Yuval Shapira, Naor Wiesel, Shahar Shabelman, and Dana Drachsler-Cohen

Technion, Haifa, Israel
{shapirayuval@campus,wieselnaor@campus,shabelman@campus,ddana@ee}.technion.ac.il

Abstract. Proving local robustness is crucial to increase the reliability of neural networks. While many verifiers prove robustness in L_∞ ϵ-balls, very little work deals with robustness verification in L_0 ϵ-balls, capturing robustness to few pixel attacks. This verification introduces a combinatorial challenge, because the space of pixels to perturb is discrete and of exponential size. A previous work relies on covering designs to identify sets for defining L_∞ neighborhoods, which if proven robust imply that the L_0 ϵ-ball is robust. However, the number of neighborhoods to verify remains very high, leading to a high analysis time. We propose *covering verification designs*, a combinatorial design that tailors effective but analysis-incompatible coverings to L_0 robustness verification. The challenge is that computing a covering verification design introduces a high time and memory overhead, which is intensified in our setting, where multiple candidate coverings are required to identify how to reduce the overall analysis time. We introduce `CoVerD`, an L_0 robustness verifier that selects between different candidate coverings *without constructing them*, but by predicting their block size distribution. This prediction relies on a theorem providing closed-form expressions for the mean and variance of this distribution. `CoVerD` constructs the chosen covering verification design *on-the-fly*, while keeping the memory consumption minimal and enabling to parallelize the analysis. The experimental results show that `CoVerD` reduces the verification time on average by up to 5.1x compared to prior work and that it scales to larger L_0 ϵ-balls.

1 Introduction

Neural networks are very successful in various applications, most notably in image recognition tasks [14]. However, neural networks are also vulnerable to adversarial example attacks [33,17]. In an adversarial example attack, an attacker slightly perturbs the input to mislead the network. Many attack models and different kinds of perturbations have been considered for neural networks implementing image classifiers [33,15,26]. The most commonly studied perturbations are L_p perturbations, where p is 0 [9,40], 1 [10], 2 [33,4] or ∞ [15,4]. For L_p perturbations, the attacker is given a small budget ϵ and the goal is to find a perturbed input in the L_p ϵ-ball that causes misclassification.

In response to adversarial attacks, many verifiers have been proposed to reason about the robustness of neural networks in a given neighborhood of inputs. Most deterministic robustness verifiers analyze robustness in

L_∞ ϵ-balls [34,25,32,13,21,2], while some deterministic verifiers analyze L_2 ϵ-balls [22,19] or L_1 ϵ-balls [38,41]. Probabilistic verifiers, often leveraging randomized smoothing [6], have been proposed for analyzing L_p ϵ-balls for $p \in \{0, 1, 2, \infty\}$ [23,28,39,11]. Other verifiers analyze neighborhoods defined by semantic or geometric features (e.g., brightness or rotation) [20,24,3]. An existing gap is deterministically verifying robustness in L_0 ϵ-balls, for a small ϵ, also known as robustness to few pixel attacks. In L_0 ϵ-balls, ϵ is the number of pixels that can be perturbed. Since ϵ is an integer (as opposed to a real number), we denote it as t. L_0 t-balls consist of *discrete perturbations*, unlike many other attack models whose perturbations are continuous. Thus, their analysis is a challenging combinatorial problem. Theoretically, robustness verification of an L_0 t-ball can be reduced into a set of robustness verification tasks over L_∞ neighborhoods, each allows a specific set of t pixels to be perturbed. However, this approach is infeasible in practice for $t > 2$, since the number of the L_∞ neighborhoods that need to be proven robust is $\binom{v}{t}$, where v is the number of pixels. To illustrate, for MNIST images, where $v = 784$, the number of neighborhoods is $1.6 \cdot 10^{10}$ for $t = 4$, $2.4 \cdot 10^{12}$ for $t = 5$, and $3.2 \cdot 10^{14}$ for $t = 6$. That is, every *minimal* increase of t (by one) increases the neighborhood size by *two orders of magnitude*.

A recent work proposes a deterministic L_0 robustness verifier for few pixel attacks, called Calzone [30]. Calzone builds on two main observations. First, if a network is robust to perturbations of a *specific* set of k pixels, then it is also robust to perturbations of any subsumed set of these pixels. Second, often L_∞ robustness verifiers can analyze robustness to arbitrary perturbations of k *specific* pixels, for values of k that are significantly larger than t. They thus reduce the problem of verifying robustness in an L_0 t-ball to proving robustness in a set of L_∞ neighborhoods defined by a set of k-sized pixel sets, subsuming all possible sets of t pixels. To compute the k-sized pixel sets, they rely on *covering designs* [16,35]. Given parameters (v, k, t), a covering is a set of k-sized sets that cover all subsets of size t of a set $[v] = \{1, \dots, v\}$ (e.g., the pixel set). Covering designs is a field in combinatorics providing construction techniques to compute coverings. The challenge is to compute a covering of minimal size. While many covering constructions have been proposed, computing an optimal covering is an open combinatorial problem for most values of v, k and t. Further, most best-known coverings for $t > 3$ are far from the best general lower bound, known as the Schönheim bound [29]. This severely impacts the analysis time of Calzone. In practice, Calzone often does not complete within the five hour timeout when analyzing L_0 5-balls. To scale, it is crucial to lower the number of analyzed sets. While there are effective covering constructions renowned for the small coverings they compute, they are limited to specific values of v and k, which are incompatible for the analysis of L_0 robustness. Since Calzone treats covering constructions as black-box, it is limited to rely on analysis-compatible coverings and cannot benefit from these effective constructions.

To boost the robustness verification of few pixel attacks, we propose a new covering type, called a *covering verification design* (`CVD`), tailoring covering designs for L_0 robustness verification. `CVD` relies on a highly effective construction to obtain an analysis-incompatible covering and *partially induces* it to an analysis-compatible covering, where sets can have different sizes. Although the exact sets and their sizes depend on a random choice, we prove that the mean and variance of the set sizes are independent of this choice and have closed-form expressions. Partially inducing this effective construction has been proposed before [27], however it has been proposed for another combinatorial design, requiring a bound on the maximal set size in the covering, unlike CVD. We demonstrate that the sizes of CVDs are *lower* by 8% for $t = 4$ and by 15% for $t = 5$ than the respective Schönheim lower bound. This improvement, enabled by considering a new type of coverings, is remarkable for scaling L_0 robustness analysis. To date, for analysis-compatible values of v and k and for $t \geq 3$, it is impossible to obtain an optimal covering design, and even if we obtained it, its size is *at least* the Schönheim bound. In particular, Calzone's considered coverings are larger by 4x than the Schönheim lower bound for $t = 4$ and by 8.4x for $t = 5$. While promising, CVDs raise a practical challenge: their construction as well as their final size introduce a high memory overhead. Further, to minimize the analysis time, the verifier chooses between *multiple* coverings. However, the total memory overhead makes it infeasible to store these coverings in a covering database without limiting their size (like Calzone does).

We introduce CoVerD, an L_0 robustness verifier, boosting Calzone's performance by leveraging CVDs. CoVerD has two main components, *planning* and *analysis*. The planning component predicts the CVD that will allow it to minimize the overall analysis time. To reduce the memory overhead, it predicts the best CVD out of many candidates, *without constructing the candidates*. This prediction relies on estimating the set size distribution of a candidate covering, using our expressions for the mean and variance. The analysis component constructs the chosen CVD. The challenge is that the original covering that is being induced may be too large to fit the memory. To cope, CoVerD induces the covering while constructing the original covering. Further, it constructs *on-the-fly* a partitioning of the CVD so that the analysis can be parallelized over multiple GPUs. Another advantage of the on-the-fly construction is that CoVerD does not need to prepare coverings for every image dimension in advance. This both saves memory consumption and makes CoVerD suitable for *any* image classifier, without requiring to precompute coverings for new image dimensions, as Calzone requires.

We evaluate CoVerD on convolutional and fully-connected networks, trained for MNIST, Fashion-MNIST, and CIFAR-10. CoVerD is faster than Calzone in verifying robust t-balls on average by 2.8x for $t = 4$ and by 5.1x for $t = 5$. Further, CoVerD scales to more challenging t-balls than Calzone. In particular, it verifies some 6-balls, which Calzone does not consider at all, within 42 minutes.

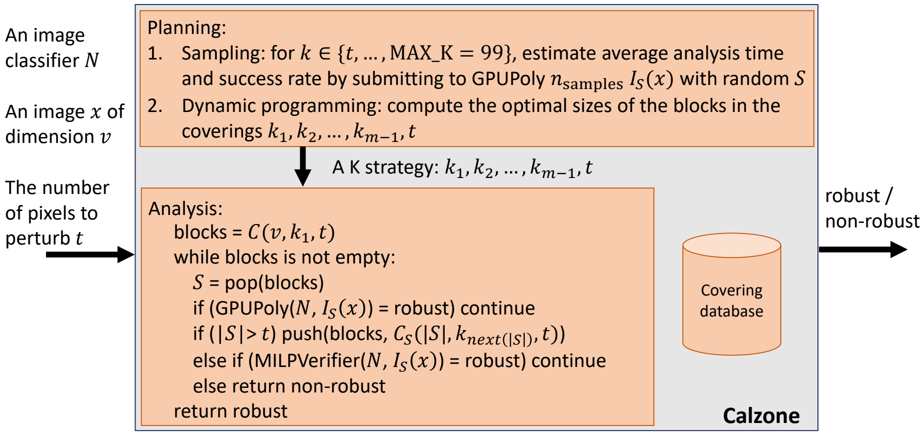
2 Background

In this section, we define the problem of verifying robustness of an image classifier in an L_0 t-ball and provide background on Calzone [30].

L_0 robustness verification We address the problem of determining the local robustness of an image classifier in an L_0 t-ball of an image x. An image classifier N takes as input an image x consisting of v pixels, each ranges over $[0,1]$ (all definitions extend to colored images, but omitted for simplicity's sake). It returns a vector consisting of a score for every possible class. The classification the classifier N assigns to an input image x is the class with the maximal score: $c_x = \texttt{argmax}(N(x))$. We focus on classifiers implemented by neural networks. Specifically, our focus is on fully-connected and convolutional networks, since many L_∞ robustness verifiers can analyze them [32,13,21,2,34,25]. However, like Calzone, `CoVerD` is not coupled to the underlying implementation of the classifier and can reason about any classifier for which there are L_∞ robustness verifiers that it can rely on. The problem we study is determining whether a classifier N is locally robust in the L_0 t-ball of an input x, for $t \geq 2$. That is, whether every input whose L_0 distance from x is at most t is classified by N as x is classified. Formally, the t-ball of x is $B_t(x) = \{x' \mid ||x' - x||_0 \leq t\}$ and N is locally robust in $B_t(x)$ if $\forall x' \in B_t(x).\ \texttt{argmax}(N(x')) = \texttt{argmax}(N(x))$. We note that the L_0 distance of two images is the number of pixels that the images differ, that is $||x' - x||_0 = |\{i \in [v] \mid x_i \neq x'_i\}|$ (where $[v] = \{1, \ldots, v\}$). In other words, an L_0 perturbation to an image x can arbitrarily perturb up to t pixels in x.

Calzone Calzone, depicted in Figure 1, is an L_0 robustness verifier. It verifies by determining the robustness of a classifier N in all neighborhoods in which a specific set of pixels S is arbitrarily perturbed, for every $S \subseteq [v]$ of size t. Namely, to prove robustness, it has to determine for every such S whether N classifies the same all inputs in the neighborhood consisting of all images that are identical to x in all pixels, but the pixels in S. We denote this neighborhood by $I_S(x) = \{x' \in [0,1]^v \mid \forall i \notin S.\ x'_i = x_i\}$. Such neighborhoods can be specified as a sequence of intervals, one for every pixel, where the i^{th} interval is $[0,1]$ if $i \in S$ (i.e., it can be perturbed) or $[x_i, x_i]$ if $i \notin S$ (i.e., it cannot be perturbed). Most existing L_∞ robustness verifiers can determine the robustness of such interval neighborhoods. However, verifying $\binom{v}{t}$ interval neighborhoods, one for every selection of t pixels to perturb, is practically infeasible for $t > 2$. Instead, Calzone builds on the following observation: if N is locally robust in a neighborhood $I_{S'}(x)$ for $S' \subseteq [v]$ of size $k > t$, then N is also robust in every $I_S(x)$, for $S \subseteq S'$ of size t. This observation allows Calzone to leverage *covering designs* to reduce the number of neighborhoods analyzed by an L_∞ verifier. Given three numbers (v, k, t), for $t \leq k \leq v$, a covering $C(v, k, t)$ is a set of *blocks*, where (1) each block is subset of size k of $[v]$ and (2) the blocks cover all subsets of $[v]$ of size t: for every $S \subseteq [v]$ of size t, there is a block $B \in C(v, k, t)$ such that $S \subseteq B$. Coverings are evaluated by their size, $|C(v, k, t)|$, where the smaller the better. We next describe the components of Calzone: analysis, planning and covering database.

Calzone's analysis Calzone begins the analysis by obtaining a covering $C(v, k_1, t)$ from its covering database, where k_1 is determined by the planning component (described shortly). It pushes all blocks in the covering into a stack. It then iteratively pops a block S from the stack and verifies the robustness of N in


An image classifier N
An image x of dimension v
The number of pixels to perturb t
A K strategy: $k_1, k_2, \ldots, k_{m-1}, t$
Covering database
Calzone
robust / non-robust

Planning:
1. Sampling: for $k \in \{t, \ldots, \text{MAX\_K} = 99\}$, estimate average analysis time and success rate by submitting to GPUPoly n_{samples} $I_S(x)$ with random S
2. Dynamic programming: compute the optimal sizes of the blocks in the coverings $k_1, k_2, \ldots, k_{m-1}, t$

```
Analysis:
  blocks = $C(v, k_1, t)$
  while blocks is not empty:
    $S$ = pop(blocks)
    if (GPUPoly($N$, $I_S(x)$) = robust) continue
    if ($|S| > t$) push(blocks, $C_S(|S|, k_{next(|S|)}, t)$)
    else if (MILPVerifier($N$, $I_S(x)$) = robust) continue
    else return non-robust
  return robust
```

Fig. 1: The Calzone L_0 robustness verifier.

$I_S(x)$ by running GPUPoly [25]. GPUPoly is a sound L_∞ robustness verifier which is highly scalable because it performs the analysis on a GPU. However, it relies on a linear relaxation and thus may fail proving robustness due to overapproximation errors. If it determines that $I_S(x)$ is robust, Calzone continues to the next block. Otherwise, Calzone performs an exact analysis or refines the block. If $|S| = t$, Calzone invokes a sound and complete mixed-integer linear programming (MILP) verifier [34]. If it determines that $I_S(x)$ is not robust, Calzone returns *non-robust*, otherwise Calzone continues to the next block. If $|S|$ is greater than t, Calzone refines S by pushing to the stack all blocks in a covering for S and t. The blocks' size is k_{i+1}, which is the block size following the current block size $k_i = |S|$, as determined by the planning component. The covering is obtained by retrieving from the covering database the covering $C(|S|, k_{i+1}, t)$ and renaming the numbers in the blocks to range over the numbers in S (instead of $[|S|]$), denoted as $C_S(|S|, k_{i+1}, t)$. If Calzone observes an empty stack, it returns *robust*. This analysis is proven sound and complete. To scale, Calzone parallelizes the analysis over multiple GPUs (for GPUPoly) and CPUs (for the MILP verifier). Technically, the first covering is split between the GPUs, each independently analyzes its assigned blocks and refines if needed.

Calzone's planning The planning determines the block sizes of the first covering and of the refinements' coverings. These are given as a *K strategy*, a decreasing series $k_1 > \ldots > k_m$, where $k_1 \leq \text{MAX\_K} = 99$ and $k_m = t$. Calzone predicts the K strategy that minimizes the overall analysis time using dynamic programming, defined over the analysis time of the first covering, the average fraction of blocks that will be refined, and the analysis time of the refined blocks. This computation requires GPUPoly's success rate and average analysis time for neighborhoods $I_S(x)$, for all $|S| \leq \text{MAX\_K}$. These are estimated by sampling $n_{\text{samples}} = 400$ sets S for every $k \leq \text{MAX\_K}$ and submitting their neighborhood $I_S(x)$ to GPUPoly.

Calzone's covering database As described, the analysis obtains coverings from a database. This database has been populated by obtaining well-optimized coverings from online resources and extending them for large values of v and k using general covering constructions. Because of these general constructions, the database's coverings tend to be far from the Schönheim bound [29], the best-known general lower bound, especially for large values of v (the image dimension). This inefficiency results in longer analysis, since more blocks are analyzed.

3 Our Approach: Covering Verification Designs

To scale Calzone's analysis, it is crucial to reduce the number of blocks that are analyzed by GPUPoly or the MILP verifier. A dominant contributor to this number is the size of the first covering, for two reasons. First, the first covering is over a large v (the image dimension), thus its size is significantly larger than the sizes of coverings added upon refinement, which are over significantly smaller v (typically $v \leq 80$ and at most $v \leq$ MAX_K). Second, the first covering has an accumulative effect on the number of refinements, and consequently it dominates the analysis time. Reducing this size is theoretically possible by relying on *finite geometry covering constructions* [27,1,16], which are renowned for computing very small coverings. However, finite geometry coverings are limited to (v, k, t) triples in which v and k are defined by related mathematical expressions over t. In Calzone's analysis, the first covering has to be defined over a given v (the image dimension) and t (the number of perturbed pixels). Thus, for some values of v and t, there is no finite geometry covering. For the other values, there are very few values for k, leading to long analysis either because they are large and have a low success rate, triggering many refinements, or small and have very large coverings. We propose to tailor *induced coverings* for L_0 robustness analysis in order to leverage finite geometry coverings. To this end, we introduce a new type of a covering design, called a *covering verification design* (`CVD`). We next provide background on finite geometry coverings and induced coverings. We then define *partially-induced coverings* and our new covering type. We discuss its properties, its effectiveness, and the practical challenges in integrating it to L_0 verification.

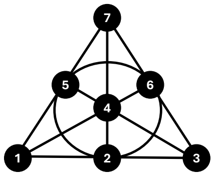


Fig. 2: The Fano Plane.

Finite geometry coverings Finite geometry covering constructions are widely known for obtaining small (sometimes optimal) coverings [27,1,16]. Popular finite geometry constructions rely on projective geometry (PG) or affine geometry (AG). We focus on PG, but our approach extends to AG. A PG construction views the problem of constructing a covering for a given (v, k, t) from a finite geometry point of view, where v is the number of points in the geometry. It constructs coverings by computing flats (linear subspaces) of dimension $t-1$, each containing k points. Since every t points from $[v]$ are contained in at least one flat [27], the flats provide a covering. Figure 2 shows *the Fano plane*, a well-known example. In this example, there are $v = 7$ points, the flats are of dimension $t - 1 = 1$ (the lines and the circle), each containing $k = 3$ points. The set of flats forms a covering, where each flat is a block: $C(7,3,2) = \{\{1,2,3\},\{1,4,6\},\{1,5,7\},\{2,4,7\},\{2,5,6\},\{3,4,5\},\{3,6,7\}\}$. PG coverings exist for triples where $v = \frac{q^{m+1}-1}{q-1}$ and $k = \frac{q^t-1}{q-1}$, for a prime power q and $m \geq t \geq 2$ (it also exists for $m = t-1$, but then $v = k$, which is unhelpful to our analysis). Because PG is restricted to such triples, Calzone cannot effectively leverage it for the first covering, whose v and t are given. This is because for common image dimensions (e.g., $v = 784$ for MNIST and $v = 1024$ for CIFAR-10), there are no suitable q and m. Even if there are suitable q and m, there are very few possible k values, which are unlikely to include or be close to an optimal value of k. Thus, either they are smaller than an optimal k, leading to larger coverings and a longer analysis time, or that they are larger than an optimal k and have a lower success rate, leading to many refinements, resulting, again, in a longer analysis time. For example, for $v = 364$ and $t = 5$, the only suitable values are $q = 3$ and $m = 5$ (i.e., $364 = (3^{5+1} - 1)/(3 - 1)$), namely there is only one triple for these values of v and t. In this triple, $k = (3^5 - 1)/(3 - 1) = 121$. Since $k \approx \frac{v}{3}$, neighborhoods $I_S(x)$ for which $|S| = 121$ are not likely to be robust, thus such k is likely to have a low success rate. *Induced coverings* [16] enable to leverage finite geometry coverings for other (v, k, t) triples, as next explained.

Induced coverings Given $v \leq v'$ and $k \leq k'$, a covering $C(v', k', t)$ can be *induced* to form a covering $C(v, k, t)$ [16]. The induced covering is obtained in three steps. First, we select a subset of numbers of size v, denoted $L \subseteq [v']$, and remove every $l \in [v'] \setminus L$ from every block in $C(v', k', t')$. This results in a set of blocks of different sizes that covers all subsets of L of size t [27, Lemma 1]. This follows since every subset $S \subseteq L$ of size t is a subset of $[v']$ and thus there is $B \in C(v', k', t)$ such that $S \subseteq B$. The first step removes from B only numbers from $[v'] \setminus L$ and thus S is contained in the respective block to B after this step. The next two steps fix blocks whose size is not k. The second step extends every block whose size is smaller than k with numbers from L. The third step refines every block whose size is larger than k to multiple blocks of size k that cover all of its subsets of size t. This step significantly increases the number of blocks, unless the number of blocks larger than k is negligible. We note that these steps provide a covering over the numbers in L (i.e., $C_L(|L|, k, t)$). A covering for $(|L|, k, t)$ can be obtained by renaming the numbers to range over $[|L|]$.

Partially-induced covering Our new covering design is an instance of a *partially-induced covering*. A partially-induced covering is the set of blocks obtained by the first step, where the blocks cover all subsets of L of size t and are of different sizes. For example, for the Fano plane and $L_1 = \{4,5,6,7\}$, the partially-induced covering is: $C_1 = \{\{\},\{4,6\},\{5,7\},\{4,7\},\{5,6\},\{4,5\},\{6,7\}\}$, while for $L_2 = \{1,2,3,4\}$, it is: $C_2 = \{\{1,2,3\},\{1,4\},\{1\},\{2,4\},\{2\},\{3,4\},\{3\}\}$. Partially-induced coverings have two benefits in our setting: (1) by not extending blocks whose size is smaller than k, we increase the likelihood that GPUPoly will prove their robustness, and (2) by not refining blocks whose size is larger than k, we (a) preserve the number of blocks as in the original covering, (b) provide GPUPoly an opportunity to verify these blocks, and (c) rely on the optimal refinement sizes (computed by the dynamic programming) for blocks that GPUPoly fails proving robustness. Our covering design partially induces PG coverings, to obtain additional benefits for L_0 robustness verification.

Covering verification designs Given the number of pixels v and the number of pixels to perturb t, a covering verification design (`CVD`) is the set of blocks obtained by partially inducing a PG covering $C(v', k', t)$, where $v \leq v'$, using a random set of numbers $L \subseteq [v']$ of size v. The numbers in the blocks can later be renamed to range over $[v]$. For example, since the Fano plane is a PG covering, the partially-induced coverings C_1 and C_2 are `CVDs`. A `CVD` has two important properties. First, it is a partially-induced covering and thus has all the aforementioned advantages in our setting. In particular, its size is equal to the size of the original covering, which is highly beneficial since `CVD` induces from PG coverings, known for their small size. Second, although different sets L lead to different block size distributions, we prove that the mean block size and its variance are *the same* regardless of the choice of L. Further, we identify closed-form expressions for them and show that the variance is bounded by the mean. For example, although the block size distributions of C_1 and C_2 are different, they have the same average block size ($\frac{12}{7}$) and the same variance ($\frac{24}{49}$). This property has practical advantages: (1) it allows us to estimate the block size distribution (Section 4.2), and (2) since the variance is bounded by the mean, the smaller the mean block size, the less likely that there are very large blocks, which are less likely to be proven robust by GPUPoly. To prove this property, we rely on the fact that PG coverings (and AG coverings) are also a combinatorial design called a *balanced incomplete block design* (BIBD) [7, Part VII, Proposition 2.36]. We next describe BIBD and then state our theorem on its mean and variance.

BIBD Given positive integers (v, b, r, k, λ), a BIBD is a set of b blocks, each is a subset of $[v]$ of size k, such that every $i \in [v]$ appears in r blocks and every $i \neq j \in [v]$ appear together in λ blocks. For example, the Fano plane is a BIBD with $v = 7, b = 7, r = 3, k = 3, \lambda = 1$. This is because it has $b = 7$ blocks, each block is a subset of $[v] = \{1, \dots, 7\}$ of size $k = 3$, every number in $\{1, \dots, 7\}$ appears in $r = 3$ blocks and every two different numbers appear together in $\lambda = 1$ block. Given a BIBD with parameters (v', b, r, k', λ), we define a partially-induced BIBD for $v \leq v'$ by selecting a subset of numbers $L \subseteq [v']$ of size v and removing every $l \in [v'] \setminus L$ from every block in the BIBD (empty blocks or repetitive blocks are kept). While the distribution of the induced blocks' sizes depends on L, the mean block size and its variance depend only on v, v', k'.

Theorem 1. *Given a (v', b, r, k', λ)-BIBD, for $v' > 1$, and $1 \leq v \leq v'$, for every $L \subseteq [v']$ of size v, the mean $\mu_{v',k',v}$ and variance $\sigma^2_{v',k',v}$ of the block sizes in the partially-induced BIBD satisfy:*

1. $\mu_{v',k',v} = \frac{vk'}{v'}$
2. $\sigma^2_{v',k',v} = \mu_{v',k',v}\left(1 + \frac{(v-1)(k'-1)}{v'-1} - \mu_{v',k',v}\right) = \frac{vk'}{v'}\left(1 + \frac{(v-1)(k'-1)}{v'-1} - \frac{vk'}{v'}\right)$

3. $\sigma^2_{v',k',v} \leq \mu_{v',k',v}$

Proof. 1. We prove $\mu_{v',k',v} = \frac{vk'}{v'}$. Since $|L| = v$ and r is the number of occurrences of every number in all blocks, the sum of the sizes of the induced blocks is vr. By counting arguments, for a BIBD it holds that $rv' = bk'$ [7, Part II, Proposition 1.2], and so $r = \frac{bk'}{v'}$. That is, the sum of the induced blocks' sizes is $\frac{vbk'}{v'}$. The mean is obtained by dividing by the number of blocks b: $\mu_{v',k',v} = \frac{vk'}{v'}$.

2. We prove $\sigma^2_{v',k',v} = \mu_{v',k',v}\left(1 + \frac{(v-1)(k'-1)}{v'-1} - \mu_{v',k',v}\right)$.

Let $Z \in {\mathbb{N}_0}^b$ be a vector such that, for every $n \in [b]$, Z_n is the size of block n in the partially-induced BIBD. It can be written as $Z = A^T x_L$, where A represents the BIBD and x_L the set L, used for partially inducing the BIBD. The matrix A is a $v' \times b$ incidence matrix, where $A[m,n] = 1$ if m is in block n and $A[m,n] = 0$ otherwise. The vector x_L is a v'-dimensional vector, where $x_L[m] = 1$ if $m \in L$ and $x_L[m] = 0$ otherwise. Thus, the average of the squares of the block sizes, denoted $\mathbb{E}[Z^2]$, is $\mathbb{E}[Z^2] = \frac{1}{b}\left(\sum_{n=1}^{b}(A^T x_L)^2_n\right) = \frac{1}{b}\|A^T x_L\|_2^2$ (1).

By the variance definition, $\sigma^2_{v',k',v} = \mathbb{E}[Z^2] - \mu^2_{v',k',v}$. Thus, we need to show: $\mathbb{E}[Z^2] = \mu_{v',k',v}(1 + \frac{(v-1)(k'-1)}{v'-1}) = \frac{vk'}{v'}(1 + \frac{(v-1)(k'-1)}{v'-1}) = \frac{k'}{v'}v + \frac{k'(k'-1)}{v'(v'-1)}v(v-1)$. By counting arguments [7], we have $\frac{k'}{v'} = \frac{r}{b}$ and $\frac{k'(k'-1)}{v'(v'-1)} = \frac{\lambda}{b}$. Namely, it suffices to show: $\mathbb{E}[Z^2] = \frac{1}{b}\left(rv + \lambda v(v-1)\right)$. By (1), we can show $\|A^T x_L\|_2^2 = rv + \lambda v(v-1)$. We prove by induction on $v = |L|$ that $\|A^T x_L\|_2^2 = rv + \lambda v(v-1)$:

Base For $v = 1$, we show $\|A^T x_L\|_2^2 = r \cdot 1 + \lambda \cdot 1 \cdot 0$: Since $v = |L| = 1$, by definition of a BIBD, the vector of the induced blocks' sizes Z has r ones and the rest are zeros. Thus, $\|Z\|_2^2 = r$. Since $Z = A^T x_L$, the claim follows.

Induction hypothesis Assume that the claim holds for every $1, \ldots, v$.

Step Let $L \subseteq [v']$ such that $|L| = v+1$. Pick some $i \in L$ and define $L' = L \setminus \{i\}$ of size v. We get $x_L = x_{L'} + e_i$, where e_i is the i^{th} standard unit vector. Thus:

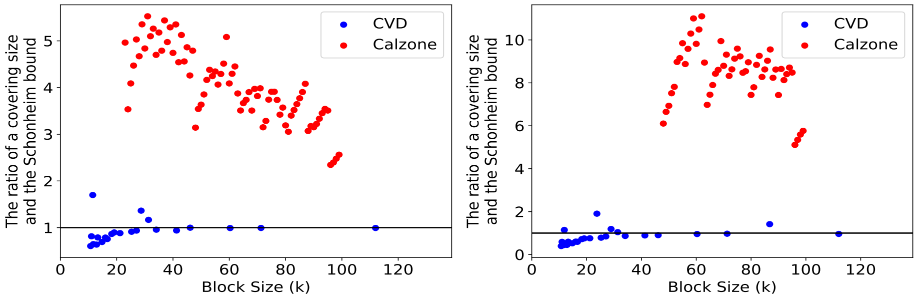
$$\|A^T x_L\|_2^2 = \|A^T(x_{L'} + e_i)\|_2^2 = \|A^T x_{L'}\|_2^2 + \|A^T e_i\|_2^2 + 2\left\langle A^T x_{L'}, A^T e_i\right\rangle$$

- By the induction hypothesis, $\|A^T x_{L'}\|_2^2 = rv + \lambda v(v-1)$.
- Since e_i can be viewed as $x_{L''}$ for some L'' of size 1, we get $\|A^T e_i\|_2^2 = r$.
- We show $\left\langle A^T x_{L'}, A^T e_i\right\rangle = x_{L'}^T\left(AA^T\right)e_i = \lambda v$: Since A is an incidence matrix of a BIBD, AA^T is the matrix with r on the diagonal and λ elsewhere [7, Part II, Theorem 1.8]. Therefore, $\left(AA^T\right)e_i$ is a vector whose entries are λ except for the i^{th} entry which is r. The vector $x_{L'}$ has v ones and 0 on the i^{th} entry (since $i \notin L'$). Thus, their dot product is $x_{L'}^T\left(AA^T\right)e_i = \lambda v$.

Putting it all together: $\|A^T x_L\|_2^2 = rv + \lambda v(v-1) + r + 2\lambda v = r(v+1) + \lambda(v+1)v$.

3. We show $\sigma^2_{v',k',v} \leq \mu_{v',k',v}$ by showing that $1 + \frac{(v-1)(k'-1)}{v'-1} - \mu_{v',k',v} \leq 1$. Since $\mu_{v',k',v} = \frac{vk'}{v'}$, we show $\frac{(v-1)(k'-1)}{v'-1} \leq \frac{vk'}{v'}$. We have $1 \leq v \leq v'$ and $1 < v'$, thus we get $\frac{v-1}{v'-1} \leq \frac{v}{v'}$. Since $k'-1 \geq 0$, we get $\frac{(v-1)(k'-1)}{v'-1} \leq \frac{v(k'-1)}{v'} \leq \frac{vk'}{v'}$. □

Size of covering verification designs `CVDs` enable us to obtain coverings whose sizes are small, often close or better than their respective Schönheim bound.



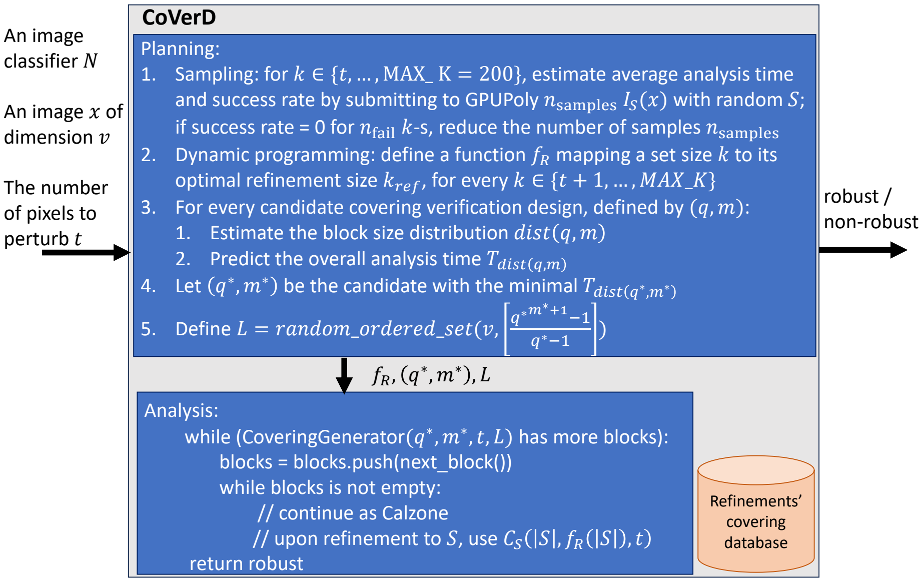
(a) The plot for $t = 4$.

(b) The plot for $t = 5$.

Fig. 3: The ratio of CVD sizes and their respective Schönheim bound vs. the ratio of Calzone's covering sizes and their Schönheim bound. The black line is ratio 1, i.e., coverings whose sizes are equal to the respective Schönheim bound.

Given a CVD whose mean block size is a real number k, we define its respective Schönheim bound as the bound for the covering design of $(v, \lceil k \rceil, t)$. Note that this bound is not a lower bound on the size of the CVD, since the CVD can have blocks larger than $\lceil k \rceil$ and thereby be smaller than covering designs for $(v, \lceil k \rceil, t)$. Still, comparing to this bound enables understanding how much smaller our coverings are compared to the coverings considered by Calzone, whose sizes are lower bounded by the Schönheim bound. Figure 3 shows the ratio of the sizes of our CVDs and their respective Schönheim bound and the ratio of the sizes of Calzone's covering designs and their Schönheim bound. The comparison is for $v = 784$ and $t = 4$ (Figure 3a) and $t = 5$ (Figure 3b). We compute CVDs from different PG coverings and the figure shows CVDs whose mean block size k is at least 10. For Calzone, we show all coverings in its database. The plots demonstrate that typically the size of a CVD is smaller or equal to the Schönheim bound, and on average, the ratio is 0.92 for $t = 4$ and 0.85 for $t = 5$. In contrast, Calzone's coverings are significantly larger than the Schönheim bound, on average the ratio is 4.04 for $t = 4$ and 8.44 for $t = 5$. The plots also show that Calzone has many more coverings than the number of CVDs. This is because Calzone relies on general techniques to compute coverings and thus it can generate a covering for every $k \leq \text{MAX\_K} = 99$ (except that it is limited to coverings with at most 10^7 blocks). In contrast, our CVDs induce PG coverings and are thus limited to coverings whose mean block size is given by the expression given in Theorem 1, over v' and k' such that there is a PG covering for (v', k', t).

Challenge: memory consumption The main challenge in computing CVDs is that it requires to compute a PG covering for large values of v' and k', which poses a high memory overhead. To illustrate, in our experiments, CoVerD uses a CVD induced from a PG covering for $(v' = 1508598, k' = 88741, t = 5)$. If CoVerD stored this covering in the memory, it would require 124GB of memory, assuming each number in a block takes a byte. To cope, CoVerD computes the


CoVerD
An image classifier N
An image x of dimension v
The number of pixels to perturb t
robust / non-robust
$f_R, (q^*, m^*), L$
Refinements' covering database

```
Planning:
1. Sampling: for $k \in \{t, \dots, \mathrm{MAX\_K} = 200\}$, estimate average analysis time
   and success rate by submitting to GPUPoly $n_{\mathrm{samples}}$ $I_S(x)$ with random $S$;
   if success rate = 0 for $n_{\mathrm{fail}}$ $k$-s, reduce the number of samples $n_{\mathrm{samples}}$
2. Dynamic programming: define a function $f_R$ mapping a set size $k$ to its
   optimal refinement size $k_{ref}$, for every $k \in \{t+1, \dots, MAX\_K\}$
3. For every candidate covering verification design, defined by $(q, m)$:
   1. Estimate the block size distribution $dist(q, m)$
   2. Predict the overall analysis time $T_{dist(q,m)}$
4. Let $(q^*, m^*)$ be the candidate with the minimal $T_{dist(q^*,m^*)}$
5. Define $L = random\_ordered\_set(v, \left[\frac{{q^*}^{m^*+1}-1}{q^*-1}\right])$

Analysis:
    while (CoveringGenerator($q^*, m^*, t, L$) has more blocks):
        blocks = blocks.push(next_block())
        while blocks is not empty:
            // continue as Calzone
            // upon refinement to $S$, use $C_S(|S|, f_R(|S|), t)$
    return robust
```

Fig. 4: `CoVerD`: An L_0 robustness verifier.

partially-induced covering during the PG covering construction. However, even the partially-induced coverings can consume a lot of memory, since the number of blocks can be large. Calzone faced a similar challenge and coped by restricting the size of the covering designs to at most 10^7, which allowed it to keep all coverings in the covering database. While `CoVerD` could take a similar approach, this would prevent it from picking `CVDs` of larger size which overall may lead to a lower analysis time (since they will require fewer refinements). Instead, `CoVerD` generates a `CVD` *on-the-fly* and uses the covering database only for the refinements, which tend to require coverings of significantly smaller size than the first covering. Another advantage of building the `CVD` on-the-fly is that it enables `CoVerD` to analyze any classifier over any image dimension, without any special adaptation. This is in contrast to Calzone, which requires to extend its covering database upon every new image dimension v.

4 CoVerD

In this section, we present `CoVerD`, our L_0 robustness verifier. We first describe our system and its components and then provide a running example.

4.1 Our System

Figure 4 shows `CoVerD` that, given an image classifier N, an image x with v pixels, and the maximal number of perturbed pixels t, returns whether N is robust in the t-ball of x. We next describe its planning and analysis components.

Planning The planning component consists of several steps. First, it samples sets of different sizes k to estimate the success rate and average analysis time of their respective neighborhoods, like Calzone. Since `CoVerD` considers `CVDs`, it can observe larger block sizes than Calzone, thus the maximal sampled set size is MAX_K $= 200$, unlike 99 in Calzone. Because of the larger bound, `CoVerD` is likely to observe many more k values whose success rate is zero. To save execution time while still enabling to determine the success rate and average analysis time of large k values, `CoVerD` reduces the number of samples after observing n_{fail} times k values whose success rate is zero. Second, the planning component relies on Calzone's dynamic programming for computing a K strategy, but uses it differently. Since `CoVerD` begins the analysis from a `CVD` consisting of different sized blocks, there is no single K strategy. Instead, it runs Calzone's dynamic programming for every $k \in \{t+1, \ldots, \text{MAX\_K}\}$ to define a function f_R mapping every set size k to the best set size to use upon a refinement of a set of size k. Then, the planning component iterates over every candidate `CVD` and picks the best `CVD` for the analysis. It picks between the candidates *without* constructing them, as the construction is time and memory intensive and we wish to execute it only for the chosen candidate. To pick the best candidate, it leverages two observations. First, a `CVD` candidate is uniquely defined by the parameters of the PG covering, (q, m) (formally, its parameters are (q, m, t) but t is identical in all our PG coverings), so it suffices to pick a pair (q, m) which can later be used to construct the `CVD`. Second, to predict the `CVD` with the minimal analysis time, only the *block sizes* are needed. In Section 4.2, we describe how to estimate a `CVD`'s block size distribution $dist(q, m)$ and estimate its analysis time $T_{dist(q,m)}$, in order to predict the best `CVD`. Given the best candidate (q^*, m^*), it randomly samples an ordered set L of v indices from v', which is a function of (q^*, m^*).

Analysis After determining the best (q^*, m^*), L, and the refinement mapping f_R, `CoVerD` continues to analyze the robustness of the classifier N in the t-ball of the given image x. The analysis constructs the `CVD` on-the-fly block-by-block. Technically, there is a covering generator that constructs the blocks one-by-one. Every block is pushed to the stack of blocks to verify, and then the analysis proceeds as Calzone. That is, the block is popped, submitted to GPUPoly, and if needed, refinement is executed. After the block is verified (directly or by refinement), the next block in the `CVD` is obtained from the covering generator. We note that although `CoVerD` could use `CVDs` for refinements, the coverings for refinements are smaller than the first covering since these coverings are for triples $(\tilde{v}, \tilde{k}, t)$ where $\tilde{v}$ is typically few dozens and at most MAX_K $= 200$, whereas the first covering is for a triple $(\tilde{v}, \tilde{k}, t)$ where $\tilde{v}$ is the image dimension. Like Calzone, `CoVerD` parallelizes the analysis on GPUs. Thus, our covering generator generates disjoint parts of the covering, described in Section 4.3.

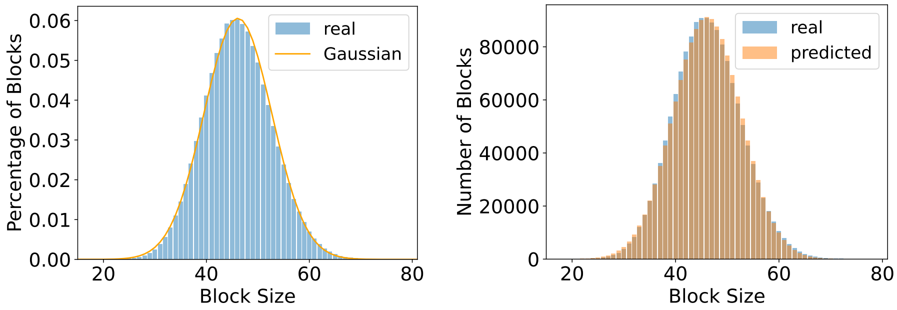
4.2 Choosing a Covering Verification Design

In this section, we describe how `CoVerD` predicts the `CVD` that enables `CoVerD` to minimize the overall analysis time. We begin with describing the `CVD` candidates, then describe how CoVerD estimates their block size distributions, and finally explain how CoVerD predicts the CVD leading to the minimal analysis time.

Candidates A CVD candidate is defined by the PG covering from which it is partially-induced. Recall that a PG covering is defined for triples (v', k', t), where $v' = \frac{q^{m+1}-1}{q-1}$ and $k' = \frac{q^t-1}{q-1}$ for a prime power q and $m \geq t \geq 2$. By Theorem 1, given a PG covering, the mean block size of the CVD has a closed-form expression $\mu_{v',k',v} = \frac{vk'}{v'} = \frac{v(q^t-1)}{q^{m+1}-1}$. By this expression, given q, as m increases $\mu_{v',k',v}$ decreases, and given m, as q increases $\mu_{v',k',v}$ decreases. Further, this expression approaches 0 for high values of q or m. Thus, to obtain a *finite* set of candidates, we provide a positive lower bound on $\mu_{v',k',v}$, denoted MIN_K (our implementation sets it to t). That is, the finite set of candidates CoVerD considers is:

$$\{(q,m) \in \mathbb{N}^2 \mid q \text{ is a prime power},\ m \geq t,\ v' \geq v,\ \mu_{v',k',v} \geq \text{MIN\_K}\}$$

Estimating the block size distribution For every CVD candidate, defined by (q, m), CoVerD estimates the distribution of its block sizes. While Theorem 1 provides expressions for the mean block size and the variance, it does not define the block size distribution. We empirically observe that our CVDs have the property that the distribution of their block sizes resembles a discrete approximation of a Gaussian distribution with mean $\mu_{v',k',v}$ and variance $\sigma^2_{v',k',v}$. The higher the mean and the number of blocks, the higher the resemblance. Figure 5a visualizes this resemblance for a CVD, with $v = 784$, induced from a PG with parameters $q = 17$, $m = 5$, and $t = 5$. We believe this resemblance exists because a CVD is partially-induced from a PG covering given *a random set of numbers L*. This resemblance may not hold for other choices of L, for example for the choice of L proposed by [27], which compute a partially-induced covering whose maximal block size is bounded (unlike our CVD). Because of this resemblance, we model the block size as drawn from the Gaussian distribution with the true mean and variance $\mathcal{G}\left(\mu_{v',k',v}, \sigma^2_{v',k',v}\right)$. Even if this modeling is imprecise, in practice, it is sufficient to allow CoVerD identify the candidate CVD leading to the minimal analysis time. Formally, given a CVD candidate defined by (q, m), the distribution of the block sizes is $dist(q, m) = \{N^{q,m}_k \mid k \leq \text{MAX\_K}\}$, where $N^{q,m}_k$ is our estimation of the number of blocks of size k in this CVD. We define the probability that a block size in this CVD is of size k as: $\mathbb{P}(k - 0.5 < Z \leq k + 0.5) = \Phi\left(\frac{(k+0.5)-\mu_{v',k',v}}{\sigma_{v',k',v}}\right) - \Phi\left(\frac{(k-0.5)-\mu_{v',k',v}}{\sigma_{v',k',v}}\right)$, where $Z \sim \mathcal{G}\left(\mu_{v',k',v}, \sigma^2_{v',k',v}\right)$ and Φ is the cumulative distribution function (CDF) of a Gaussian distribution with mean 0 and variance 1. The number of blocks $b^{q,m}$ is identical to the number of blocks in the PG covering, which has a closed-form expression [16]. Thus, the estimated number of blocks of size k is: $\tilde{N}^{q,m}_k = b^{q,m} \cdot \mathbb{P}(k-0.5 < Z \leq k+0.5)$. To make the estimated number an integer, we define $N^{q,m}_k$ as the floor of $\tilde{N}^{q,m}_k$ and add 1 with probability of the remainder: $N^{q,m}_k = \left\lfloor \tilde{N}^{q,m}_k \right\rfloor + X$ where $X \sim \text{Bern}\left(\tilde{N}^{q,m}_k - \left\lfloor \tilde{N}^{q,m}_k \right\rfloor\right)$. Figure 5b visualizes how close our estimation of the block size distribution is to the distribution of



(a) The block size distribution of a `CVD` and the respective Gaussian distribution.

(b) Our estimated block size distribution vs. the distribution of a respective `CVD`.

Fig. 5: Block size distributions.

the `CVD` shown in Figure 5a. We note that `CoVerD` considers a candidate and estimates its block size distribution only if its estimated number of overly large blocks (larger than MAX_K) is close to zero. Formally, it considers candidates that satisfy $b^{q,m} \cdot \left(1 - \Phi\left(\frac{\text{MAX\_K} - \mu_{v',k',v}}{\sigma_{v',k',v}}\right)\right) \leq \epsilon$, where ϵ is a small number. This is the reason that $dist(q,m) = \{N_k^{q,m} \mid k \leq \text{MAX\_K}\}$ consists of estimations only for blocks whose size is at most MAX_K.

Predicting the best `CVD` Given the candidates and their estimated block size distributions, `CoVerD` chooses the `CVD` which will enable `CoVerD` to minimize the overall analysis time. To this end, it predicts for every candidate `CVD` its overall analysis time. The prediction relies on: (1) the estimated number of blocks $N_k^{q,m}$ of size k, (2) the average analysis time of a block of size k, denoted $k\_array[k][time]$ (given by the initial sampling), (3) the fraction of the non-robust blocks of size k, which is one minus the success rate of k, denoted $k\_array[k][success]$ (given by the initial sampling) and (4) the analysis time of refining a non-robust block of size k, denoted $T(k)$ (given by the dynamic programming, as defined in [30]). Similarly to Calzone's dynamic programming, the analysis time is the sum of the analysis time of verifying all blocks in the `CVD` and the analysis time of the refinements of the non-robust blocks:

$$T_{dist(q,m)} = \sum_{k=t}^{\text{MAX\_K}} N_k^{q,m} \cdot (k\_array[k][time] + (1 - k\_array[k][success]) \cdot T(k))$$

This computation ignores blocks of size less than t since they do not cover any subset of size t and need not be analyzed to prove L_0 robustness. After predicting the analysis time of every candidate, `CoVerD` picks the candidate with the minimal time.

```
Algorithm 1: CoveringGenerator($q$, $m$, $t$, $L$, $i_{GPU}$)
  Input: PG parameters ($q$, $m$, $t$), an ordered set of $v$ indices $L = [s_1, \ldots, s_v]$
         which is a subset of $[\frac{q^{m+1}-1}{q-1}]$, and an index of a GPU $i_{GPU}$.
  Output: A stream of the covering verification design's blocks.
1 $\forall i \in [v]$. $P[:, i]$ = a unique vector in $\mathbb{F}_q^{m+1}$ computed for $s_i$    // $P \in \mathbb{F}_q^{(m+1)\times v}$
2 $\mathcal{M} = [M \in \mathbb{F}_q^{(m-t+1)\times(m+1)} \mid M$ is full rank and in reduced row echelon form]
3 for $j = 0$; $j < |\mathcal{M}|$; $j++$ do
4     if $j$ modulo GPUs $\neq i_{GPU}$ then continue
5     $R = \mathcal{M}[j] \times P$
6     block = $\left\{i \in [v] \mid R[:, i] = \vec{0}\right\}$    // generate induced block
7     output block
```

4.3 Constructing a Covering Verification Design

In this section, we present our covering generator that computes a CVD. The covering generator operates as an independent process, one for every GPU, that outputs blocks a-synchronically. At every iteration, every GPU worker obtains a block from its covering generator, analyzes it with GPUPoly, and refines if needed. If the block is robust or its refinement does not detect an adversarial example, the GPU worker obtains the next block from the covering generator. The covering generator relies on the chosen CVD's parameters q and m and the ordered set L from the planning component. It computes the PG covering for (q, m, t) block-by-block and induces it to obtain a CVD. Generally, its construction follows the meta-algorithm of generating PG coverings described in [16]. The novel parts are our implementation of inducing blocks immediately upon generating them and partitioning them to enable their analysis to proceed in parallel over the available GPUs. We next describe the covering generator.

Algorithm 1 shows the algorithm of our covering generator. It takes as input the PG parameters (q, m, t), an ordered set L of v indices from $[v']$ (where $v' = \frac{q^{m+1}-1}{q-1}$), and the GPU index i_{GPU}. As described in Section 3, a PG construction for (v', k', t) views v' as the number of points in the geometry. Formally, given the finite field $\mathbb{F}_q$ of order q, we identify the points of the geometry as a subset $W \subset \mathbb{F}_q^{m+1}$ of size v' (technically, these points are representatives of equivalence classes over $\mathbb{F}_q^{m+1}$, as explained in [16]). To later partially-induce the covering using L, Algorithm 1 maps every index in L to a unique point in W and stores all points (column vectors) in a matrix P (Line 1). Then, Algorithm 1 begins to construct the PG covering by computing flats (linear subspaces) of dimension $t-1$, each containing $k' = \frac{q^t-1}{q-1}$ points. As described in [16], every block in the PG covering is a solution (a set of points in W) to $m-t+1$ independent linear equations over $m+1$ variables. Such a linear system can be represented as a full rank matrix, where its solutions are vectors in the matrix's null space. Thus, to compute the blocks in the PG covering, Algorithm 1 defines a set of matrices $\mathcal{M}$, each is over $\mathbb{F}_q$, of dimension $(m-t+1) \times (m+1)$, and full rank (equal to $m-t+1$). Each matrix has exactly k' points in W in its null space. These points form a PG block (a flat of dimension $t-1$). To avoid block duplication, the matrices in $\mathcal{M}$ need to have different null spaces. Thus, Algorithm 1 considers matrices in reduced row echelon form, i.e., $\mathcal{M}$ is all full rank $(m-t+1)\times(m+1)$ matrices over $\mathbb{F}_q$ in reduced row echelon form (Line 2). The covering generator then iterates these matrices. To avoid a high memory overhead, the matrix $\mathcal{M}[j]$ is generated only upon reaching its index j. If j belongs to the disjoint part of the given GPU, its induced block is generated (Line 4). To construct a PG block, one needs to compute all the points in the null space of $\mathcal{M}[j]$. However, the generator requires only the partially-induced blocks. Thus, it *immediately induces* the block by obtaining all points s_i, for $i \in [v]$, whose respective point $P[:,i]$ belongs to the null space of $\mathcal{M}[j]$. To this end, it defines R as the multiplication of $\mathcal{M}[j]$ and P (Line 5), forms the induced block by identifying the points that are in the null space of $\mathcal{M}[j]$ (i.e., every s_i satisfying $R[:,i]=\vec{0}$), and makes the induced block a subset of $[v]$ by mapping every s_i in the induced block to i (Line 6).

4.4 A Running Example

In this section, we describe a real execution of CoVerD, for an MNIST image, a fully-connected network (6×200_PGD in Section 5), and $t=4$. CoVerD begins with the planning component. It first estimates the success rate and average analysis time of blocks. For every $k \in \{4,5,\dots,200\}$, it samples blocks S (subsets of [784]) of size k and submits their neighborhood $I_S(x)=\{x' \in [0,1]^v \mid \forall i \notin S.\ x'_i = x_i\}$ to GPUPoly. Based on all samples for k, it estimates the success rate and the average analysis time. For instance, for $k=34$ the success rate is 94.05% and the average analysis time is 16.19ms, while for $k=41$ they are 65.85% and 16.96ms. Then, CoVerD runs Calzone's dynamic programming to map every $k \in \{5,6,\dots,200\}$ to the refinement size. For example, $k=34$ is mapped to 28 and $k=41$ to 33. Next, CoVerD determines the CVD for the first covering, out of 50 candidates. For each candidate, it predicts the block size distribution and the respective overall analysis time of this candidate. To this end, it computes the mean, variance, and number of blocks using the closed-form expressions. For example, the CVD of the candidate $(q=23, m=4)$ has mean block size 34.087, variance 32.518 and $292,561$ blocks. The CVD of $(q=19, m=4)$ has mean block size 41.263, variance 38.867, and $137,561$ blocks. Although the second candidate has less than half the number of blocks of the first candidate, CoVerD predicts that using the first candidate will enable a faster analysis. This is because its success rate is significantly higher and thus it will require fewer refinements (e.g., the success rate of its mean block size is 94.05%, whereas the second candidate's success rate of the mean block size is 65.85%). The estimated analysis times (in minutes) are $T_{dist(23,4)}=21.20$ and $T_{dist(19,4)}=27.92$. The last step of the planning component samples an ordered set L of size 784 (the number of pixels in the MNIST image) from $[\frac{23^5-1}{23-1}]=[292561]$. In total, the planning component takes 63.5 seconds.

Then, CoVerD continues to the analysis component. It starts by creating eight instances of the covering generator (Algorithm 1), one for each GPU. A covering generator creates blocks for its GPU one-by-one, given $q^* = 23$, $m^* = 4$, $t = 4$ and L. For every `CVD` block S, the GPU worker defines its neighborhood $I_S(x)$ and submits to GPUPoly. If GPUPoly verifies successfully, the next `CVD` block is obtained. If GPUPoly fails proving robustness, S is refined. As example, if a block S of size 34 is refined, the analysis pushes to the stack all blocks in the covering $C_S(34, 28, 4)$, which is the covering for $(34, 28, 4)$ that is in the covering database, where the numbers are renamed to range over the numbers in S. In this example, GPUPoly is invoked $659,326$ times, where 44% of these calls are for blocks in the `CVD`. The maximal size of block submitted to GPUPoly is 62 and the minimal size is 8. In particular, `CoVerD` did not submit any block of size $t = 4$ (i.e., there are no calls to the MILP verifier). The analysis takes 23.49 minutes, which is only 10.8% higher than the estimated time.

5 Evaluation

In this section, we describe the experimental evaluation of `CoVerD` on multiple datasets and networks and compare it to Calzone.

Implementation and setup We implemented `CoVerD`[1] as an extension of Calzone[2]. Experiments ran on a dual AMD EPYC 7713 server, 2TB RAM, eight NVIDIA A100 GPUs and Ubuntu 20.04.1. We evaluate `CoVerD` on the networks evaluated by Calzone, whose architectures are described in ERAN[3]. We consider networks trained for popular image datasets: MNIST and Fashion-MNIST, consisting of 28×28 greyscale images, and CIFAR-10, consisting of 32×32 colored images. `CoVerD`'s hyper-parameters are: the maximal block size is MAX_K = 200, the number of samples is initially $n_{\text{samples}} = 400$ and after $n_{\text{fail}} = 10$ failures, it is reduced to $n_{\text{samples}} = 24$, and the bound on the estimated number of overly large blocks is $\epsilon = 0.01$. Our covering database, used for the refinement steps, contains coverings for $v, k \leq 200$, $t \leq 6$. The covering sizes are restricted to at most $500,000$ blocks. This limitation is stricter than Calzone, which limited to 10^7, but in practice this is unnoticeable since `CoVerD` only uses the coverings for refinements, and even Calzone typically refines to coverings whose size is at most $500,000$. Like Calzone, the database consists of coverings computed by extending coverings from the La Jolla Covering Repository Tables[4] using construction techniques from [16, Section 6.1]. Additionally, our database includes finite geometry coverings (for $v, k \leq 200$, $t \leq 6$) and extends coverings using the dynamic programming of [16, Section 5]. Like Calzone, We ran `CoVerD` with eight GPUPoly instances and five MILP instances, except for the CIFAR-10 network where it ran 50 MILP instances. For the matrix multiplication over finite fields (Algorithm 1), `CoVerD` relies on an effective library [18] and considers only prime numbers for q (since matrix multiplication is too slow for prime powers).

[1] `https://github.com/YuvShap/CoVerD`

[2] `https://github.com/YuvShap/calzone`

[3] `https://github.com/eth-sri/eran`

[4] `https://ljcr.dmgordon.org/cover/table.html`

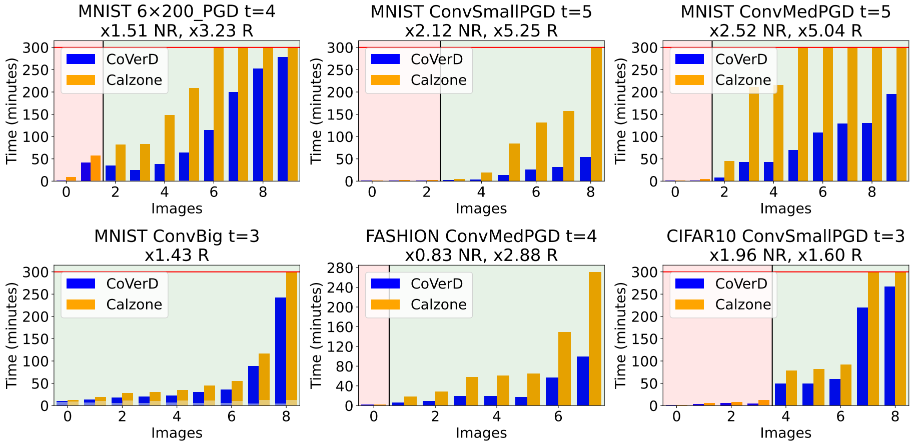


Fig. 6: `CoVerD` vs. Calzone on Calzone's most challenging benchmarks.

Comparison to Calzone We begin by evaluating `CoVerD` on Calzone's benchmarks (i.e., the same networks, images and timeouts) for $t \geq 3$. Figure 6 shows the comparison for the most challenging benchmarks of Calzone, and Figure 7 shows comparisons for $t = 4$ (the plots for $t = 3$ are shown in [31, Appendix A]). For a given network and t, the plot shows the execution time in minutes of `CoVerD` and Calzone for every t-ball. The x-axis orders the t-balls by `CoVerD`'s output: non-robust (in light red background), robust (in light green background), and timeout (in light blue background, e.g., Figure 7, top). Within each section, the t-balls are sorted by their execution time for clearer visuality. Timeouts, of `CoVerD` or Calzone, are shown by bars reaching the red horizontal line. The lower part of each bar shows in a lighter color the execution time of the initial sampling (unless it is too short to be visible in the plot). The sampling time is highlighted since Calzone and `CoVerD` sample slightly differently: Calzone samples 400 sets of size k, for every $k \leq 99$, while `CoVerD` samples up to $k \leq 200$ and reduces the number of samples after observing ten k values whose average success rate is zero. We note that the other computations of the planning component take a few seconds. The plots' titles include the speedup in the average analysis time of `CoVerD` over Calzone for non-robust t-balls (NR) and for robust t-balls (R).

The plots show that, on the most challenging benchmarks (Figure 6), `CoVerD` is always faster than Calzone, except for two non-robust t-balls which `CoVerD` completes their analysis within 140 seconds. In the plots of Figure 7, `CoVerD` is always faster than Calzone except for thirteen 4-balls whose analysis terminates within seven minutes by both verifiers. In the other plots (Figure 9 in [31, Appendix A]), where $t = 3$, Calzone is sometimes faster, but in these cases the analysis time is typically short. In other words, the significance of `CoVerD` is in shortening the analysis time of t-balls with long analysis time. On average, `CoVerD` is faster than Calzone in verifying robust t-balls by 1.3x for $t = 3$, by 2.8x for $t = 4$, and by 5.1x for $t = 5$.

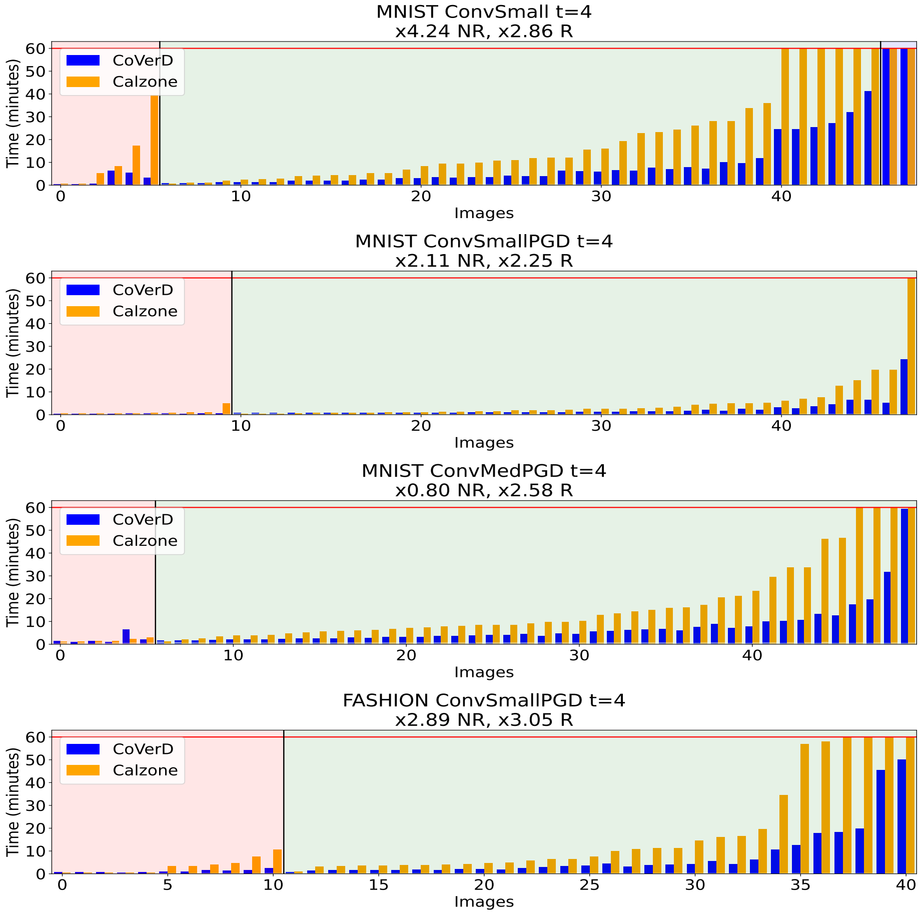


Fig. 7: `CoVerD` vs. Calzone for $t = 4$.

Challenging benchmarks Next, we show more challenging benchmarks than Calzone. We evaluate the robustness of three networks for t-balls with larger values of t than Calzone considers, for $t = 5$ and for $t = 6$ (we remind that Calzone is evaluated for $t \leq 5$). Similarly to Calzone's most challenging benchmarks, these benchmarks evaluate `CoVerD` for ten images (misclassified images are discarded) and a five hour timeout. Figure 8 shows `CoVerD`'s analysis time. `CoVerD` completes the analysis for 73% t-balls. Further, it verifies robustness in some 6-balls within 42 minutes. As before, `CoVerD` is significantly faster for non-robust t-balls.

We provide additional statistics on `CoVerD` in [31, Appendix A].

6 Related Work

In this section, we discuss the closest related work.

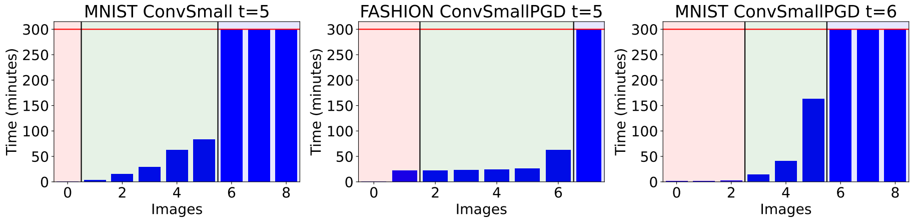


Fig. 8: `CoVerD`'s new benchmarks.

Robustness verification of neural networks Many works propose robustness verifiers for neural networks. Most works focus on local robustness in L_∞ neighborhoods, defined by a series of intervals [25,34,32,13,21,2,37,36,12]. Some verifiers provide a complete analysis, i.e., they determine whether a network is robust in the given neighborhood [34,21,12]. These approaches typically rely on constraint solving (SAT/SMT solvers or MILP solvers) and thus they often do not scale to large networks. Incomplete verifiers scale the analysis by over-approximating the non-linear computations of the network (e.g., the activation functions) by linear approximations or abstract domains [25,32,13,36]. Several local robustness verifiers address L_2-balls, e.g., by computing a bound on the network's global or local Lipschitz constant [22,19], or L_1-balls [38,41]. Other approaches analyze robustness in L_p-balls for $p \in \{0, 1, 2, \infty\}$ using randomized smoothing [6,39,23,28,11], providing probabilistic guarantees. To the best of our knowledge, Calzone [30] is the first work to deterministically verify local robustness in L_0-balls. Other works prove robustness in neighborhoods defined by high-level features [20,24,3].

Covering and combinatorial designs `CVD` is related to several combinatorial designs: the combinatorial design defined by [27], covering designs [16] and balanced incomplete block designs [7]. Covering designs, in particular finite geometry coverings, have been leveraged in various domains, including file information retrieval [27], file organization [1] and coding theory [5]. General combinatorial designs have also been leveraged in various domains in computer science [8].

7 Conclusion

We present `CoVerD`, an L_0 robustness verifier for neural networks. `CoVerD` boosts the performance of a previous L_0 robustness verifier by employing several ideas. First, it relies on a covering verification design (`CVD`), a new combinatorial design partially inducing a projective geometry covering. Second, it chooses between candidate `CVDs` without constructing them but only predicting their block size distribution. Third, it constructs the chosen `CVD` on-the-fly to keep the memory overhead minimal. We evaluate `CoVerD` on fully-connected and convolutional networks. We show that it boosts the performance of proving a network's robustness to at most t perturbed pixels on average by 2.8x, for $t = 4$, and by 5.1x, for $t = 5$. For $t = 6$, `CoVerD` sometimes proves robustness within 42 minutes.

References

1. Abraham, C., Ghosh, S., Ray-Chaudhuri, D.: File organization schemes based on finite geometries. Information and Control **12**(2), 143–163 (1968)
2. Anderson, G., Pailoor, S., Dillig, I., Chaudhuri, S.: Optimization and abstraction: a synergistic approach for analyzing neural network robustness. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI (2019)
3. Balunovic, M., Baader, M., Singh, G., Gehr, T., Vechev, M.T.: Certifying geometric robustness of neural networks. In: Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS (2019)
4. Carlini, N., Wagner, D.A.: Towards evaluating the robustness of neural networks. In: IEEE Symposium on Security and Privacy, SP (2017)
5. Chan, A., Games, R.: (n,k,t))-covering systems and error-trapping decoding (corresp.). IEEE Transactions on Information Theory **27**(5), 643–646 (1981)
6. Cohen, J., Rosenfeld, E., Kolter, J.Z.: Certified adversarial robustness via randomized smoothing. In: Proceedings of the 36th International Conference on Machine Learning, ICML. vol. 97. PMLR (2019)
7. Colbourn, C.J., Dinitz, J.H. (eds.): Handbook of Combinatorial Designs. Chapman and Hall/CRC, 2nd edn. (2006). https://doi.org/10.1201/9781420010541
8. Colbourn, C.J., van Oorschot, P.C.: Applications of combinatorial designs in computer science. ACM Comput. Surv. **21**(2), 223–250 (jun 1989)
9. Croce, F., Andriushchenko, M., Singh, N.D., Flammarion, N., Hein, M.: Sparse-rs: A versatile framework for query-efficient sparse black-box adversarial attacks. In: Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI , The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI. AAAI Press (2022)
10. Croce, F., Hein, M.: Mind the box: l_1-apgd for sparse adversarial attacks on image classifiers. In: Proceedings of the 38th International Conference on Machine Learning, ICML. PMLR (2021)
11. Dvijotham, K.D., Hayes, J., Balle, B., Kolter, J.Z., Qin, C., György, A., Xiao, K., Gowal, S., Kohli, P.: A framework for robustness certification of smoothed classifiers using f-divergences. In: 8th International Conference on Learning Representations, ICLR. OpenReview.net (2020)
12. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: Automated Technology for Verification and Analysis - 15th International Symposium, ATVA. Lecture Notes in Computer Science, vol. 10482. Springer (2017)
13. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.T.: AI2: safety and robustness certification of neural networks with abstract interpretation. In: IEEE Symposium on Security and Privacy, SP (2018)
14. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), `http://www.deeplearningbook.org`
15. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: 3rd International Conference on Learning Representations, ICLR (2015)
16. Gordon, D.M., Kuperberg, G., Patashnik, O.: New constructions for covering designs. J. COMBIN. DESIGNS **3**, 269–284 (1995)

17. Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P.D.: Adversarial perturbations against deep neural networks for malware classification. CoRR **abs/1606.04435** (2016)
18. Hostetter, M.: Galois: A performant NumPy extension for Galois fields (11 2020), `https://github.com/mhostetter/galois`
19. Huang, Y., Zhang, H., Shi, Y., Kolter, J.Z., Anandkumar, A.: Training certifiably robust neural networks with efficient local lipschitz bounds. In: Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems NeurIPS (2021)
20. Kabaha, A., Drachsler-Cohen, D.: Boosting robustness verification of semantic feature neighborhoods. In: Static Analysis - 29th International Symposium, SAS. Lecture Notes in Computer Science, vol. 13790. Springer (2022)
21. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: Computer Aided Verification - 29th International Conference, CAV (2017)
22. Leino, K., Wang, Z., Fredrikson, M.: Globally-robust neural networks. In: Proceedings of the 38th International Conference on Machine Learning, ICML. Proceedings of Machine Learning Research, vol. 139. PMLR (2021)
23. Li, B., Chen, C., Wang, W., Carin, L.: Certified adversarial robustness with additive noise. In: Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS (2019)
24. Mohapatra, J., Weng, T., Chen, P., Liu, S., Daniel, L.: Towards verifying robustness of neural networks against A family of semantic perturbations. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR (2020)
25. Müller, C., Serre, F., Singh, G., Püschel, M., Vechev, M.T.: Scaling polyhedral neural network verification on gpus. In: Proceedings of Machine Learning and Systems MLSys (2021)
26. Papernot, N., McDaniel, P.D., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: IEEE European Symposium on Security and Privacy, EuroS&P (2016)
27. Ray-Chaudhuri, D.K.: Combinatorial information retrieval systems for files. SIAM J. Appl. Math. **16**(5), 973–992 (sep 1968)
28. Salman, H., Li, J., Razenshteyn, I.P., Zhang, P., Zhang, H., Bubeck, S., Yang, G.: Provably robust deep learning via adversarially trained smoothed classifiers. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS. pp. 11289–11300 (2019)
29. Schönheim, J.: On coverings. Pacific Journal of Mathematics **14**(4), 1405 – 1411 (1964)
30. Shapira, Y., Avneri, E., Drachsler-Cohen, D.: Deep learning robustness verification for few-pixel attacks. Proc. ACM Program. Lang. **7**(OOPSLA1) (2023)
31. Shapira, Y., Wiesel, N., Shabelman, S., Drachsler-Cohen, D.: Boosting few-pixel robustness verification via covering verification designs. CoRR **abs/2405.10924** (2024)
32. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. PACMPL **3**(POPL) (2019)
33. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: 2nd International Conference on Learning Representations, ICLR (2014)

34. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In 7th International Conference on Learning Representations, ICLR (2019)
35. Todorov, D.: Combinatorial coverings. Ph.D. thesis, PhD thesis, University of Sofia, 1985 (1985)
36. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C., Kolter, J.Z.: Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In: Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems (2021)
37. Wu, H., Ozdemir, A., Zeljic, A., Julian, K., Irfan, A., Gopinath, D., Fouladi, S., Katz, G., Pasareanu, C.S., Barrett, C.W.: Parallelization techniques for verifying neural networks. In: Formal Methods in Computer Aided Design, FMCAD. IEEE (2020)
38. Wu, Y., Zhang, M.: Tightening robustness verification of convolutional neural networks with fine-grained linear approximation. In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI. pp. 11674–11681. AAAI Press (2021)
39. Yang, G., Duan, T., Hu, J.E., Salman, H., Razenshteyn, I.P., Li, J.: Randomized smoothing of all shapes and sizes. In: Proceedings of the 37th International Conference on Machine Learning, ICML. Proceedings of Machine Learning Research, vol. 119. PMLR (2020)
40. Yuviler, T., Drachsler-Cohen, D.: One pixel adversarial attacks via sketched programs. Proc. ACM Program. Lang. **7**(PLDI) (2023)
41. Zhang, H., Weng, T., Chen, P., Hsieh, C., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems, NeurIPS (2018)