Hard-label Black-box Universal Adversarial Patch Attack

Guanhong Tao, Shengwei An, Siyuan Cheng, Guangyu Shen, and Xiangyu Zhang, *Purdue University*

https://www.usenix.org/conference/usenixsecurity23/presentation/tao

This paper is included in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Hard-label Black-box Universal Adversarial Patch Attack

Guanhong Tao, Shengwei An, Siyuan Cheng, Guangyu Shen, Xiangyu Zhang

Purdue University

Abstract

Deep learning models are widely used in many applications. Despite their impressive performance, the security aspect of these models has raised serious concerns. Universal adversarial patch attack is one of the security problems in deep learning, where an attacker can generate a patch trigger on pre-trained models using gradient information. Whenever the trigger is pasted on an input, the model will misclassify it to a target label. Existing attacks are realized with access to the model's gradient or its output confidence. In this paper, we propose a novel attack method HARDBEAT that generates universal adversarial patches with access only to the predicted label. It utilizes historical data points during the search for an optimal patch trigger and performs focused/directed search through a novel importance-aware gradient approximation to explore the neighborhood of the current trigger. The evaluation is conducted on four popular image datasets with eight models and two online commercial services. The experimental results show HARDBEAT is significantly more effective than eight baseline attacks, having more than twice high-ASR (attack success rate) patch triggers (>90%) on local models and 17.5% higher ASR on online services. Three existing advanced defense techniques fail to defend against HARDBEAT.

1 Introduction

Deep learning (DL) has become an integral part of many critical systems, such as face recognition [47], autonomous driving [9, 10], and drug discovery [12]. For example, Tesla vehicles [33] fully rely on DL-based computer vision models to recognize traffic signs, road conditions, etc., without the assistance of LiDAR [10]. DL models are also widely used in security applications, such as malware detection [66], forensics [23, 58], binary reverse-engineering [48, 65], etc. Recent ChatGPT [44] demonstrates great potential in serving a wide range of applications.

While the success of deep learning is astonishing, the security concerns of those models haven't eased off. They are susceptible to universal adversarial patch attacks [4, 8, 20, 32, 64], where an attacker generates a *patch trigger* on a pre-trained model. The model will misclassify any input inserted with the trigger to a *target label*. Universal adversarial patches have a prompt attack effect as the attacker only needs to stamp the generated patch (e.g., a sticker) on unseen inputs, which differs from traditional adversarial attacks that require generating perturbations for each input [11, 40]. Existing attacks [4, 8, 32] assume white-box access to the model, allowing them to compute the gradient from the model output (e.g., the prediction confidence) to the patch trigger for optimization. Later studies [20, 64] show that it is viable to craft universal adversarial patches even if the attacker cannot access the model weight parameters but only knows the output confidence/probability, which is called *soft-label* or *score-based* attacks.

A natural question one would ask is "*can we craft universal adversarial patches without access to the model internals and the output confidence?*" More specifically, if the attacker can only query the model and obtain the predicted label for each query, is it feasible to construct a patch that can cause misclassification for a large set of unseen samples? This would be a severe security problem for DL models if such an attack is realizable. We aim to answer the question in this paper.

There is a line of work studying hard-label black-box adversarial attacks. These attacks focus on generating per-instance perturbations with access only to the predicted label. For example, HSJA [13] and QEBA [36] leverage gradient estimation to generate adversarial examples. They start from a target-class image and gradually reduce the adversarial example's distance to the victim image (by adding adversarial perturbations). The final adversarial example is visually similar to the victim image while being misclassified to the target label. GRAPHITE [25] and SparseEvo [57] aim to perturb a small number of pixels on the victim image such that it can be misclassified in a black-box setting. They search for critical adversarial pixels using a binary search or an evolutionary algorithm. These hard-label black-box adversarial attacks share many similarities with black-box universal adversarial patch attacks, such as the access only to the predicted label and causing misclassification to the target label. Hence, they can be extended to construct patch triggers.

The essence of existing black-box adversarial attacks is similar to that of the *Metropolis-adjusted Langevin Algorithm* (MALA) [5], a *Markov chain Monte Carlo* (MCMC) method [27]. It constructs a Markov chain to approximate (high-dimensional) probability distributions. Specifically, it starts from a random point and chooses a candidate from the neighborhood of the current sample by adding a perturbation drawn from a distribution (e.g., a 0-mean Gaussian distribution). With a probability, the candidate is accepted as the current sample if it is more valuable; or discarded otherwise. MALA leverages gradients when perturbing the sample. As gradients are inaccessible in the black-box attack setting, existing attacks estimate the gradient information by randomly sampling perturbations in a 0-mean Gaussian distribution and measuring which perturbation could cause misclassification.

MCMC methods, however, are known to require extensive initial steps in order to achieve good performance. The Markov property of MCMC methods implies that they are state-less, where a new sample is only dependent on the current sample. In other words, they do not leverage historical information. Although this is less of a problem in traditional MCMC applications, which often have a very large sampling budget, it is not ideal for black-box universal adversarial patch attacks that have a limited sampling budget. Another design proposal is to leverage stateful search algorithms such as the *Genetic Algorithm* (GA) [3]. GA maintains a pool of past samples and continuously updates these samples using *cross-over* and *mutation* operations as an evolution process. It thus takes advantage of historical samples. However, its performance in black-box attacks is still limited (see Section 6.2). This is because each evolution step requires a large number of queries to derive new samples, although many new samples are not sufficiently good and are therefore excluded.

We propose to take advantage of both MCMC and GA methods. Given the limited query budget, instead of discarding previous query results, we utilize historical samples, similar to GA methods. Our method hence is stateful. Moreover, we also perform focused/directed gradient-direction search as in MALA, which helps avoid less effective evolution steps in GA that produce low-quality offsprings. In addition to searching in the neighborhood of the current sample, with a probability, we also explore the neighborhood of certain previous local minima. That is, we identify and record past good samples, and linearly interpolate between two local minima. Since two minima are often connected by a ridge, we call it *ridge interpolation*.

The novelty of this paper lies in leveraging historical high-ASR (attack success rate) data points and interpolating local minima, which have not been studied in existing attacks [13, 20, 25, 30, 51, 57, 64]. Our gradient estimation is specially designed for misclassifying multiple samples, which has rarely been explored in hard-label black-box adversarial attacks targeting individual inputs. Our importance-aware estimation adaptively adjusts weights on gradient directions for different samples, which is novel and effective, and can be generalized to other universal hard-label black-box attacks.

Our contributions are summarized as follows.

- We identify the limitations of existing hard-label black-box attacks and propose a new method that possesses the advantages of both MCMC and GA types of methods.
- We propose to explore the neighborhood of previous high-ASR samples by interpolating between two close-by local minima.
- We introduce an importance-aware gradient-direction estimation method that better approximates the directions from diverse model inputs.
- We implement an attack prototype called HARDBEAT (Hard-label black-box universal adversarial patch attack), which is publicly available at [1]. We evaluate the attack performance of HARDBEAT on four popular image datasets, with two models for each dataset. Compared to eight baseline black-box attacks, HARDBEAT generates more than twice as many patch triggers with a high ASR (>90%) using the same query budget. We also apply HARDBEAT to two online commercial services, Microsoft Azure [41] and Clarifai [18]. It achieves an average ASR of 74%, outperforming the baseline by 17.5%. Regarding possible countermeasures against HARDBEAT, we employ a state-of-the-art certifiable defense, a query-based defense, and a universal adversarial patch detection approach. The results show existing defense techniques can hardly defend against HARDBEAT.

2 Related Work on Black-box Attacks

Existing black-box adversarial attacks can be categorized into substitute model based attacks and query-based attacks.

Substitute Model Based Attacks. This type of attacks aims to build a substitute model to conduct the attack. It either assumes access to the training data of the victim model or uses synthetic data to query the model for labeling. With the labeled data, the attacker can train a substitute model to approximate the victim model. The adversarial examples are generated on the substitute model and transferred to the victim model [22, 26, 28, 39, 45, 46]. Many works focus on improving the transferability of generated adversarial examples such that they can have a high misclassification rate on the victim model [21, 31, 38, 60, 62].

Query-based Attacks. Different from substitute model based attacks, query-based attacks directly optimize adversarial examples by iteratively querying the victim model [2, 6, 15, 42, 56]. Specifically, they usually start from a target-class image and aim to reduce its distance to the victim image by adding adversarial perturbations. The goal is to have an adversarial example visually similar to the victim image while being misclassified to the target label. This is achieved by adding random noises on the input and iteratively querying the victim model to approximate the direction for perturbing the input. This type of attacks can achieve performance comparable to white-box attacks. Based on the knowledge of the attacker, query-based attacks can be further divided into two categories: *soft-label* (or score-based) attacks [14, 16, 20, 29, 30, 30, 42] and *hard-label* (or decision-based) attacks [7, 13, 36, 63]. Soft-label attacks have access to the output confidence from the victim model, while hard-label attacks can only obtain the predicted label.

In this paper, we study hard-label black-box attack. Boundary attack [7] uses random noises to perturb a target-class image and gradually reduces the noise magnitude to make it closer to the victim image. HSJA [13] leverages gradient approximation to estimate the direction for perturbing the input at each step. QEBA [36] improves HSJA by estimating a core subset of the gradient direction. Policy-Driven [63] proposes a policy network to learn the best optimization direction at each step. GRAPHITE [25] and SparseEvo [57] are L^0 based attacks. They aim to have a small number of perturbed pixels for generating adversarial examples. GRAPHITE uses a binary search to reduce the number while SparseEvo leverages an evolutionary algorithm. As we focus on hard-label black-box universal adversarial patch attacks, we adapt several representative black-box adversarial attacks as our baselines.

3 Threat Model

We describe the attacker's goal, knowledge, and capabilities in this section.

Attacker's Goal. The attacker aims to find universal adversarial patches in a pre-trained clean image classifier such that pasting the patch trigger on any input samples can cause the classifier to predict the attacker-chosen target label. The trigger shall have a prompt effect, meaning once crafted, it can be directly applied on any new image (without modification) and cause target-class misclassification. This requires the patch trigger to be persistent and robust when deployed on unseen inputs. The attack success rate to the target label shall be $>1/C$, where C is the number of classes in the classifier. Since the trigger is stamped on input images, it shall not significantly affect the visibility of the original input. Otherwise, it will raise suspicion and be easily detected by human inspectors or automatic tools. The goal is to craft a small universal adversarial patch that occupies <5% of the input image.

Attacker's Knowledge and Capabilities. We consider the hard-label black-box attack setting. The attacker has no knowledge of the victim classifier, including its weight parameters, hyper-parameters, the training procedure, the original training data, etc. This means the attacker cannot observe the behavior of the model or its gradient. The attacker also does not have access to the confidence of predicted labels or their rankings. The only feedback that the attacker can obtain from the victim classifier is the predicted label of a given input image.

We assume the attacker has a small set of images ($\leq$100 samples) for crafting universal adversarial patches. He/she can query the victim model to obtain the predicted label. The attacker can repeatedly query the model. In practice, the attacker has constraints on computational resources. Online services also prevent excessive model queries. We consider a budget for the number of queries $\leq$50k to carry out the attack, following the literature [13, 15, 20].

4 Attack Intuition

The attack goal is to find a universal adversarial patch such that pasting it on any samples from a class causes misclassification to a target label. This can be seen as an optimization problem, where the changing variable is the patch trigger and the objective is to have target-class predictions for a set of input samples (inserted with the trigger).

$$\arg\max_{\boldsymbol{m},\boldsymbol{p}} \mathbb{E}_{\boldsymbol{x}\sim\mathbb{X}}\Big[f\big((1-\boldsymbol{m})\odot\boldsymbol{x}+\boldsymbol{m}\odot\boldsymbol{p}\big)=y_t\Big], \tag{1}$$

where $\boldsymbol{m}$ and $\boldsymbol{p}$ are trigger variables, with $\boldsymbol{m}$ denoting the location of the trigger and $\boldsymbol{p}$ representing the trigger pattern; f is the subject model; $\boldsymbol{x}$ is an input sample from a set $\mathbb{X}$ and y_t is the target label. Figure 1a illustrates the optimization objective. It displays the landscape of the objective loss with the x-y plane denoting input features and z the loss value on the target label. Note that the figure illustrates the average loss for a set of samples, as a valid universal adversarial patch ought to be able to flip a large set of samples. A small loss value denotes that many samples are misclassified by the subject model to the target label. Observe that there are multiple plummets on the loss surface, representing local minima. The goal is to reach the optimal (lowest) point in the middle red plummet, leading to the maximum number of misclassified samples.

Assuming a white-box scenario where the model parameters are known, a typical approach is to construct a loss function (e.g., cross-entropy loss) to describe the objective in Figure 1a. The attack can hence leverage the fine-grained loss change and the gradient information from back-propagation (by computing the gradient of the loss function w.r.t. the input in Equation 1) to find the optimal patch trigger. It is often able to avoid getting stuck in the local minima by adjusting the learning rate or with the assistance of advanced optimization algorithms. In Figure 1a, from the initial A, the gradient points to B as it leads to a smaller loss value. Since there is a local minimum near B, it wanders a bit in the region. Then it escapes the local minimum with learning rate variations (see the longer arrow from C to D) and eventually finds the optimal point denoted by the red star.

The problem, however, becomes much harder in the black-box setting, especially with only access to the predicted label.

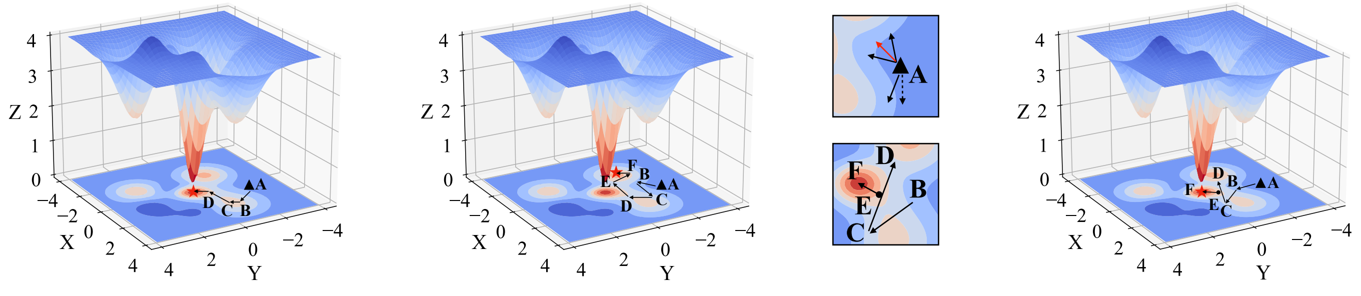

(a) White-box (b) HSJA [13] (c) (d) HARDBEAT

Figure 1: Illustration of the loss landscape of different attack methods. The x-y plane denotes input features and z the loss value on the target label. The top figure of (c) is the zoomed-in view of (b) and the bottom figure is the zoomed-in view of (d).

This is also known as the *hard-label* or *decision-based black-box* attack. The knowledge of the adversary is particularly limited. He/she can only query the subject model by providing an input and then obtain a predicted label. In other words, the attacker can only observe the number of misclassified samples. Each region with the same color in the 2-dimensional surface on the bottom of Figure 1b denotes the same number of misclassified samples. In other words, the attacker can only determine the region into which a data point (i.e., a potential patch trigger) falls, without knowing its gradient.

Searching for a good universal adversarial patch in such a setting is hence challenging. Existing attacks [13,36][1] approximate the direction to change the patch trigger by sampling a set of random points (around the current data point). The points that lead to more misclassifications are considered positive directions and otherwise negative directions. Figure 1c (the top figure) illustrates this concept. For the trigger at A, four random perturbations, denoted by the four black arrows, are individually added onto the trigger. The perturbed trigger is applied to the set of input samples, and the number of misclassifications is measured. Observe the arrows point in different directions. The dashed arrow indicates that the additive noise cannot cause more misclassifications (as it remains in the same blue region), so its opposite direction on the top is used. Three out of the four noises are able to increase the number of misclassified samples. The four arrows are then aggregated to form the red arrow, which represents the final direction for mutating the trigger at A. Note that the negative direction for the dashed arrow is necessary, as many times all the additive noises are unable to cause more misclassifications. In such cases, the negative directions indicate those that the trigger mutation should avoid.

However, it is not easy to determine the magnitude of the added noise. Unlike adversarial perturbations in which per-pixel changes are commonly bounded by L^2 or L^∞, there are no such constraints on pixels within the trigger area. A large value of random noise causes the search to overshoot potential optimal regions, and a small value can hardly change the model prediction. For example in Figure 1b, the attack starts from A (the black triangle) and continues along the path denoted by the letters in alphabetical order. At each point (e.g., A), the direction of the arrow is estimated through random noises, as described above and illustrated in Figure 1c. Observe that the attack misses the regions along the path from C to D and from D to E. The optimal region lies between D and E, which is not reached by the attack. Finally, it ends up at a local minimum denoted by a red star besides F. One possible proposal is to try dynamically adjusting the magnitude of added noises during the search. For instance, one can increase or decrease the noise magnitude when the number of misclassified samples does not change. However, the randomness in the increment or decrement may hinder any real improvement. Having a substantial number of random samples for direction estimation could improve the attack but entail too many queries to the subject model, which is a critical metric for black-box attacks.

[1]To our knowledge, hard-label black-box universal adversarial patch attack has not yet been explored in the literature. We adapt existing hard-label black-box adversarial attacks to this new setting as they share a lot of similarities.

Our Idea. As discussed in the introduction, existing black-box attacks follow a procedure similar to the *Metropolis-adjusted Langevin Algorithm* (MALA) [5], a MCMC method [27]. However, it is well-known that MCMC methods require expensive burn-ins, namely, the initial steps of random walks in order to achieve good results, especially in high-dimensional spaces. Constrained by the number of queries to the subject model, performing large burn-ins is prohibitively expensive in our context. An alternative is to perform stateful exploration of the space, which involves choosing the next sample based on all (or a subset) of previous samples, such as the *Genetic Algorithm* (GA) [3]. GA has been used in black-box attacks [57]. However, based on our experiments in Section 6.2, its performance is also limited. The reason is that each evolution step in GA requires a large number of queries to derive new samples, but many of these new samples are not discarded. As a result, the query budget quickly runs out.

The overarching idea of our technique is hence to leverage the advantages of both MCMC and GA methods. That is, we utilize historical samples so that every query result can be fully leveraged given the limited query budget. We also perform focused/directed search to avoid the cost-ineffective evolution steps in GA that generate many low-quality offsprings. In addition, with a probability, HARDBEAT also explores the neighborhood of some previous local minima by linear interpolation between two of them.

Figure 1d illustrates how HARDBEAT works for the example. Starting from A, it gets to B and then C, following the approximated gradient directions. Due to the uncertainty in gradient approximation, instead of finding the global minimum, it goes to D. HARDBEAT identifies that C and D denote two close-by local minima and speculates the area in between may be worth exploring. It hence samples an interpolated point E in between, that is, walking the ridge between C and D (see the bottom part of Figure 1c). The gradient approximation from E easily points to F, the global minimum. Note that when it reaches D, with a probability, HARDBEAT does not continue from D. Instead, it picks a historical point or an interpolated point, E in this case. Moreover, the interpolation between close-by minima allows exploring the hot area around E and F.

5 Attack Design

Our idea of black-box universal adversarial patch attack is to utilize historical samples such that previous query results can be fully leveraged given the limited query budget. We also perform focused/directed search through gradient approximation to explore the neighborhood of the current sample. The overall attack procedure consists of the following three steps.

- **Step 1:** Given a set of inputs and a target label, we initialize the location and the pattern of the patch trigger through random sampling, based on the attack success rate (ASR) to the target label on the input set. We then fix the trigger location and aim to find a better trigger pattern with a higher ASR. This substantially reduces the search space.
- **Step 2:** We select a data point for further exploration. With probabilities, the selection could be one of the three: (1) the current data point if its ASR exceeds a certain threshold; (2) a historical data point with a top-k ASR; (3) an interpolated data point between a historical top-k data point and its neighbor with a high ASR (a ridge point).
- **Step 3:** Gradients are approximated by sampling perturbations from a normalized uniform distribution. The perturbations are then applied to the data point selected at step 2. We leverage the importance of different inputs during gradient estimation. Gradient directions from samples that can increase ASR are enlarged, while other directions are reduced. The next data point is hence derived by mutating the selected point using the aggregated gradient direction.

Algorithm 1 Universal Adversarial Patch Trigger Generation

```
1: function TRIGGERGEN(decision function $f$, input $\boldsymbol{x}$, target label $y_t$, trigger size $s$)
2:     $\boldsymbol{\Delta}, \omega \leftarrow$ TRIGGERINIT($f$, $\boldsymbol{x}$, $y_t$, $s$)                ▷ Algorithm 2
3:     $hist_{\boldsymbol{\Delta}}, hist_{\omega} \leftarrow$ add $\boldsymbol{\Delta}, \omega$        ▷ Record historical pattern and ASR
4:     Initialize a similarity matrix $\boldsymbol{G}$ ▷ Record pairwise pattern similarities
5:     for $step$ in $0 \ldots max\_steps$ do
6:         /*** Pick a trigger ***/
7:         $[\omega]_{topk}, [\boldsymbol{\Delta}]_{topk} \leftarrow TopK(hist_{\omega}, hist_{\boldsymbol{\Delta}})$
8:         $\boldsymbol{\Delta}_{top}, idx \leftarrow$ Select from $[\boldsymbol{\Delta}]_{topk}$ with the probability $[\omega]_{topk}$
9:         $\mathbb{P} \leftarrow \min(1, hist_{\omega}[idx]/hist_{\omega}[step])$
10:        $u \leftarrow$ Draw a uniform random variable from $\mathcal{U}(0,1)$
11:        if $u \leq \mathbb{P}$ then
12:            $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta}_{top}$
13:            /*** Ridge interpolation ***/
14:            if $u \leq 0.5$ then
15:                $\boldsymbol{\Delta}_{neighbor} \leftarrow$ Select a neighbor of $\boldsymbol{\Delta}$ based on $\boldsymbol{G}$ and ASR
16:                $\alpha \leftarrow$ Draw a uniform random variable from $\mathcal{U}(0,1)$
17:                $\boldsymbol{\Delta} = \alpha \cdot \boldsymbol{\Delta} + (1-\alpha) \cdot \boldsymbol{\Delta}_{neighbor}$
18:            end if
19:        end if
20:        /*** Gradient-direction estimation ***/
21:        $\boldsymbol{\delta} \leftarrow$ GRADESTIMATE($f$, $\boldsymbol{x}$, $y_t$, $\boldsymbol{\Delta}$)                ▷ Algorithm 3
22:        $\boldsymbol{\Delta} = Optimizer(\boldsymbol{\Delta}, \boldsymbol{\delta})$
23:        Evaluate ASR and update $hist_{\omega}$, $hist_{\boldsymbol{\Delta}}$, $\boldsymbol{G}$
24:    end for
25:    return $\boldsymbol{\Delta}$
26: end function
```

Algorithm 1 describes our overall universal adversarial patch trigger generation method. We first initialize the trigger using Algorithm 2, which will be discussed in the next section. At each generation step, HARDBEAT gathers the top-k triggers with the highest ASR (line 7). We use $k = 4$ in the paper and study other choices in Appendix D. A top trigger is then randomly selected from this set based on their probabilities, which are their ASRs (line 8). HARDBEAT compares the ASR of the top trigger to that of the current-step trigger (line 9). It then draws a uniform random variable u from $\mathcal{U}(0,1)$ (line 10). If u is no larger than the probability, we choose the top trigger (lines 11-12). With the selected trigger, HARDBEAT further picks one of its neighbors and then interpolates between the two (lines 15-17). We then estimate the direction for perturbing the trigger and leverage an optimizer to update it (lines 21-22). Adam optimizer [34] is used in this paper. Other suitable optimizers can also be adopted. HARDBEAT iterates the process for a number of steps and finally outputs a trigger with high ASR. The initialization is discussed in Section 5.1; the neighbor selection is discussed in Section 5.2; and the importance-aware gradient-direction estimation is in Section 5.3.

5.1 Initializing Patch Trigger

At the very beginning of the attack, there is no prior knowledge of where the trigger shall be located and what it looks like. Here, we consider a square-shaped trigger such as a 4×4 patch for discussion simplicity. We study different trigger sizes and shapes in Section 6.3. A straightforward idea is

Algorithm 2 Trigger Initialization

```
1:  function TRIGGERINIT(decision function $f$, input $\boldsymbol{x}$, target label $y_t$, trigger size $s$)
2:      for $step$ in $0 \dots max\_steps1$ do
3:          $pos_x, pos_y \leftarrow$ Sample random values $< (input\_size - s + 1)$
4:          if $step > \lambda \cdot max\_steps1$ then
5:              $pos_x, pos_y \leftarrow$ Mutate the best position in the neighborhood
6:          end if
7:          Set mask $\boldsymbol{m}$ to 1 at position $(pos_x, pos_y)$ with size $s$ (0 otherwise)
8:          $\boldsymbol{p} \leftarrow$ Random values from clipped uniform distribution $\mathcal{U}(-4,4)$
9:          Obtain average ASR of $(1-\boldsymbol{m}) \odot \boldsymbol{x} + \boldsymbol{m} \odot \boldsymbol{p}$ and record the best
10:     end for
11:     for $step$ in $0 \dots max\_steps2$ do
12:         Fix mask $\boldsymbol{m}$ and add random noise on pattern $\boldsymbol{p}$
13:         Record $\boldsymbol{m}$ and $\boldsymbol{p}$ with a better ASR
14:     end for
15:     return $(\boldsymbol{m}, \boldsymbol{p})$, ASR
16: end function
```

to randomly sample a few locations and patterns, and choose a combination of the two with the highest ASR. This is not query-efficient as the search space is massive. For an input with a shape of 32×32, there are $(32-4)^2 = 784$ locations to consider, not to mention the large number of color pattern combinations ($256^{4\times4\times3}$ where 256 denotes the pixel value choices from 0 to 255 and 3 denotes the RGB channels).

We divide the initial search into two stages as shown in Algorithm 2. In the first stage (lines 2-10), we mainly focus on finding a reasonable trigger location. We then focus on finding the optimal trigger pattern in the second stage (lines 11-14).

To determine whether a location is good, it is common to randomly sample a set of patterns at that location and examine the average ASR for this set. However, it is inefficient to traverse all possible locations and perform the aforementioned testing. We observe that trigger positions are not totally independent, meaning that a valid trigger likely achieves a high ASR for many positions within a region due to the continuity of deep learning models. Therefore, instead of blind search, we apply a local search near the (possibly) best trigger position after a few iterations (lines 4-6). We use $\lambda = 0.7$ and the neighborhood range $[-2, 2]$ in the paper and study other options in Appendix D.

As mentioned earlier, the quality of a trigger location also relies on sampled patterns (for the measurement). It is straightforward to draw patterns from a uniform distribution $\mathcal{U}(0,1)$ (see Figure 2a), meaning every pixel in the trigger pattern has the same probability of being from 0 to 1. (Here, we normalize the pixel values to range $[0,1]$ for simplicity.) Most pattern pixel values fall in the normal image distribution (e.g., $[0.2, 0.8]$), to which the subject model may not be sensitive. We observe that sampling the pattern from extreme values can better cause the misclassification. For example, we use uniform distribution $\mathcal{U}(-4,4)$ and then clip the values to $[0,1]$, which is shown in Figure 2b. Observe the two peaks at the two ends (extreme values). By applying this distribution for trigger initialization, we can improve the ASRs of initial triggers. Figure 3 compares the ASRs of initial triggers using

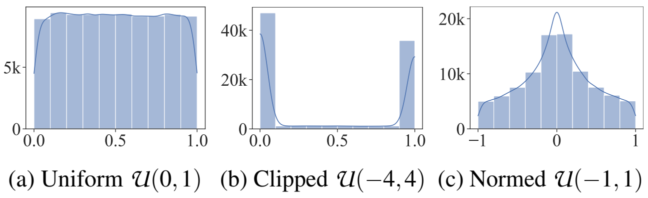


Figure 2: Uniform distribution. The x-axis shows the sampled value and the y-axis the frequency of the corresponding value.

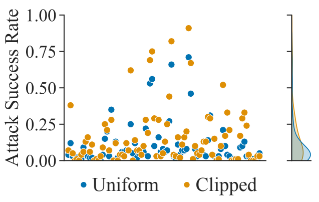


Figure 3: ASR of initial triggers using different uniform distributions. The x-axis denotes the class pair in CIFAR-10 and the y-axis is the attack success rate.

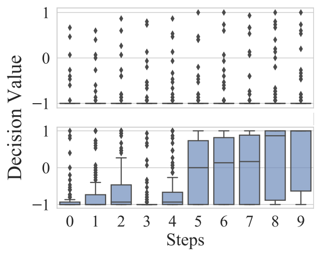


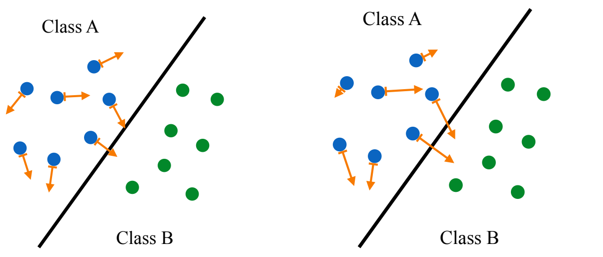
Figure 4: Average decision values during gradient estimation for the baseline (top) and ours (bottom). The x-axis is the generation step. Decision 1 means the target prediction.

different distributions. The y-axis denotes the ASR. Each dot in the figure denotes a trigger for a class pair in CIFAR-10 (flipping the prediction from one class to another), where the blue ones are sampled from the standard uniform distribution $\mathcal{U}(0,1)$ and the orange ones from the clipped uniform distribution $\mathcal{U}(-4,4)$ shown in Figure 2b. The right-hand side sub-figure shows the ASR distributions for the two sets of triggers. Observe that there are more orange dots with high ASRs than the blue ones. The orange ASR distribution is shifted towards a higher ASR value compared to the blue one. Therefore, we use the clipped uniform distribution for initializing trigger patterns (line 8 in Algorithm 2). Note that the initialized trigger pattern here also serves as the starting point for further trigger generation in Algorithm 1. It is hence important to have a pattern with a relatively high ASR.

After the initial search, a location $\boldsymbol{m}$ is found and fixed. The next stage is to search for a good trigger pattern at the location.

5.2 Neighbor Selection

According to Algorithm 1, HARDBEAT may decide to interpolate between a historical data point with a top-k ASR and one of its neighbors. In this subsection, we discuss how the neighbor is selected. Specifically, we compute the (cosine) similarity between every pair of (queried) triggers during the attack and maintain a similarity matrix $\boldsymbol{G}$, which is used to measure their distances. We also need to consider the ASR when choosing the neighbor (in order to form a ridge). The



(a) Baseline estimation (b) Importance-aware estimation

Figure 5: Illustration of gradient direction estimation

selection criterion is formulated as follows.

$$\arg\max_{\mathbf{\Delta}'} \boldsymbol{G}[\mathbf{\Delta}][\mathbf{\Delta}'] \cdot ASR[\mathbf{\Delta}'], \tag{2}$$

$$\boldsymbol{G}[\mathbf{\Delta}][\mathbf{\Delta}'] = \big(\cos(\mathbf{\Delta}, \mathbf{\Delta}') + 1\big)/2, \tag{3}$$

where $\mathbf{\Delta}$ is the current trigger and $\mathbf{\Delta}'$ its neighbor; $\cos(\cdot)$ calculates the cosine similarity between two vectors.

5.3 Importance-aware Gradient Estimation

In this section, we discuss how to estimate the gradient direction for perturbing a trigger (at line 21 in Algorithm 1). Existing attacks [13, 25, 36] use the Monte Carlo estimation to approximate the direction of gradient. In specific, they apply a set of random noises on the trigger and see whether these noises can change the prediction. The average of these perturbation directions are averaged as the final direction for mutating the trigger. However, such a design is not efficient. Figure 5a illustrates the concept. The blue and green circles denote samples in two classes A and B. The black straight line denotes the decision boundary of the victim model. The orange arrows show the gradient directions estimated for individual samples and the length denotes the magnitude. The attack goal is to move all the blue samples across the decision boundary to class B. Observe that the arrows in the top point to the opposite direction of those on the left and in the bottom. There are only a very few arrows pointing towards the decision boundary. Most gradient directions are canceled out with each other and the attack result can hardly be improved.

We propose an importance-aware direction estimation method. We adaptively adjust the weights for different samples, based on how these sampled noises can cause the target-class misclassification. Specifically, we leverage the improvement on average decision value (1 means the target prediction and -1 otherwise) of the current direction over the one from the previous trigger generation step as the direction importance. As illustrated in Figure 5b, the lengths of those arrows pointing towards the decision boundary are greatly increased while the others are reduced. The overall direction is hence more focused towards the boundary compared to the baseline in Figure 5a. The bottom plot in Figure 4 shows the results of applying our importance-aware estimation method and the top

Algorithm 3 Importance-aware Gradient-direction Estimation

```
1:  function GRADESTIMATE(decision function $f$, input $\boldsymbol{x}$, target label $y_t$, trigger $\mathbf{\Delta}$)
2:      Initialize historical average decision values $hist_\omega$ for each sample
3:      $\boldsymbol{m}, \boldsymbol{p} \leftarrow \mathbf{\Delta}$
4:      for $i$ in $0 \ldots n$ do                                ▷ Number of samples $n$
5:          $[\boldsymbol{p}']_b \leftarrow$ Add a set of random noises to $\boldsymbol{p}$ and clip to $[0,1]$
6:          for $j$ in $0 \ldots b$ do
7:              $\boldsymbol{x}'_{ij} = (1-\boldsymbol{m}) \odot \boldsymbol{x}_i + \boldsymbol{m} \odot \boldsymbol{p}'_j$
8:              $\boldsymbol{\eta}_j = \boldsymbol{x}'_{ij} - \boldsymbol{x}_i - \boldsymbol{m} \odot \boldsymbol{p}$        ▷ Perturbation by the trigger
9:          end for
10:         $[y']_b \leftarrow f(\boldsymbol{x}', y_t)$ ▷ Whether the prediction is the target $\in \{1, -1\}$
11:         $\omega = mean([y']_b)$
12:         $d_\omega = \omega - hist_\omega[i]$        ▷ Decision value compared to the history
13:         if $d_\omega > 0$ then
14:             $d_\omega = exp(d_\omega)$
15:             if $\omega$ is 1 then $d_\omega = d_\omega / \gamma$
16:             end if
17:         else
18:             $d_\omega = ln(d_\omega + 3)$
19:         end if
20:         if $\omega$ equals to 1 or -1 then        ▷ Estimate the gradient direction
21:             $\boldsymbol{\delta}_i = \omega \cdot mean([\boldsymbol{\eta}]_b)$
22:         else
23:             $\boldsymbol{\delta}_i = mean(([y']_b - \omega) \cdot [\boldsymbol{\eta}]_b)$
24:         end if
25:         $\boldsymbol{\delta}_i = d_\omega \cdot \boldsymbol{\delta}_i / ||\boldsymbol{\delta}_i||_2$
26:         Update $hist_\omega[i] \leftarrow \omega$
27:     end for
28:     return $mean([\boldsymbol{\delta}]_n)$
29: end function
```

plot shows the baseline. The x-axis denotes the trigger generation steps and the y-axis the average decision value for a set of samples (the larger, the better). Observe that the average decision values are significantly improved with the value at the final step close to 1 (the target prediction). For a class pair cat→dog from CIFAR-10, the baseline estimation only obtains 20.9% attack success rate, while our method can achieve 80.1% ASR, delineating the effectiveness of our estimation.

Algorithm 3 formally describes our importance-aware estimation of the gradient direction for perturbing the patch trigger. It computes the importance values of n samples and then the gradients. For each sample i, it adds random noises to the trigger and creates b samples with perturbed triggers (lines 6-9). The importance of i is updated by comparing its averaged decision value ω with the prior (lines 10-19), which is used to compute the gradients (lines 20-26). The order of the n samples can be shuffled at each attack iteration. The following explains the details and intuitions for important parts of the algorithm.

As we aim to estimate the landscape near the given patch pattern $\boldsymbol{p}$ in Algorithm 3, the added noises shall be small. Otherwise, the perturbed pattern will be far from the current one, which cannot be used to accurately approximate the gradient direction. We hence use a normalized uniform distribution $u \cdot exp(u-1), u \sim \mathcal{U}(-1,1)$ (at line 5), whose shape is shown in Figure 2c. Other distributions such as Gaussian distribution can also be employed here. For those additive noises, we measure whether they can cause the target prediction using a decision function (line 10), which yields 1 for the target prediction and -1 otherwise. Using -1 for the non-target prediction is because although going along the opposite direction of the additive noise may not imply the desired target prediction, it is still the best option [13]. Using value 0 may waste the opportunity of trying other directions and a small negative value leads to limited exploration. The empirical comparison of using different values is shown in Appendix D.

As illustrated in Figure 5b, we aim to lift the importance of directions that can lead to misclassification and reduce others, which is realized using the equations at lines 14 and 18, respectively. Many times, the direction from already-misclassified samples may overwhelm the direction from other samples. We decrease their importance by a factor γ (line 15). We use $\gamma = 5$ in the paper and evaluate other values in Appendix D. The direction improvement d_ω is the difference between the current decision value and the prior, which can be as small as -2 $(= -1-1)$. The value 3 at line 18 is to prevent $ln(\cdot)$ being smaller than 0. The resultant d_ω will be mostly smaller than 1, denoting a small weight on gradient directions that cannot increase the chance of producing the target label.

The gradient direction for each sample is estimated by the weighted average of additive noises (lines 20-24). Lines 20-21 handle the case where all the noises lead to misclassification or not. If only some perturbations cause the target prediction, we subtract the average decision value from individual ones and multiply them with their corresponding noises at line 23. This is to ensure the weights on different noises do not deviate too much away from the average as it denotes the quality of the current trigger. Note that the case in lines 20-21 cannot be combined with line 23 as $\omega = mean([y']_b)$ (line 11). The term $[y']_b - \omega$ will be zero, which wastes the estimated gradient direction that leads to the target prediction.

6 Evaluation

The evaluation is conducted on various datasets and model architectures. We compare the attack performance of HARDBEAT with eight baseline attacks. We also carry out our attack on two online commercial services. To study the attack factors, we consider different numbers of attack samples, trigger sizes and shapes. A set of ablation studies are also conducted.

6.1 Experiment Setup

Datasets & Models. We adopt four popular image datasets, including CIFAR-10 [35], SVHN [43], STL-10 [19], and GTSRB [52] (details in Appendix A). For each dataset, we evaluate the attack performance on two models with different architectures. For CIFAR-10, we use ResNet18 and VGG11, with an accuracy of 93.07% and 92.39%, respectively. For SVHN, a ResNet18 model with 95.03% accuracy and a ResNet34 model with 95.42% accuracy are utilized. We employ GoogleNet and DenseNet121 for STL-10, which have 71.03% and 69.64% accuracy, respectively. For GTSRB, MobileNet_v2 with 97.70% accuracy and ResNet50 with 96.60% accuracy are considered for evaluation.

Online Commercial Services. Other than the experiments on local models, we also conduct experiments on two online commercial services, Microsoft Azure [41] and Clarifai [18]. This is to demonstrate the feasibility of launching HARDBEAT in real-world settings. We make use of the SVHN dataset and upload it to the two commercial services, respectively. We then train one image classifier using each service. The model trained by Azure has an accuracy of 94.8% and the one by Clarifai has 89.0% accuracy. We use the prediction API provided by each service to query the model for carrying out hard-label black-box universal adversarial patch attacks.

Baselines. To our knowledge, there does not exist a hard-label black-box universal adversarial patch attack in the literature. We adapt eight existing black-box attacks to our setting. In particular, we consider three state-of-the-art hard-label black-box adversarial attacks (HSJA [13], GRAPHITE [25], and SparseEvo [57]), three well-known soft-label black-box attacks (Bandits [30], SPSA [51], and Sparse-RS [20]), and two baselines based purely on MCMC [27] and GA [3] respectively. Details of these baselines are described in Appendix B.

Metrics. Two standard metrics are used for evaluating attack performance. Attack success rate (ASR) measures the percentage of clean samples injected with the same generated universal adversarial patch that are predicted as the target label by the victim model. The ASR is measured on the whole test dataset. We evaluate the attack performance on every class pair of a victim model. That is, we generate a patch trigger that can cause misclassifications for images from a victim class to a target class. We then count the number of class pairs that the attack can achieve a certain ASR. The L^0 norm of the generated trigger is utilized to measure the stealthiness.

Attack Settings. For all the attacks, we assume 100 images from the victim class are available. As some classes in the GTSRB dataset have less than 100 test images, we use 50 images for attack. GRAPHITE and SparseEvo have access to a target-class image to conduct the attack. For the remaining baselines and HARDBEAT, a size of 7×7 patch (taking up 4.79% of the input) is used. We do not constrain the number of queries for GRAPHITE as its binary search requires a fixed number of queries. Otherwise, it cannot reduce the trigger size. For the remaining attacks, we allow 50k (25k for GTSRB) model queries in total. We study the impact of different attack factors in Section 6.3.

6.2 Attack Performance

6.2.1 Comparison with Hard-label Attacks

We report the main attack results for HARDBEAT in Table 1 and compare attack performance with prior works. We use a fixed trigger size that occupies 4.79% of the input. For GRAPHITE and SparseEvo that generate larger triggers, we reduce the sizes of their triggers. The results on their full-size triggers are reported in Table 9 (see Appendix C). For all the models except those on GTSRB, the attack is conducted on all class pairs (90 pairs in total). The GTSRB dataset has 43 classes, constituting 1,806 pairs. Baseline GRAPHITE takes around 12 minutes to generate a trigger for a class pair. Running on all pairs requires 15 days of computation. We hence use a random seed `714725708` to choose 200 pairs for evaluation. We also show the results on all GTSRB class pairs for attacks other than GRAPHITE in Table 8 (see Appendix C).

Columns 4-12 in Table 1 present the number of class pairs whose attack success rate (ASR) is above the corresponding threshold shown in the table head. The more class pairs are in the higher ASR range, the better the attack performance is. Observe that baseline attacks cannot find any >90% ASR trigger in many cases. For example, on CIFAR-10 with ResNet18, all three baselines have zero pair with >90% trigger. HardBeat can successfully attack 7 class pairs with >90% ASR. Furthermore, HardBeat has 11 pairs with >80% ASR, whereas baselines still do not have any successful pairs. For lower ASR ranges such as >50%, HardBeat has significantly more attacked pairs than existing attacks (25 vs. the best result 7 by HSJA). On GTSRB, the baselines are able to generate a few high-ASR triggers. For ResNet50, HSJA has 8 pairs with >90% ASR, GRAPHITE and SparseEvo both have two pairs. This may be because the traffic signs are more structured and the victim model only needs to learn simple features. Baseline attacks hence are able to craft feasible patch triggers. HardBeat greatly outperforms them with 2 times more successful pairs achieving >90% ASR. Overall, existing hard-label attacks can hardly generate high-ASR universal adversarial patches. HardBeat is able to construct successful high-ASR triggers for more than twice pairs compared to baselines. Although the total number does not cover all class pairs, it is still critical to these models as it is analogous to finding multiple vulnerabilities in a software. Any of the identified problems can be exploited.

Table 1: Attack performance in comparison with hard-label attacks when the universal adversarial patch occupies 4.79% of the input. Columns 4-12 denote the number of class pairs whose attack success rate is above the corresponding threshold. All the attacks except GRAPHITE take 25k queries on GTSRB, and 50k on other datasets. GRAPHITE on average takes 54k queries on GTSRB, and 120k-160k queries on the remaining datasets.

D	M	Attack	>90%	>80%	>70%	>60%	>50%	>40%	>30%	>20%	>10%
CIFAR-10	ResNet18	HSJA	0	0	3	4	7	8	8	9	14
		GRAPHITE	0	0	0	0	1	3	7	10	18
		SparseEvo	0	0	0	0	0	0	0	2	14
		HardBeat	7	11	17	19	25	33	40	46	60
	VGG11	HSJA	0	3	5	8	9	10	13	19	21
		GRAPHITE	1	3	3	3	5	8	9	13	17
		SparseEvo	0	0	0	0	0	2	2	5	13
		HardBeat	12	20	21	22	26	28	35	40	48
SVHN	ResNet18	HSJA	0	0	2	4	9	15	18	25	35
		GRAPHITE	0	0	0	0	1	2	6	11	31
		SparseEvo	0	0	0	0	0	3	14	31	55
		HardBeat	4	9	14	21	34	48	57	64	74
	ResNet34	HSJA	0	1	2	8	12	16	21	30	40
		GRAPHITE	0	0	0	0	1	1	6	14	32
		SparseEvo	0	0	0	0	0	1	8	26	58
		HardBeat	3	12	20	23	33	43	50	62	70
STL-10	GoogleNet	HSJA	4	5	8	12	12	12	18	22	38
		GRAPHITE	0	0	0	1	1	5	6	10	15
		SparseEvo	0	0	0	1	2	3	4	13	23
		HardBeat	12	16	22	28	35	44	44	50	58
	DenseNet	HSJA	0	2	3	4	6	7	8	14	24
		GRAPHITE	0	0	0	0	0	2	3	6	14
		SparseEvo	0	0	1	2	3	7	8	12	20
		HardBeat	3	4	9	14	21	27	31	45	60
GTSRB	MobileNet	HSJA	4	4	9	10	12	15	16	27	31
		GRAPHITE	1	1	2	3	4	7	13	20	31
		SparseEvo	0	0	1	5	10	17	25	32	47
		HardBeat	8	17	22	29	38	46	56	64	77
	ResNet50	HSJA	8	8	11	14	17	18	24	26	31
		GRAPHITE	2	4	4	6	9	13	15	22	37
		SparseEvo	2	6	8	11	17	21	25	32	51
		HardBeat	20	28	34	44	49	53	65	74	92

Detailed Attack Results. The previously discussed table only shows the number of pairs above a certain ASR threshold. Here, we report the exact ASR for each class pair. We show the attack results on CIFAR-10 with ResNet18 in Figure 6 and on other datasets and models in Appendix C due to page limit. Each cell in the heat map denotes the ASR for a generated universal adversarial patch flipping all the test samples from a victim class (row) to a target class (column).

Observe that HSJA has a very limited number of pairs having a reasonable ASR. The best ASR it has is 77% for pair frog$\rightarrow$bird. For the 7 pairs with >50%, HSJA outperforms GRAPHITE and SparseEvo. This denotes the gradient approximation method in HSJA is able to find some local minima compared to the other two baselines. Both GRAPHITE and SparseEvo have low attack performance, with most pairs having less than 20% ASR. These two attacks mainly rely on the target-class image. In the adversarial attack setting, they only need to find a few pixels to flip the prediction of a victim image. This is fairly easy as existing adversarial attacks show only one pixel can cause misclassification [53]. In the universal adversarial patch attack scenario, however, the problem is much harder. The attack needs to find a persistent trigger that can flip the predictions for a set of samples. The pixels from the target-class image may not have such a property. Even with the boosting of gradient estimation as in GRAPHITE, it still cannot achieve a good attack performance.

HardBeat has the best results compared to existing attacks. Most successful pairs are for target classes car, bird, and cat. This may be due to the victim model only learning low-level features such as a few pixel colors. HardBeat can easily find such features and construct a successful trigger. There still exist pairs for which HardBeat has near-zero ASR. By inspecting the generation process, we find the sampled random noises are not able to cause misclassification

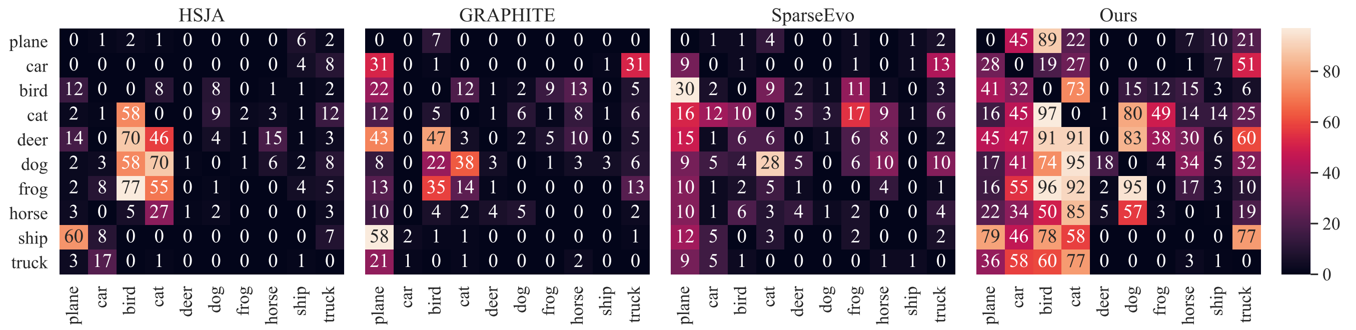


Figure 6: Attack results on CIFAR-10 with ResNet18. GRAPHITE on average takes 128k queries. Other attacks take 50k queries. Each cell denotes ASR for a universal patch trigger flipping all the samples from a victim class (row) to a target class (column).

Table 2: Attack performance in comparison with adapted soft-label attacks with a patch size of 4.79% of the input.

D	M	Attack	>90%	>80%	>70%	>60%	>50%	>40%	>30%	>20%	>10%
CIFAR-10	ResNet18	Bandits	0	0	0	0	3	3	4	7	14
		SPSA	1	3	3	3	5	7	8	10	19
		Sparse-RS	3	5	6	7	9	10	11	13	23
		HARDBEAT	7	11	17	19	25	33	40	46	60
SVHN	ResNet18	Bandits	0	0	1	1	2	2	4	12	25
		SPSA	1	1	1	2	6	8	12	20	29
		Sparse-RS	0	0	1	3	4	5	6	10	15
		HARDBEAT	4	9	14	21	34	48	57	64	74

on input images. Without the guidance of positive directions (from misclassification), the black-box attack has no information on how to mutate the trigger. We tried leveraging a target-class image as the guidance. The result is still not promising. For these pairs, a white-box attack with access to the model weight parameters also has limited ASR. We think they are intrinsically more robust. Using a larger number of queries or a larger trigger size may increase ASR. We believe this is a big challenge and worth exploring in future work.

6.2.2 Comparison with Adapted Soft-label Attacks

Soft-label black-box attacks leverage the change of output label probabilities as the guidance to approximate the direction for perturbing the input. We adapt them to the hard-label setting where they can only observe the change of the predicted label. Table 2 reports the attack results for three adapted soft-label attacks. Observe that these attacks have a limited number of high-ASR (>90%) triggers. Bandits and SPSA have no more than one trigger that achieves >90% ASR. Sparse-RS is slightly better on CIFAR-10 with 3 high-ASR triggers but ineffective on SVHN. The observation is the same for the results in other ASR ranges. This is understandable as these attacks were originally designed for the soft-label setting, where rich information such as label probability change can be used for crafting the perturbation. It is much harder in the hard-label setting as the attack can only observe whether the prediction is the target label. Overall, HARDBEAT has significantly better attack performance with more than twice successful triggers than these baselines.

Table 3: Results of baseline attacks based purely on MCMC and GA with a patch size of 4.79% of the input.

D	M	Attack	>90%	>80%	>70%	>60%	>50%	>40%	>30%	>20%	>10%
CIFAR	RsNet18	MCMC	0	1	3	3	4	6	9	9	14
		GA	0	1	2	3	4	5	6	9	22
		HARDBEAT	7	11	17	19	25	33	40	46	60
SVHN	RsNet18	MCMC	0	2	2	5	6	11	16	20	32
		GA	0	1	2	3	7	10	18	25	43
		HARDBEAT	4	9	14	21	34	48	57	64	74

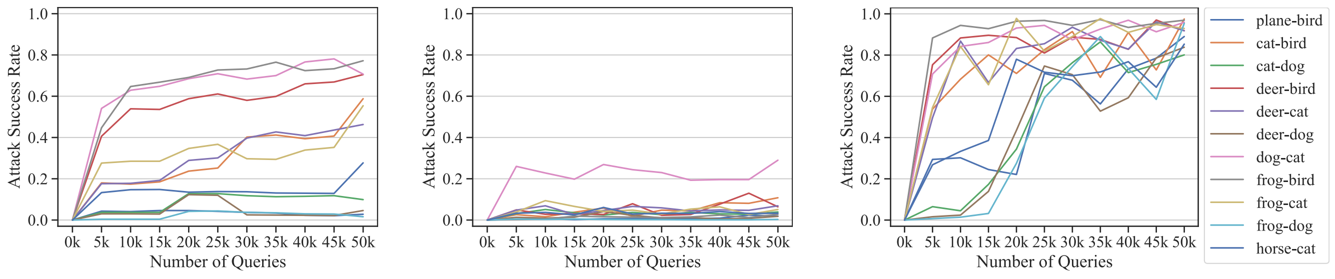
6.2.3 Comparison with Other Baselines

HARDBEAT takes advantage of both MCMC [27] and GA [3] in generating universal adversarial patches. Here, we study the attack performance of purely using each base method and report the results in Table 3. As expected, these baselines have limited attack performance with zero >90% trigger. HARDBEAT generates 3 times more triggers in most ASR ranges. This demonstrates the necessity of considering both focused/directed search as in MCMC and historical data points as in GA for achieving good attack results in hard-label black-box universal adversarial patch attack.

6.3 Studying Attack Factors

In this section, we study three major attack factors, namely, number of model queries, number of attack samples, and trigger size and shape. We use existing hard-label attacks (HSJA and SparseEvo) as baselines in the study. The observations on adapted soft-label attacks are similar and hence elided.

Number of Model Queries. In the last section, we limit the number of queries to 50k (25k for GTSRB) for all the attacks. Here, we study the relation between number of queries and the attack success rate (ASR). As we have seen earlier that there exist pairs having low ASR, it is not informative to study those pairs. We hence select pairs with >80% ASR as the subject. Due to page limit, we use the results on CIFAR-10 with ResNet18 as an example and refer readers to Appendix C for results on other datasets. Observe that in Figure 7a, for pairs deer→bird, dog→cat, and frog→bird, HSJA is able to quickly find good patch triggers and have >50% ASR using



(a) HSJA (b) SparseEvo (c) HardBeat

Figure 7: Attack success rate versus number of model queries on CIFAR-10 with ResNet18

Table 4: Attack performance using different numbers of attack samples

#Samples	Attack	>90%	>80%	>70%	>60%	>50%	>40%	>30%	>20%	>10%
10	HSJA	1	3	4	4	6	11	14	14	19
	SparseEvo	0	0	0	1	2	4	5	14	33
	HardBeat	2	4	8	10	12	15	19	25	35
30	HSJA	2	5	5	8	9	10	11	13	19
	SparseEvo	0	0	0	0	0	2	6	12	31
	HardBeat	4	7	9	15	19	22	29	37	54
50	HSJA	2	3	5	7	9	9	9	10	13
	SparseEvo	0	0	0	0	0	1	1	8	23
	HardBeat	3	8	13	14	22	26	33	42	56

just 10k queries. For other pairs on the bottom of the figure, it however fails to generate successful triggers even given the 50k query limit. This demonstrates that HSJA works when the surrounding area of the minima is flat. If the optimal region has a plummet like the one shown in the middle of Figure 1b, HSJA is not able to find a good trigger. SparseEvo has very low ASR when using 50k queries and the results are similar when fewer queries are used. For HardBeat in Figure 7c, half of the studied pairs have more than 70% ASR with only 10k queries. For harder cases such as pair frog→dog (light blue line), HardBeat is able to gradually improve the trigger with the increase of queries. It achieves 60% ASR when 25k queries are used and 95% ASR with 50k queries. These harder cases may be due to a more rugged loss surface, making HardBeat take longer to reach the optimal. This demonstrates the effectiveness of our gradient approximation algorithm compared to HSJA. HardBeat is able to maneuver on a wrinkled search surface and finally reach the optimal point as illustrated in Figure 1d.

Number of Attack Samples. The goal of universal adversarial patch attack is to cause misclassification for as many samples as possible. To achieve this, the attack needs to leverage a set of samples such that the trigger can be persistent and robust on unseen test images. We used 100 samples in previous experiments. Here, we study the effect of using smaller numbers of samples, denoted as attack samples. The study is conducted on CIFAR-10 with ResNet18 and the results are shown in Table 4. The first column shows the number of attack samples. With only 10 attack samples, HSJA is able to

Table 5: Attack performance with different patch shapes (possible rectangle shapes that consist of <5% image pixels) and multiple patches (two 2.5%-pixel square patches)

Trigger	Attack	>90%	>80%	>70%	>60%	>50%	>40%	>30%	>20%	>10%
3 × 3	HSJA	0	0	0	0	0	1	3	4	8
	HardBeat	0	1	1	3	5	7	11	12	23
5 × 5	HSJA	0	0	1	3	3	4	5	6	9
	HardBeat	3	5	8	11	13	17	21	26	36
7 × 7	HSJA	0	0	3	4	7	8	8	9	14
	HardBeat	7	11	17	19	25	33	40	46	60
3 × 16	HSJA	0	0	2	2	6	7	8	10	16
	HardBeat	3	7	12	18	25	29	36	44	53
5 × 10	HSJA	1	1	2	3	6	6	8	8	16
	HardBeat	5	8	13	15	20	26	36	42	55
Two 5 × 5	HSJA	0	2	3	5	6	7	9	12	18
	HardBeat	3	9	17	23	24	27	35	40	47

generate 4 triggers with >70% ASR. HardBeat has twice as many pairs. SparseEvo has only 1 pair with >60% ASR. When increasing the number of attack samples, the attack performance of HSJA and SparseEvo slightly decreases. This is because more samples mean more diverse input features. Baselines are not able to balance among those features when estimating the gradient direction. HardBeat, on the other hand, has better attack performance when more samples are used. The number of pairs with >70% ASR is increased from 8 to 13. With more samples considered, our novel gradient approximation is able to leverage more information to better balance across multiple samples. We also show the ASR of HardBeat for the pairs from Figure 7 when different numbers of attack samples are used. Please see Appendix C.

Trigger Size and Shape. We used a fixed shape with size of 7 × 7 as the trigger for HSJA and HardBeat previously. We study their attack performance using different sizes and shapes of the trigger. As SparseEvo dynamically determines the size and shape of the trigger during generation, we exclude it here. Table 5 reports the results. Increasing the trigger size from 9 (3 × 3) to 49 (7 × 7), both HSJA and HardBeat have better attack results, which is expected. We also change the shape of the trigger from a square (7 × 7) to different rectangles (3 × 16 and 5 × 10). We observe variations on the attack performance but most are similar. HSJA has a slightly

Table 6: Attacking online services. The patch trigger occupies 4.79% of the input. All the attacks take 240 queries.

Service	Attack	0->4	0->5	1->2	1->7	2->3	3->2	3->9	4->1	6->4	9->2	Avg.
Azure	HSJA	76%	74%	38%	63%	53%	39%	58%	83%	81%	38%	60%
	HARDBEAT	90%	76%	59%	74%	70%	63%	75%	88%	84%	57%	74%
		0->4	1->4	2->4	3->4	5->3	6->3	7->4	8->3	8->4	9->4	Avg.
Clarifai	HSJA	70%	87%	53%	55%	32%	30%	63%	27%	48%	60%	53%
	HARDBEAT	90%	93%	74%	85%	65%	54%	76%	56%	78%	71%	74%

better result using the 5×10 trigger and HARDBEAT has the best result with the square trigger. In addition, we study a trigger with two square patterns (two 5×5). It produces more >80% pairs for HSJA and more >60% pairs for HARDBEAT compared to the 7×7 trigger. Overall, a larger trigger size can improve attack performance and different trigger shapes have marginal effect. We also show the ASR of HARDBEAT for the 11 pairs from Figure 7 when different trigger sizes are used. Please see Figure 10 in Appendix C.

6.4 Attacking Online Services

To demonstrate the feasibility of launching attacks in the real world, we employ two online commercial services Azure and Clarifai, and apply HSJA and HARDBEAT to attack their trained models. Note that these models were not deployed and can only be accessed by authors (through the prediction API). It hence does not pose a real societal threat. We use the same trigger size and shape (7×7) but with a reduced number of attack samples (10 images) and model queries (240 queries). This is to show a practical attack scenario. We test the ASR for generated triggers on 100 test images. Table 6 reports the attack results of HSJA and HARDBEAT on 10 class pairs. Observe that HARDBEAT can achieve 90% ASR on pair $0 \to 4$ on both Azure and Clarifai. It also has 93% ASR on pair $1 \to 4$ on Clarifai. This shows HARDBEAT is very effective in generating high-ASR triggers with such a limited number of attack samples and queries, delineating its practicality. On average, HARDBEAT has 74% ASR on both platforms, substantially outperforming baseline HSJA (60% on Azure and 53% on Clarifai).

6.5 Ablation Study

There are three main components in the design of HARDBEAT: (1) using historical data points with high ASR, (2) using interpolated data points of previous local minima, and (3) the importance-aware gradient estimation. Our ablation study shows the interpolation between previous local minima and the importance-aware gradient estimation have large impacts on the final attack results, delineating their importance. We also study the impact of different choices for the decision value denoting the non-target prediction (used in Algorithm 3). Value -1 has the best results. See details in Appendix D.

There are several hyper-parameters in HARDBEAT including λ in Algorithm 2 (line 4) that balances the exploration of more random positions and the vicinity of the best trigger position; the neighborhood range used in Algorithm 2 (line 5) to mutate the current best trigger position; the parameter k for selecting top-k historical triggers in Algorithm 1 (line 7); and γ for adjusting the impact of directions of already-misclassified samples in Algorithm 3 (line 15). Our experimental results show different hyper-parameters have a slight impact on the final attack performance. The overall results are similar. Please see details in Appendix D.

7 Countermeasures

We employ a state-of-the-art certifiable defense against adversarial patches PatchCleanser [61], a query-based defense against black-box attacks Blacklight [37], and a universal adversarial patch detection approach SentiNet [17] for measuring HARDBEAT's attack performance.

7.1 Certifiable Defense

Certifiable defense aims to have a correct prediction for a given input regardless how the input is adversarially perturbed within the threat model (e.g., adding a fixed-size patch at random locations). PatchCleanser [61] is a state-of-the-art certifiable defense against adversarial patches. It uses double-masking to certify the prediction. Specifically, it traverses all the positions on the input and adds a mask (i.e., a black patch) for each position. If the predictions are consistent, it considers this the final output. Otherwise, PatchCleanser applies a second round of masking (on top of the first round mask). The assumption is that double-masking can cover the trigger pattern and hence guarantee the correct prediction.

We apply PatchCleanser to certify all class pairs on CIFAR-10 and SVHN. Specifically, we apply HARDBEAT to generate triggers for each class pair, and then run PatchCleanser to certify the trigger-injected images. The average certified robust accuracy is 0.17% for CIFAR-10 and 0.0% for SVHN. This is because PatchCleanser is not designed for adversarial patches with a large trigger size. An important assumption is that double-masking does not change the predictions of clean inputs. However, in order to cover the trigger pattern, a large mask is needed for PatchCleanser, which cannot guarantee the correct prediction for clean samples. As the double-masking can cover the trigger, we report the prediction accuracy of the victim model before and after applying PatchCleanser. The results show it cannot improve prediction accuracy in most cases. Please see details in the supplementary material [1].

7.2 Query-based Defense

Query-based defense is specifically designed for detecting malicious queries by black-box attacks. Blacklight [37] is a

Table 7: Bypassing query-based defense. The patch trigger occupies 4.79% of the input. All the attacks take 240 queries.

Metric	0->4	1->4	2->4	3->4	5->3	6->3	7->4	8->3	8->4	9->4	Avg.
ASR	85%	92%	78%	80%	47%	41%	97%	48%	64%	68%	70%
Detection	0.0%	0.0%	0.0%	0.0%	0.0%	1.3%	0.0%	0.4%	0.4%	0.0%	0.2%

state-of-the-art defense technique that detects black-box attacks based on the similarity among queries. Specifically, it uses a fixed window size and traverses all the positions on the input to obtain a set of image segments. It then generates a hash value for each image segment. A hash representation is then created for each incoming query. A new query is regarded as adversarial if its hash representation matches any prior queries. Blacklight heavily relies on the similarity between two images in consecutive queries. Black-box adversarial attacks need to gradually reduce the adversarial perturbation during the attack, masking consecutive query images very similar to each other. Universal adversarial patch attacks, on the other hand, do not rely on the similarity among attack samples. Instead, a diverse set of images may even improve attack performance. There are two options. We can either augment attack samples by adding random noises or use different images for each query. We choose the former one and conduct the experiment.

Table 7 shows the results, with the first row denoting the ASR by HARDBEAT and the second row denoting the detection rate by Blacklight. Observe that Blacklight can hardly detect the attack. There are only 3 out of 10 class pairs having non-zero detection rate. Further inspection shows fewer than 3 queries are detected as malicious by Blacklight and all of them happened after 160 queries (240 queries in total), meaning HARDBEAT almost finished the attack. Figure 8 shows original inputs (row 1) and trigger-injected images in two consecutive generation iterations (rows 2-3). Observe that the images in two iterations are very similar and yet Blacklight cannot detect them. Our observation is consistent with the literature [24] that Blacklight can be evaded by adding noise/changing noise variance on the input. In addition, compared to the results reported in Table 6, the ASRs actually increase for pairs 2→4 (from 74% to 78%) and 7→4 (from 76% to 97%). Adding random noises on attack samples can indeed improve the attack performance of HARDBEAT. Therefore, Blacklight can hardly defend against HARDBEAT.

7.3 Universal Adversarial Patch Detection

Universal adversarial patch detection approaches focus on inspecting and rejecting a given input injected with a universal patch trigger. One of the well-known detection methods SentiNet [17] cuts out the feature on a given input A that is responsible for the prediction, and pastes it on a set of clean inputs B. It inspects the confidence of cut A and the prediction of B to determine whether A is a malicious input. We apply SentiNet on attacked samples by HARDBEAT for all >60%

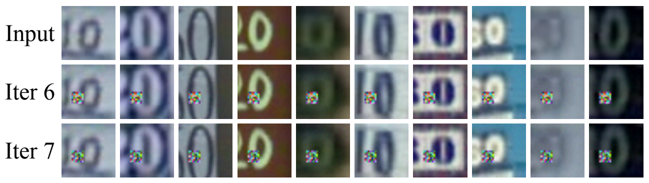


Figure 8: Query images from two consecutive attack iterations (iterations 6 and 7) when evading Blacklight on Clarifai. Row Input shows the original 10 images used for the attack.

ASR class pairs from CIFAR-10. The average detection accuracy is 50.53%. According to the original paper [17], the detection rate of SentiNet on attacked samples by a white-box counterpart of HARDBEAT (Adversarial Patch [8]) is 98.5%. This is because HARDBEAT's patch regions have weaker correlations with the target class due to its black-box nature.

7.4 Alternative Defense

Adversarial training is one of the most effective defense techniques against adversarial attacks [40,49,59]. The problem of current adversarial training is that it may greatly affect normal functionality (causing non-trivial accuracy degradation). Existing works [54,55] demonstrate that fine-tuning the subject model using generated triggers can improve model robustness while maintaining the accuracy. It is possible to leverage HARDBEAT to generate universal adversarial patches and then apply existing methods to harden the model. We leave the experimental exploration to future work.

8 Conclusion

Deep learning models are susceptible to universal adversarial patch attacks. In this paper, we show that even under the most restricted scenario (with access only to the predicted label), the attacker can still generate high-ASR triggers. Our proposed attack HARDBEAT achieves 74% ASR on two online commercial services, delineating the severity of the problem. The evaluation on existing state-of-the-art defenses reveals that more advanced techniques are urgently needed to defend against strong attacks such as the one proposed in this paper.

Acknowledgments

We thank the anonymous reviewers for their constructive comments. This research was supported, in part by IARPA TrojAI W911NF-19-S-0012, NSF 1901242 and 1910300, ONR N000141712045, N000141410468 and N000141712947. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

References

[1] HardBeat. https://github.com/Gwinhen/HardBeat, 2023.

[2] Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. In *ECCV*, pages 484–501, 2020.

[3] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc., 1998.

[4] Philipp Benz, Chaoning Zhang, Tooba Imtiaz, and In So Kweon. Double targeted universal adversarial perturbations. In *ACCV*, 2020.

[5] Julian Besag. Comments on "Representations of knowledge in complex systems" by U. Grenander and MI Miller. *Journal of the Royal Statistical Society*, 1994.

[6] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. Practical black-box attacks on deep neural networks using efficient query mechanisms. In *ECCV*, pages 154–169, 2018.

[7] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *ICLR*, 2018.

[8] Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *arXiv preprint arXiv:1712.09665*, 2017.

[9] Yulong Cao, Ningfei Wang, Chaowei Xiao, Dawei Yang, Jin Fang, Ruigang Yang, Qi Alfred Chen, Mingyan Liu, and Bo Li. Invisible for both camera and lidar: Security of multi-sensor fusion based perception in autonomous driving under physical-world attacks. In *S&P*, 2021.

[10] Yulong Cao, Chaowei Xiao, Benjamin Cyr, Yimeng Zhou, Won Park, Sara Rampazzi, Qi Alfred Chen, Kevin Fu, and Z Morley Mao. Adversarial sensor attack on lidar-based perception in autonomous driving. In *CCS*, 2019.

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

[12] Hongming Chen, Ola Engkvist, Yinhai Wang, Marcus Olivecrona, and Thomas Blaschke. The rise of deep learning in drug discovery. *Drug discovery today*, 23(6):1241–1250, 2018.

[13] Jianbo Chen, Michael I Jordan, and Martin J Wainwright. Hopskipjumpattack: A query-efficient decision-based attack. In *S&P*, 2020.

[14] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *AISec*, pages 15–26, 2017.

[15] Minhao Cheng, Simranjit Singh, Patrick H Chen, Pin-Yu Chen, Sijia Liu, and Cho-Jui Hsieh. Sign-opt: A query-efficient hard-label adversarial attack. In *ICLR*, 2019.

[16] Shuyu Cheng, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. Improving black-box adversarial attacks with a transfer-based prior. In *NeurIPS*, 2019.

[17] Edward Chou, Florian Tramer, and Giancarlo Pellegrino. Sentinet: Detecting localized universal attack against deep learning systems. *SPW*, 2020.

[18] Clarifai. Clarifai. https://www.clarifai.com/.

[19] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, 2011.

[20] Francesco Croce, Maksym Andriushchenko, Naman D Singh, Nicolas Flammarion, and Matthias Hein. Sparse-rs: a versatile framework for query-efficient sparse black-box adversarial attacks. In *AAAI*, 2022.

[21] Guneet S Dhillon, Kamyar Azizzadenesheli, Zachary C Lipton, Jeremy D Bernstein, Jean Kossaifi, Aran Khanna, and Animashree Anandkumar. Stochastic activation pruning for robust adversarial defense. In *ICLR*, 2018.

[22] Bao Gia Doan, Minhui Xue, Shiqing Ma, Ehsan Abbasnejad, and Damith C Ranasinghe. Tnt attacks! universal naturalistic adversarial patches against deep neural network systems. *IEEE TIFS*, 17:3816–3830, 2022.

[23] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *CCS*, 2017.

[24] Ryan Feng, Ashish Hooda, Neal Mangaokar, Kassem Fawaz, Somesh Jha, and Atul Prakash. Investigating stateful defenses against black-box adversarial examples. *arXiv preprint arXiv:2303.06280*, 2023.

[25] Ryan Feng, Neal Mangaokar, Jiefeng Chen, Earlence Fernandes, Somesh Jha, and Atul Prakash. Graphite: Generating automatic physical examples for machine-learning attacks on computer vision systems. In *EuroS&P*, 2022.

[26] Lianli Gao, Qilong Zhang, Jingkuan Song, Xianglong Liu, and Heng Tao Shen. Patch-wise attack for fooling deep neural network. In *ECCV*, 2020.

[27] Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. *Markov chain Monte Carlo in practice*. CRC press, 1995.

[28] Yingzhe He, Guozhu Meng, Kai Chen, Xingbo Hu, and Jinwen He. Drmi: A dataset reduction technology based on mutual information for black-box attacks. In *USENIX Security*, 2021.

[29] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In *ICML*, 2018.

[30] Andrew Ilyas, Logan Engstrom, and Aleksander Madry. Prior convictions: Black-box adversarial attacks with bandits and priors. In *ICLR*, 2019.

[31] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is bert really robust? a strong baseline for natural language attack on text classification and entailment. In *AAAI*, 2020.

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

[33] Andrej Karpathy. Tesla use per-pixel depth estimation with self-supervised learning, 2020. https://youtu.be/hx7BXih7zx8?t=1334.

[34] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[35] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[36] Huichen Li, Xiaojun Xu, Xiaolu Zhang, Shuang Yang, and Bo Li. Qeba: Query-efficient boundary-based blackbox attack. In *CVPR*, 2020.

[37] Huiying Li, Shawn Shan, Emily Wenger, Jiayun Zhang, Haitao Zheng, and Ben Y Zhao. Blacklight: Scalable defense for neural networks against query-based black-box attacks. In *USENIX Security*, 2022.

[38] Jiadong Lin, Chuanbiao Song, Kun He, Liwei Wang, and John E Hopcroft. Nesterov accelerated gradient and scale invariance for adversarial attacks. In *ICLR*, 2019.

[39] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. In *ICLR*, 2017.

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

[41] Microsoft. Azure. https://azure.microsoft.com/en-us/services/cognitive-services/.

[42] Seungyong Moon, Gaon An, and Hyun Oh Song. Parsimonious black-box adversarial attacks via efficient combinatorial optimization. In *ICML*, 2019.

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

[44] OpenAI. ChatGPT. https://openai.com/blog/chatgpt/.

[45] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *ASIACCS*, 2017.

[46] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *S&P*, 2016.

[47] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. In *BMVC*, 2015.

[48] Kexin Pei, Jonas Guan, David Williams-King, Junfeng Yang, and Suman Jana. Xda: Accurate, robust disassembly with transfer learning. In *NDSS*, 2021.

[49] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *NeurIPS*, 2019.

[50] Congzheng Song, Alexander M Rush, and Vitaly Shmatikov. Adversarial semantic collisions. In *EMNLP*, 2020.

[51] James C Spall. A one-measurement form of simultaneous perturbation stochastic approximation. *Automatica*, 33(1):109–112, 1997.

[52] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, 32:323–332, 2012.

[53] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019.

[54] Guanhong Tao, Yingqi Liu, Guangyu Shen, Qiuling Xu, Shengwei An, Zhuo Zhang, and Xiangyu Zhang. Model orthogonalization: Class distance hardening in neural networks for better security. In *S&P*, 2022.

[55] Guanhong Tao, Guangyu Shen, Yingqi Liu, Shengwei An, Qiuling Xu, Shiqing Ma, Pan Li, and Xiangyu Zhang. Better trigger inversion optimization in backdoor scanning. In *CVPR*, 2022.

[56] Chun-Chen Tu, Paishun Ting, Pin-Yu Chen, Sijia Liu, Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, and Shin-Ming Cheng. Autozoom: Autoencoder-based zeroth order optimization method for attacking black-box neural networks. In *AAAI*, 2019.

[57] Viet Vo, Ehsan M Abbasnejad, and Damith Ranasinghe. Query efficient decision based sparse attacks against black-box deep learning models. In *ICLR*, 2022.

[58] Guozhu Wang, Yiwen Cui, Jie Wang, Lihua Wu, and Guanyu Hu. A novel method for detecting advanced persistent threat attack based on belief rule base. *Applied Sciences*, 11(21):9899, 2021.

[59] Eric Wong, Leslie Rice, and J Zico Kolter. Fast is better than free: Revisiting adversarial training. In *ICLR*, 2020.

[60] Dongxian Wu, Yisen Wang, Shu-Tao Xia, James Bailey, and Xingjun Ma. Skip connections matter: On the transferability of adversarial examples generated with resnets. In *ICLR*, 2020.

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

[62] Cihang Xie, Zhishuai Zhang, Yuyin Zhou, Song Bai, Jianyu Wang, Zhou Ren, and Alan L Yuille. Improving transferability of adversarial examples with input diversity. In *CVPR*, 2019.

[63] Ziang Yan, Yiwen Guo, Jian Liang, and Changshui Zhang. Policy-driven attack: learning to query for hard-label black-box adversarial examples. In *ICLR*, 2021.

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

[65] Sheng Yu, Yu Qu, Xunchao Hu, and Heng Yin. Deepdi: Learning a relational graph convolutional network model on instructions for fast and accurate disassembly. In *USENIX Security*, 2022.

[66] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. Droid-sec: deep learning in android malware detection. In *SIGCOMM*, 2014.

Appendix

A Setup

CIFAR-10 [35] is an image classification dataset with 10 classification classes. It consists of 60,000 images and is divided into a training set with 50,000 images and a test set with 10,000 images.

SVHN [43] is a dataset containing house digital numbers extracted from Google Street View images. It has 73,257 training images and 26,032 test images. We divide the original training set into 67,257 images for training and 6,000 images for validation.

STL-10 [19] is an image recognition dataset with 10 classification classes. It consists of 5,000 training images and 8,000 test images.

GTSRB [52] is a German traffic sign recognition dataset with 43 classes. We split the dataset into a training set (35,289 images), a validation set (3,920 images), and a test set (12,630 images).

B Details of Baselines

HSJA [13] leverages gradient estimation to generate L^2/L^∞ based adversarial examples. It starts from a target-class image and gradually reduces the adversarial example's distance to the victim image. The final adversarial example is visually similar to the victim image while being misclassified to the target label. We adapt it to generate universal adversarial patches, where the perturbation is restricted within a certain area (e.g., a 7×7 square) and not bounded by L^2 or L^∞ norms. A small set of input samples instead of one single image are used for constructing the patch.

GRAPHITE [25] and SparseEvo [57] are state-of-the-art L^0 adversarial attacks. They aim to perturb a small number of pixels on the input such that it can be misclassified. GRAPHITE assumes a target-class image and uses a mask to interpolate it with the victim image, i.e., $\boldsymbol{m} \odot \boldsymbol{x}_t + (1 - \boldsymbol{m}) \odot \boldsymbol{x}_v$, where $\boldsymbol{x}_t$ is the target-class image and $\boldsymbol{x}_v$ is the victim image. The attack goal is to reduce the L^0 norm of the mask, i.e., number of perturbed pixels. It leverages binary search to find areas where the pixel values of the victim image can be retained. After there is no more area for reducing the mask, it then applies an existing gradient approximation method to further mutate adversarial pixels for improving the attack result. SparseEvo shares a similar design and leverages an evolutionary algorithm to reduce the mask size. These two attacks can be naturally extended to universal adversarial patch attack. We hence provide a set of victim samples to the attacks and use them to generate universal adversarial patches.

Bandits [30] and SPSA [51] are soft-label black-box attacks that estimate the gradient for perturbing an input sample by adding/subtracting a noise vector on/from the input and observing the output difference (e.g., the change on the target label probability). The output change served as a coefficient is multiplied with the noise vector, which is utilized as the estimated gradient for updating the adversarial perturbation. Bandits [30] also considers the gradient information from previous attack iterations and from near-by pixel positions as priors for better estimation. Sparse-RS [20] is a soft-label black-box L^0 adversarial attack. It uses random search by greedily perturbing pixels to search for a successful perturbation. Particularly, at every step, it randomly selects a set of pixels and applies random mutations to them. Sparse-RS accepts the perturbation if the cross-entropy loss is reduced and rejects it otherwise. As these three attacks are soft-label, we adapt them to the hard-label setting, where they can only observe the predicted label change as the output difference. We also provide a set of input samples so as to generate universal adversarial patches.

Table 8: Attack performance on all class pairs. All the attacks take 25k queries.

D	M	Attack	>90%	>80%	>70%	>60%	>50%	>40%	>30%	>20%	>10%
GTSRB	MobileN	HSJA	28	42	63	77	103	147	173	222	276
		SparseEvo	16	29	48	69	99	146	208	297	447
		HARDBEAT	70	127	173	231	293	377	452	551	673
	ResNt50	HSJA	74	91	103	131	157	183	214	249	289
		SparseEvo	26	40	62	89	127	174	237	321	462
		HARDBEAT	156	238	314	371	428	496	557	635	725

Baseline attacks based purely on MCMC [27] and GA [3] have been discussed in Section 1 and Section 4, and hence omitted here.

C More Results

Comparison with Baseline Attacks. Table 8 shows the results on all class pairs of GTSRB. We do not report the results for GRAPHITE because it is very slow as explained in Section 6.2. Observe that in the table, HARDBEAT has more than twice pairs with >90% ASR. In particular, it is able to generate 156 triggers on ResNet50 that have >90% ASR, significantly outperforming baselines with 74 triggers at best. More than 400 pairs on ResNet50 (293 pairs on MobileNet_v2) have a non-trivial ASR (>50%), delineating the effectiveness of HARDBEAT.

Figure 13-Figure 17 (in the supplementary material [1]) present the exact ASR for models on CIFAR-10, SVHN, and STL-10. The results on GTSRB are omitted as it has more than 1,800 pairs, which is not feasible to show here. Baseline attacks have a very limited number of successful triggers on CIFAR-10 and STL-10. HSJA is able to generate a few >80% ASR triggers. For those pairs, HARDBEAT can usually have >90% ASR. The results are slightly better for baselines on SVHN. Especially, SparseEvo has a reasonable number of non-zero triggers. But the ASRs are mostly around 30%. HARDBEAT, on the other hand, has many >50% ASRs.

Table 9: Attack performance with full-size trigger. GRAPHITE on average takes 54k queries on GTSRB, and 120k-160k queries on other datasets. Other attacks take 25k queries on GTSRB, and 50k on the remaining datasets.

D	M	Attack	Size↓	>90%	>80%	>70%	>60%	>50%	>40%	>30%	>20%	>10%
CIFAR-10	ResNt18	GRAPHITE	170	1	1	1	2	6	9	18	27	37
		SparseEvo	95	0	0	0	0	0	0	1	2	20
		HARDBEAT	49	7	11	17	19	25	33	40	46	60
	VGG11	GRAPHITE	128	4	4	5	7	8	12	15	16	27
		SparseEvo	178	0	0	0	0	0	1	3	8	23
		HARDBEAT	49	12	20	21	22	26	28	35	40	48
SVHN	ResNt18	GRAPHITE	97	0	0	1	1	4	6	15	31	57
		SparseEvo	62	0	0	0	0	2	10	30	57	76
		HARDBEAT	49	4	9	14	21	34	48	57	64	74
	ResNt34	GRAPHITE	95	0	0	0	2	4	7	20	37	57
		SparseEvo	60	0	0	0	0	3	9	24	51	79
		HARDBEAT	49	3	12	20	23	33	43	50	62	70
STL-10	GoogleN	GRAPHITE	265	0	1	2	2	3	6	7	12	15
		SparseEvo	271	0	2	2	3	4	6	9	18	25
		HARDBEAT	49	12	16	22	28	35	44	44	50	58
	DenseN	GRAPHITE	265	1	1	1	2	2	5	8	13	22
		SparseEvo	362	0	0	1	3	7	8	12	14	20
		HARDBEAT	49	3	4	9	14	21	27	31	45	60
GTSRB	MobileN	GRAPHITE	92	1	3	4	8	12	23	33	45	69
		SparseEvo	75	4	9	13	17	25	41	52	66	89
		HARDBEAT	49	8	17	22	29	38	46	56	64	77
	ResNt50	GRAPHITE	99	2	3	3	8	11	18	24	38	63
		SparseEvo	77	6	16	21	26	31	40	54	65	90
		HARDBEAT	49	20	28	34	44	49	53	65	74	92

Comparison on Full-size Trigger. Baselines GRAPHITE and SparseEvo aim to reduce the number of perturbed pixels during the attack. They may not be able to reduce the size to an L^0 of 49 (4.79% of the input). The results in Table 1 (Section 6.2.1) are obtained by reducing the trigger size to 49 after it is generated. Table 9 presents the attack results for the two baselines when their full-size triggers are used. Column 4 shows the average trigger size. Observe that the generated triggers by GRAPHITE and SparseEvo are much larger than 49. For CIFAR-10 with VGG11, the average trigger sizes by GRAPHITE and SparseEvo are respectively 2.6 times and 3.6 times larger than HARDBEAT's. GRAPHITE however has only 4 pairs with >90% ASR and SparseEvo has zero. HARDBEAT, on the other hand, obtains 12 successful pairs. Even with such a large trigger size, baselines still cannot generate many high-ASR triggers. The observations are the same on other three datasets, delineating the effectiveness of HARDBEAT.

Studying Attack Factors. Figure 18-Figure 20 (in the supplementary material [1]) show the relation between number of queries and the ASR on SVHN, STL-10, and GTSRB. The observations are similar to that on CIFAR-10. For some pairs, baselines are able to quickly find good universal adversarial patch triggers. But for most pairs, they fail to generate successful triggers even with the maximum query limit. HARDBEAT can generate triggers with reasonable ASR using a small number of queries for most pairs. For a very few pairs, it needs more queries to get a high ASR. Overall, HARDBEAT converges much faster than baselines and has much higher ASR.

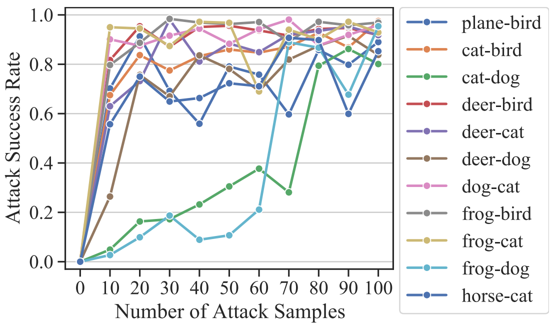


Figure 9: Attack performance with different numbers of attack samples. The experiment is conducted on CIFAR-10 with ResNet18. The trigger occupies 4.79% of the input and the attack takes 50k queries.

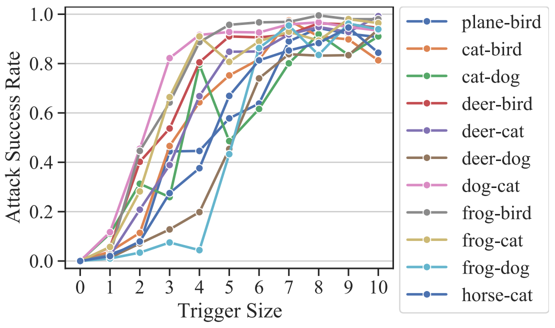


Figure 10: Attack performance with different trigger sizes. The experiment is conducted on CIFAR-10 with ResNet18 and the attack takes 50k queries.

Figure 9 and Figure 10 report effects of number of attack samples and trigger size on HARDBEAT, respectively. Observe that in Figure 9, HARDBEAT can achieve >70% ASR for most pairs by using just 20 images. The two pairs in light blue and green are harder. It requires 70-80 attack samples for HARDBEAT to achieve high ASR. Regarding the trigger size, the ASRs are all above 60% using a trigger larger than 5×5. The relation between the trigger size and the ASR is positively correlated.

D Ablation Study

Design Choices. There are three main components in the design of HARDBEAT: (1) using historical data points with high ASR, (2) using interpolated data points of previous local minima, and (3) the importance-aware gradient estimation. We remove each component from HARDBEAT and evaluate each ablated version on CIFAR-10. The results are reported in Table 10. Using historical high-ASR data points has an impact on pairs in the low ASR ranges (e.g., decreased from 33 to 25 for >40% ASR). The interpolation between previous local minima and the importance-aware gradient estimation have large impacts on the final attack results. Without them,

Table 10: Impact of different design choices

Attack	>90%	>80%	>70%	>60%	>50%	>40%	>30%	>20%	>10%
w/o History	7	10	16	17	20	25	31	42	52
w/o Interpolation	3	8	9	11	14	16	18	26	35
w/o Importance	2	6	8	9	9	10	13	23	36
HARDBEAT	7	11	17	19	25	33	40	46	60

Table 11: Impact of the decision value for non-target predictions

Value	>90%	>80%	>70%	>60%	>50%	>40%	>30%	>20%	>10%
-0.1	6	8	10	13	19	23	30	36	47
0	6	9	13	19	20	27	35	40	50
1	7	11	17	19	25	33	40	46	60

Table 12: Impact of λ

λ	>90%	>80%	>70%	>60%	>50%	>40%	>30%	>20%	>10%
0.5	5	8	15	17	19	26	33	42	55
0.6	5	8	11	16	22	24	28	44	58
0.7	7	11	17	19	25	33	40	46	60
0.8	4	9	12	14	17	22	37	45	55

the number of successful patch triggers significantly decreases (by half) in all ASR ranges, highlighting their importance.

In Algorithm 3, we use the value -1 for the decision function f to denote the added noises that are not able to cause the target prediction. Here, we study the impact of different choices of this value on the final attack results. Table 11 shows that using the value 0 or a small negative value -0.1, reduces the number of successful patch triggers in all ASR ranges. This is because using the value 0 may waste the opportunity to try other directions and a small negative value leads to limited exploration.

Hyper-parameters. There are several hyper-parameters in HARDBEAT and here we study the impact of each parameter while fixing others. λ in our Algorithm 2 balances the exploration of more random positions and the vicinity of the best trigger position. Table 12 reveals our attack performance w.r.t λ. With too small λ (such as 0.5 and 0.6), our method focuses too much on the exploration of the neighborhood of the current best trigger position (which can be a local optimum) and misses the global optimum. On the other hand, too large λ (such as 0.8) leads to too much random targetless search and thus harms the performance. Therefore, we set $\lambda = 0.7$ in our main experiments. The neighborhood range used in Algorithm 2 to mutate the current best trigger position has a similar effect as λ. As shown in Table 13, $[-1,1]$ is too small to efficiently probe the neighborhood while $[-3,3]$ is large enough to deviate from the good neighborhood. Both of them degrade the performance so we use $[-2,2]$. Another parameter is k for selecting the top-k elements in Line 7 of Algorithm 1. Its effect on the performance is listed in Table 14. A larger k allows for re-selection from more historical patterns with high performance, but may also reduce the probability of selecting the best historical pattern. This is reflected by

Table 13: Impact of the neighborhood range

Range	>90%	>80%	>70%	>60%	>50%	>40%	>30%	>20%	>10%
$[-1,1]$	6	11	15	18	24	30	36	43	58
$[-2,2]$	7	11	17	19	25	33	40	46	60
$[-3,3]$	4	10	16	18	21	29	38	46	55

Table 14: Impact of top k

k	>90%	>80%	>70%	>60%	>50%	>40%	>30%	>20%	>10%
3	5	11	14	18	22	25	34	44	60
4	7	11	17	19	25	33	40	46	60
5	6	9	14	18	25	29	41	45	58

Table 15: Impact of γ

γ	>90%	>80%	>70%	>60%	>50%	>40%	>30%	>20%	>10%
1	5	9	14	20	26	30	37	51	59
4	6	10	18	20	24	33	39	48	60
5	7	11	17	19	25	33	40	46	60
6	6	9	15	18	25	31	40	48	58

larger numbers in the columns from >50% to >10% of rows $k = 4$ and $k = 5$ compared to $k = 3$, and smaller numbers in the columns from >90% to >60% of rows $k = 3$ compared to $k = 4$. Hence we set $k = 4$ as a trade-off. γ is used to adjust the impact of directions of already misclassified samples (Line 15 of Algorithm 3). A proper γ should be set so that we can get sufficient yet not overwhelming guidance from the directions of successful samples. As shown in Table 15, increasing γ from 1 (not changing the weight) to 5 (dividing the weight by 5) benefits our attack performance by preventing the over-dominance of successful samples. However, dividing the weight by 6 over suppresses those samples, makes the guidance less informative, and thus diminishes the ASR. So we use $\gamma = 5$ in our main experiments.

E Discussion

Pre-defined Trigger Pattern. For certain universal patch attacks, it might require the attacker to pre-define the trigger pattern (e.g., as a sticker to put on the traffic signs) and then compute the trigger location and size. For this attack scenario, our Algorithm 2 can be used for finding the location. That is, we fix the trigger pattern and run lines 2-10 (except line 8) to search for the location. After obtaining a promising location, we then model the width and height of the trigger as two parameters and apply Algorithm 3 to approximate the gradients for mutating them. We leave it to future work.

Generalization to Other Classifiers. This paper focuses on studying image classifiers as described in our threat model. For classifiers in other fields such as natural language processing (NLP), HARDBEAT can leverage off-the-shelf word embeddings, and search for a set of embeddings that cause misclassification using the proposed gradient estimation. The embeddings can be mapped back to words using softmax [50]. Empirical exploration can be a future study.