# AMEBA: An Adaptive Approach to the Black-Box Evasion of Machine Learning Models

Stefano Calzavara
Università Ca' Foscari Venezia
stefano.calzavara@unive.it

Lorenzo Cazzaro
Università Ca' Foscari Venezia
864683@stud.unive.it

Claudio Lucchese
Università Ca' Foscari Venezia
claudio.lucchese@unive.it

## ABSTRACT

Machine learning models are vulnerable to evasion attacks, where the attacker starts from a correctly classified instance and perturbs it so as to induce a misclassification. In the black-box setting where the attacker only has query access to the target model, traditional attack strategies exploit a property known as transferability, i.e., the empirical observation that evasion attacks often generalize across different models. The attacker can thus rely on the following two-step attack strategy: (i) query the target model to learn how to train a surrogate model approximating it; and (ii) craft evasion attacks against the surrogate model, hoping that they "transfer" to the target model. This attack strategy is sub-optimal, because it assumes a strict separation of the two steps and under-approximates the possible actions that a real attacker might take. In this work we propose AMEBA, the first adaptive approach to the black-box evasion of machine learning models. AMEBA builds on a well-known optimization problem, known as Multi-Armed Bandit, to infer the best alternation of actions spent for surrogate model training and evasion attack crafting. We experimentally show on public datasets that AMEBA outperforms traditional two-step attack strategies.

## CCS CONCEPTS

• **Security and privacy → Database and storage security**; • **Computing methodologies → Machine learning**.

## KEYWORDS

Adversarial machine learning; evasion attacks; transferability

**ACM Reference Format:**
Stefano Calzavara, Lorenzo Cazzaro, and Claudio Lucchese. 2021. AMEBA: An Adaptive Approach to the Black-Box Evasion of Machine Learning Models. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21), June 7–11, 2021, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3433210.3453114

## 1 INTRODUCTION

Machine Learning (ML) has become phenomenally popular in recent years and found a wide range of practical applications, yet it is now acknowledged that the adoption of ML in security-oriented applications should be done with care [3]. Many papers studied the security of *supervised learning* and, in particular, its application to *classification* tasks, where ML models are trained to predict one out of a set of possible classes, e.g., spam vs. ham.

A prominent class of threats against ML comes from *evasion attacks*. In an evasion attack, the attacker starts from an instance which is classified correctly by a ML model and perturbs it so as to induce a misclassification [2]. For example, the attacker may corrupt the image of a panda through tiny pixel perturbations, which are imperceptible to humans, yet suffice to fool a ML model into predicting a gibbon with high confidence [11, 28]. Depending on the information available to the attacker, evasion attacks might be *white-box* or *black-box* [3]. White-box attacks assume the attacker to know everything about the model under attack, e.g., the learning algorithm, the training data and the model hyper-parameters. Black-box attacks, instead, only require the attacker to have query access to the model under attack (i.e., ask for predictions) and are thus particularly important from a practical perspective, since this minimal capability is inherent to the model functionality.

A traditional approach to the black-box generation of evasion attacks exploits a subtle, surprising property known as *transferability*, i.e., the empirical observation that evasion attacks often generalize across different models [20]. Hence, the attacker can adopt the following two-step attack strategy:

(1) *Surrogate Model Training*: the attacker queries the target model to extract information about its behavior and trains a surrogate model approximating the target;
(2) *Evasion Attack Crafting*: the attacker crafts successful evasion attacks against the surrogate model and feeds them to the target model, hoping that they "transfer" to it, i.e., lead to misclassification by the target.

This approach is appealing, because the attacker can train the surrogate model such that crafting successful evasion attacks against it is feasible using known algorithms. For example, evasion attack crafting algorithms like the Fast Gradient Sign Method (FGSM) [11] work for any differentiable model. Though some prominent ML models are not differentiable, e.g., decision trees, the attacker can train a differentiable surrogate model, attack it through FGSM and then evade a non-differentiable target model via transferability.

In this paper, we question the effectiveness of the two-step attack strategy proposed in previous work [20] and briefly reviewed above. In particular, we observe that there is a *tension* between the two steps of the attack strategy. On the one hand, the attacker needs to query the target model multiple times in order to disclose its behavior and train a faithful surrogate model. On the other hand, the attacker wants to query the target model with as many evasion attacks as possible to maximize the number of misclassifications. This means that when the number of queries to the target model


*ASIA CCS '21, June 7–11, 2021, Hong Kong, Hong Kong*

ACM ISBN 978-1-4503-8287-8/21/06…$15.00
https://doi.org/10.1145/3433210.3453114

is limited, e.g., because the attacker pays a price for each query or wants to behave surreptitiously, the optimal attack strategy is far from straightforward. When shall the attack strategy switch from step 1 to step 2? And, more generally, why should the attacker follow a fixed two-step strategy and not resort to a more sophisticated approach which dynamically learns how to behave?

Here we propose to move away from the two-step attack strategy of previous work and we present a new *adaptive* attack strategy, which dynamically learns whether queries to the target model should be leveraged for surrogate model training (step 1) or for evasion attack crafting (step 2), thus making the two steps of the attack intertwined. Our proposal subsumes the traditional two-step attack strategy of prior work, making black-box evasion attacks more effective and practical by automatically dealing with the delicate tension discussed above.

*Contributions.* To sum up, we contribute as follows:

(1) We propose the first adaptive approach to the black-box generation of evasion attacks against ML models. Our technique builds on a connection between the black-box evasion problem and a traditional optimization problem, known as Multi-Armed Bandit (MAB) [24]. In particular, we show how the black-box evasion problem can be reduced to MAB, hence it can be solved using existing approaches like the Thompson sampling algorithm [23]. We call the resulting attack strategy AMEBA (Adversarial Multi-armEd BAndit).
(2) We implement AMEBA and we show it at work on different datasets, considering multiple ML models and attackers. We experimentally show that, at worst, AMEBA accurately approximates the behavior of the optimal two-step attack strategy, where the attacker leverages an oracle to find the best moment to switch from step 1 to step 2. At best, instead, AMEBA leads to the creation of a large number of evasion attacks which cannot be crafted even by the optimal two-step attack strategy. This shows that future research on adversarial ML should take adaptive attack strategies into due consideration.

## 2 BACKGROUND

In this section we introduce the technical ingredients required to appreciate the rest of the paper. In particular we first clarify the notion of evasion attack, introducing appropriate terminology, and we then review the MAB problem.

### 2.1 Evasion Attacks

Let $\mathcal{X}$ be a vector space of features and $\mathcal{Y}$ a finite set of class labels, a *classifier* $h : \mathcal{X} \mapsto \mathcal{Y}$ is a function assigning a class label $y$ to each element $\vec{x} \in \mathcal{X}$ of the feature space (also called *instance*). The goal of a classifier is approximating the behaviour of an unknown *target function* $f : \mathcal{X} \mapsto \mathcal{Y}$, which assigns the correct class label to every instance. For instance, $h$ might try to approximate human understanding by discriminating between ham and spam emails based on features like: length of the email, presence of suspicious words from a blacklist, abuse of capitalization, etc. Classifiers are normally trained using *supervised learning* algorithms, which exploit a set of correctly labeled instances $\{(\vec{x}_i, f(\vec{x}_i))\}_i$, called *training set*, to identify the best-performing classifier out of a set of possible hypotheses. Technically, this is done by minimizing a *loss function* which estimates the "cost" of the prediction errors performed by the classifier on the training set.

Unfortunately, even well-performing classifiers which accurately approximate the target function might become useless when they are deployed in an adversarial setting, where an attacker actively manipulates instances to force mispredictions [3]. Formally, an attacker can be represented as a function $A : \mathcal{X} \to 2^{\mathcal{X}}$, which maps each instance into a set of possible perturbations, e.g., those instances which are located within a given distance from the original. We assume that $A$ is restricted to those perturbations which share the same true label of the original instance $\vec{x}$, i.e., $\forall \vec{z} \in A(\vec{x}) : f(\vec{z}) = f(\vec{x})$. Remarkably, even small perturbations which are negligible to human experts might suffice to perform *evasion attacks* against ML models, leading to mispredictions. For example, adding just a few so-called "good words" to a spam message might fool a spam filter into incorrectly marking the message as ham [17].

**Definition 1** (Evasion Attack)**.** Given a classifier $h$ and an instance $\vec{x}$ such that $h(\vec{x}) = f(\vec{x})$, an *evasion attack* against $\vec{x}$ is any instance $\vec{z} \in A(\vec{x})$ such that $h(\vec{z}) \neq f(\vec{x})$.

Of course, crafting evasion attacks by enumerating $A(\vec{x})$ might be infeasible in practice. For instance, if $\vec{x}$ includes 30 binary features and the attacker is able to flip 6 features at will, then $|A(\vec{x})| = \binom{30}{6} = 593775$, leading to a combinatorial explosion of the number of attacks. However, previous work identified heuristics to efficiently craft evasion attacks against differentiable models, like the FGSM algorithm [11] and its variants like Fast Gradient Value (FGV) [22]. For example, FGV operates as follows: for a given instance $\vec{x}$ with true label $y$, the perturbation added to $\vec{x}$ to generate the evasion attack $\hat{z}$ is a scaled gradient of the loss function $J$ optimized by the surrogate model $\hat{h}$. In particular, given a fixed step $\varepsilon > 0$ we have:

$$\vec{z} = \vec{x} + \varepsilon \cdot \frac{\nabla_{\vec{x}} J(\vec{x}, y)}{\|\nabla_{\vec{x}} J(\vec{x}, y)\|_2}. \tag{1}$$

This represents an $\ell_2$-norm attack where the instance $\vec{x}$ is moved along the direction of the gradient of the loss function. In this work, we use line search to find the smallest $\varepsilon$ required to evade $\hat{h}$. Indeed, such $\varepsilon$ is equivalent to the $\ell_2$-distance between $\vec{x}$ and $\vec{z}$.

Transferability allows one to leverage heuristics like FGV to attack arbitrary target models, even non-differentiable ones. The attacker first queries the target model $h$ to collect a set of predictions $\{(\vec{x}_i, h(\vec{x}_i))\}_i$ and uses it to train a differentiable surrogate model $\hat{h}$. The attacker then crafts an evasion attack $\vec{z}$ against the surrogate $\hat{h}$ using FGV and tries to evade the target model $h$ by feeding $\vec{z}$ to it. This attack strategy is shown in Figure 1, where the left part represents surrogate model training (step 1) while the right part represents evasion attack crafting (step 2).

### 2.2 Multi-Armed Bandit

The Multi-Armed Bandit (MAB) is a well-known optimization problem [24]. MAB is one of the several *reinforcement learning* approaches, where learning happens by interactions [26]. An *agent* may perform some *action* in a given *environment*, which responds to such action with a *reward* and by possibly changing its *state*. The learning task is to find the sequence of actions which leads to the

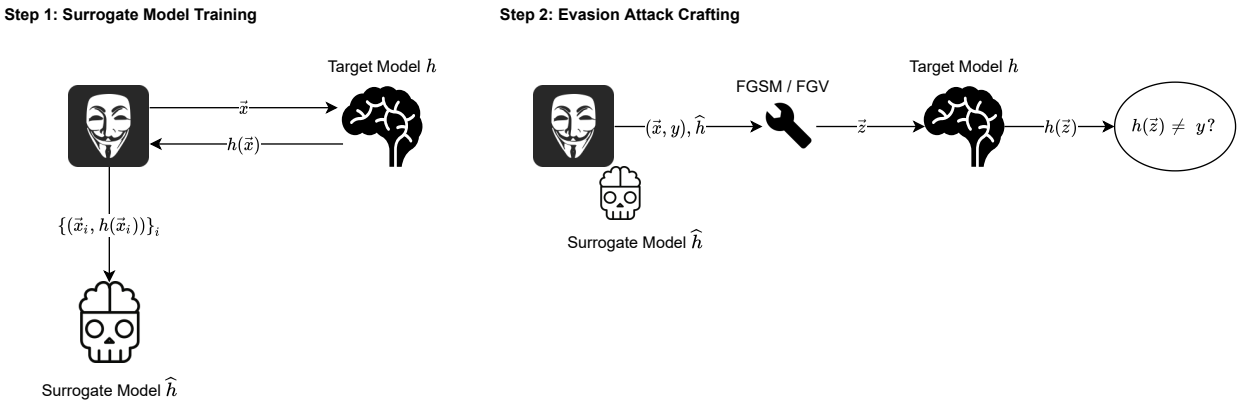



**Figure 1: Black-box evasion attack strategy through transferability**

largest reward in the long run. The agent thus struggles between exploiting the best actions observed so far and exploring the revenue of new actions. Different reinforcement learning scenarios have been investigated, depending on the nature of the rewards (stationary or non-stationary, context-dependent or context-independent) and the environment (which may be governed by a stochastic process). MAB is one such reinforcement learning approach, where the rewards are stationary and the environment is stateless.

Formally, given a set of $K \geq 2$ possible *actions* $\mathcal{A}$, also called *arms*, and $T \geq 1$ rounds, MAB requires to choose the sequence of $T$ actions from $\mathcal{A}$ which maximizes a *reward*. In its most common formulation, known as MAB with *stochastic bandits*, the problem relies on three assumptions:

(1) It is only possible to observe the reward for the selected action and nothing else. In particular, rewards for the other actions that could have been selected are unknown, even after committing to an action.
(2) For each action $a \in \mathcal{A}$, there is a distribution $D_a$ over reals, called the *reward distribution*. Every time $a$ is chosen, the reward $r$ is independently sampled from $D_a$. The reward distributions are unknown and can only be estimated when solving the problem.
(3) Per-round rewards are bounded: the standard range for rewards is the continuous interval $[0, 1]$.

The reward distributions induce a *mean reward vector* $\mu \in [0, 1]^K$, where $\mu(a) = \mathbb{E}[D_a]$ is the mean reward of the action $a$. The goal of MAB is thus finding the sequence of actions $a_1, \ldots, a_T \in \mathcal{A}$ which maximizes the *cumulative reward* $\sum_{i=1}^{T} \mu(a_i)$.

A simpler variant of MAB with stochastic bandits assumes a Bernoulli distribution of rewards. In this formulation, each action $a$ has a probability of success $\theta_a$ and produces a reward of 1, called *success*, with probability $\theta_a$ and a reward of 0, called *failure*, with probability $1 - \theta_a$. The mean reward vector $\mu$ thus coincides with the vector of the probabilities of success of each action, i.e., $\mu = (\theta_1, ..., \theta_K)$. A well-known solution to this variant of the problem is given by the *Thompson sampling* algorithm [23].

The key idea of Thompson sampling is that each action $a$ has an independent prior belief over $\theta_a$, the *estimate* $\hat{\theta}_a$, and at each round the action with the highest estimate is chosen. The estimate

**Algorithm 1** Thompson sampling

```
for a in A do
    (S_a, F_a) ← (1, 1)                          ▷ Initialization
for t = 1, ..., T do
    for a in A do
        Sample θ̂_a ~ Beta(S_a, F_a)
    a_t ← arg max_{a∈A} θ̂_a
    r_t ← PERFORM(a_t)                           ▷ Get reward of a_t
    (S_{a_t}, F_{a_t}) ← (S_{a_t} + r_t, F_{a_t} + (1 − r_t))
```

$\hat{\theta}_a$ is sampled from a Beta distribution with parameters $S_a$ and $F_a$. These two parameters are often called *pseudo-counts*, since $S_a$ and $F_a$ increase by 1 with each observed success or failure, respectively. Algorithm 1 presents the pseudocode of Thompson sampling, where the PERFORM function takes the chosen action and returns the corresponding reward (0 or 1).

## 3 ADAPTIVE BLACK-BOX ATTACKS

We first discuss our threat model, describing the attacker's goals and capabilities, and we then present the details of our adaptive attack strategy. In particular, we show how the black-box evasion problem can be reduced to the MAB problem.

### 3.1 Threat Model

We consider an attacker whose goal is to craft successful evasion attacks against a target model $h$, approximating an unknown target function $f$. We do not make any assumption on $h$, but we assume the attacker has black-box access to it. In particular, the attacker can perform a limited number of queries to $h$ by asking for class predictions on arbitrarily chosen instances. Queries can be used either to train a (differentiable) surrogate model $\hat{h}$ or to attempt evasion attacks against the target model $h$, by feeding it with evasion attacks working against $\hat{h}$ via transferability. Queries are limited because the attacker might not have unconstrained access to the target model for several reasons. For example, query access to the target might require a payment, like in the case of the Google Cloud

Vision API[1] and Amazon Machine Learning,[2] or the target might be equipped with an intrusion detection system which limits the number of queries during the attack opportunity window.

More specifically, we assume the attacker has access to the following datasets:

- $\mathcal{D}_{trn}$: a set of instances $\{(\vec{x}_i, h(\vec{x}_i))\}_i$ labeled with the class predictions of the target $h$, used for surrogate model training. For example, $\mathcal{D}_{trn}$ might be a collection of known spam and ham messages available to the attacker.
- $\mathcal{D}_{atk}$: a set of instances $\{(\vec{x}_i, f(\vec{x}_i))\}_i$ labeled with their true labels, used for evasion attack crafting. For example, $\mathcal{D}_{atk}$ might be a set of spam messages that the attacker wants to evade a spam filter.
- $\mathcal{D}_{un}$: a set of unlabeled instances $\{\vec{x}_i\}_i$, used to collect additional class predictions from the target $h$. For example, $\mathcal{D}_{un}$ might include messages that the attacker has written himself, whose class predictions would be unknown.

We do not make any assumption on the instances in these three datasets: they can be equal, overlapping or disjoint. We assume the datasets are organized as queues, i.e., they have standard push and pop operations.

We are finally ready to discuss how the attacker operates. We assume the attacker first uses $\mathcal{D}_{trn}$ to train the initial surrogate $\hat{h}$ via supervised learning. Once this is done, the attacker can choose between two possible actions:

(1) *Train*: the attacker pops an instance $\vec{x} \in \mathcal{D}_{un}$, queries $h$ to learn the prediction $y = h(\vec{x})$ and extends $\mathcal{D}_{trn}$ with $(\vec{x}, y)$. The attacker then updates the surrogate $\hat{h}$ by retraining it over the extended $\mathcal{D}_{trn}$.
(2) *Attack*: the attacker pops an instance $(\vec{x}, y) \in \mathcal{D}_{atk}$. If $\hat{h}(\vec{x}) = y$, the attacker uses $\vec{x}$ to craft an evasion attack $\vec{z}$ against $\hat{h}$ by using an appropriate attack strategy, e.g., FGV. If a successful evasion attack is found, i.e., if $\hat{h}(\vec{z}) \neq y$, the attacker submits $\vec{z}$ to the target model $h$ and verifies whether $h(\vec{z}) \neq y$.

The attacker stops when the maximum number of queries to the target model $h$ has been performed. Note that the *Attack* action might fail without querying $h$ in two cases: (i) when $(\vec{x}, y)$ is misclassified by $\hat{h}$, or (ii) when it is impossible to turn $\vec{x}$ into a successful evasion attack against $\hat{h}$. In both cases, we assume that $(\vec{x}, y)$ is temporarily discarded and pushed back into $\mathcal{D}_{atk}$ for later use. The reason for this choice is that, since the *Train* action aims at improving the quality of the surrogate model $\hat{h}$, it might get easier to evade $\hat{h}$ over time.

## 3.2 Adaptive Attack Strategy via MAB

Previous work relies on a basic two-step attack strategy where a sequence of *Train* actions is performed first, in order to build a representative surrogate model, and then a sequence of *Attack* actions is taken to craft evasion attacks [20]. We rather propose a dynamic attack strategy where the attacker may intertwine *Train* and *Attack* actions at will, so as to minimize the number of instances used to build the surrogate $\hat{h}$ and to maximize the number of generated evasion attacks against the target $h$. Our adaptive attack strategy operates by reduction to the Bernoulli MAB problem.

We let the available actions be $\mathcal{A} = \{Train, Attack\}$ and we let $T$ represent the number of available queries to the target model $h$. Each action consumes one query to the target model and actions are interleaved up to the exhaustion of the query budget $T$. By adopting a Bernoulli MAB model, we assume the two actions provide a reward in terms of a binary outcome, i.e., success vs. failure. The definition of the reward of the two actions is crucial to make sure that the maximization of the cumulative reward of MAB matches the goal of our black-box attack strategy, i.e., generating as many successful evasion attacks as possible.

The key observation to make here is that the success rate of the *Attack* actions depends on the quality of the surrogate model $\hat{h}$. A high quality surrogate model should provide similar predictions to those of the target model $h$, so as to increase the transferability of the attacks. The quality of $\hat{h}$, in turn, improves thanks to the *Train* actions, which enrich the surrogate model's training set $\mathcal{D}_{trn}$. The attacker thus aims at finding a sequence of *Train* and *Attack* actions that maximizes both the number of evasion attacks against the target model and the quality of the surrogate model. Hence, we want to reward *Attack* actions leading to successful evasion attacks and *Train* actions leading to improved surrogate quality.

For the *Attack* action, it is straightforward to define the notion of success: we stipulate success when the crafted evasion attack $\vec{z}$ against the instance $\vec{x}$ with label $y$ transfers from the surrogate to the target, i.e., when $\hat{h}(\vec{z}) \neq y$ and $h(\vec{z}) \neq y$. As to the *Train* action, we want to define success when the surrogate model improves its similarity with the target model. Since similarity cannot be estimated on the basis of the single query encompassed by the *Train* action, we pragmatically choose to compare the accuracy score of $\hat{h}$ over $\mathcal{D}_{trn}$ before and after the *Train* action: if the accuracy score increases according to 10-fold cross-validation, we report a success. This is effective because $\mathcal{D}_{trn}$ is populated with class predictions from the target model, hence the accuracy score of the surrogate on $\mathcal{D}_{trn}$ is a reliable proxy of its similarity to the target.

The reason why the proposed reward scheme is effective is that, when we have a low success rate of the *Attack* actions, the best choice is to increase the number of the *Train* actions to improve the quality of the surrogate and its transferability, which would lead to increasing the success rate of the *Attack* actions. Iteration after iteration, the *Train* actions do not provide any benefit anymore, as the surrogate's similarity to the target reaches a plateau, and the *Attack* actions normally become the most valuable choice.

There are still a couple of subtle points to note though. First, the proposed approach ignored the MAB assumption that the reward distributions of the actions $D_a$ are permanent and independent, since the surrogate model $\hat{h}$ is updated after every *Train* action and its quality affects the probability of success of both actions. Yet, our experimental evaluation (Section 4) shows the effectiveness of the Thompson sampling algorithm for Bernoulli MAB, which is due to the fact that the reward distributions do not negatively interfere, but rather boost one another as discussed.

Moreover, while the *Train* action always performs one query to the target model $h$, the described *Attack* action may not perform any query to $h$ when the evasion attack against the surrogate $\hat{h}$ fails, i.e.,

[1] https://cloud.google.com/vision/pricing?hl=en
[2] https://aws.amazon.com/getting-started/projects/build-machine-learning-model/services-costs/

there is a potential disconnect between the number of actions and the number of actual queries to the target model $h$. To close this gap, we assume that this type of failure does not discourage the attacker, who just moves to the next instance of $\mathcal{D}_{atk}$ until a successful evasion attack against $\hat{h}$ is found. This has the side-effect of giving some instances a second chance in a future round, featuring an updated surrogate model, which might be more similar to the target and present higher transferability. In the extreme case where the attacker cannot craft any successful evasion attack against $\hat{h}$ for any of the instances in $\mathcal{D}_{atk}$, we assume that the *Train* action is taken instead. This is a reasonable choice, because it is the only option available to the attacker given the current surrogate, and at the same time it ensures the invariant that each action consumes exactly one query to the target model $h$.

### 3.3 AMEBA

Having clarified these points, the reduction from the black-box evasion problem into Bernoulli MAB is done, hence one can leverage the Thompson sampling algorithm to implement an adaptive attack strategy. The details of the resulting attack strategy, called AMEBA, are formalized in Algorithm 2. Note that, although the pseudocode relies on FGV for evasion attack crafting, any other algorithm for the same task could be used.

The algorithm starts by initializing the pseudo-counts of the two actions (lines 1–2). Then, at each of the $T$ rounds it selects the action with the highest estimate of success. Lines 7–21 implement the *Attack* action as previously discussed. First, the attacker iterates through $\mathcal{D}_{atk}$ in search of an instance $(\vec{x}, y)$ for which it is possible to craft a successful evasion attack against the surrogate model $\hat{h}$. If such an attack $\vec{z}$ is found (line 17), it is submitted to the target model gaining a reward $r_t = \mathbb{1}(h(\vec{z}) \neq y)$, where $\mathbb{1}(p)$ equals 1 if the predicate $p$ is true and 0 otherwise. In other words, $r_t = 1$ if $h$ misclassifies the perturbed instance $\vec{z}$ and $r_t = 0$ otherwise.

If no evading instance is found, or if the estimated probability of attack success $\hat{\theta}_{Attack}$ is not greater than the estimated probability of train success $\hat{\theta}_{Train}$, then the *Train* action is performed (lines 22–30). In this case, a new instance $\vec{x}$ retrieved from $\mathcal{D}_{un}$ is submitted to the target model $h$ to get the corresponding prediction and is then used to enrich the training set $\mathcal{D}_{trn}$. A new surrogate model is finally trained and used in the subsequent iterations. We set the reward to 1 if we observe an increase of the cross-validation score of the surrogate, to 0 otherwise.

**Algorithm 2** The AMEBA attack strategy

1: **for** $a$ **in** $\{Train, Attack\}$ **do**
2: $(S_a, F_a) \leftarrow (1, 1)$ ▷ Initialization
3: **for** $t = 1, ..., T$ **do**
4: Sample $\hat{\theta}_{Train} \sim Beta(S_{Train}, F_{Train})$
5: Sample $\hat{\theta}_{Attack} \sim Beta(S_{Attack}, F_{Attack})$
6: $evading \leftarrow \perp$ ▷ Evasion attack yet not found
7: **if** $\hat{\theta}_{Attack} > \hat{\theta}_{Train}$ **then** ▷ *Attack* action
8: $available \leftarrow |\mathcal{D}_{atk}|$
9: **while** $evading = \perp \wedge available > 0$ **do**
10: $available \leftarrow available - 1$
11: $(\vec{x}, y) \leftarrow \textsc{Pop}(\mathcal{D}_{atk})$
12: $\vec{z} \leftarrow \text{FGV}((\vec{x}, y), \hat{h})$ ▷ Craft evasion attack
13: **if** $\hat{h}(\vec{x}) = y \wedge \hat{h}(\vec{z}) \neq y$ **then** ▷ Confirm evasion
14: $evading \leftarrow (\vec{z}, y)$
15: **else**
16: $\mathcal{D}_{atk} \leftarrow \textsc{Push}(\mathcal{D}_{atk}, (\vec{x}, y))$
17: **if** $evading \neq \perp$ **then** ▷ Found evasion attack on $\hat{h}$
18: $a_t \leftarrow Attack$
19: $(\vec{z}, y) \leftarrow evading$
20: $r_t \leftarrow \mathbb{1}(h(\vec{z}) \neq y)$ ▷ Check transferability on $h$
21: $(S_{a_t}, F_{a_t}) \leftarrow (S_{a_t} + r_t, F_{a_t} + (1 - r_t))$
22: **else** ▷ *Train* action
23: $a_t \leftarrow Train$
24: $old\_cv\_score \leftarrow \textsc{CrossValScore}(\hat{h}, \mathcal{D}_{trn})$
25: $\vec{x} \leftarrow \textsc{Pop}(\mathcal{D}_{un})$
26: $y \leftarrow h(\vec{x})$
27: $\mathcal{D}_{trn} \leftarrow \textsc{Push}(\mathcal{D}_{trn}, (\vec{x}, y))$
28: $\hat{h} \leftarrow \textsc{Train}(\mathcal{D}_{trn})$
29: $r_t \leftarrow \mathbb{1}(\textsc{CrossValScore}(\hat{h}, \mathcal{D}_{trn}) > old\_cv\_score)$
30: $(S_{a_t}, F_{a_t}) \leftarrow (S_{a_t} + r_t, F_{a_t} + (1 - r_t))$

## 4 EXPERIMENTAL EVALUATION

In this section we report on an experimental evaluation of AMEBA. We first introduce the experimental setup, then we explain the methodology and finally we present the key results of our analysis, including a performance evaluation in terms of the running times of our adaptive attack strategy.

### 4.1 Experimental Setup

We perform an experimental evaluation of AMEBA on three public datasets: Spambase,[3] Wine Quality[4] and CodRNA.[5] All datasets are associated with a binary classification task and their main statistics are shown in Table 1. For each dataset, we consider multiple scenarios to cover a wide range of possible application settings and threats. In particular, we train a Linear SVM model as surrogate and we exploit it to craft black-box evasion attacks against three different target models available in the state-of-the-art scikit-learn library [21]:

(1) a decision tree ensemble trained using the Random Forest algorithm. This is a non-differentiable model based on multiple, independently trained decision trees;
(2) a decision tree ensemble trained using the AdaBoost algorithm. This is also a non-differentiable model, but it differs from Random Forest because each tree $t_i$ is trained so as to improve upon the shortcomings of the tree ensemble $t_1, \ldots, t_{i-1}$ including the previously trained trees;
(3) a logistic regression classifier. This is a simple differentiable model, which is closer to Linear SVM and thus complementary to decision tree ensembles.

The choice of using Linear SVM as surrogate model is motivated by previous work, which showed that such type of model allows for the construction of effective black-box attacks with strong transferability [8]. We do not consider other surrogates, since Linear SVM

[3] https://archive.ics.uci.edu/ml/datasets/Spambase
[4] https://archive.ics.uci.edu/ml/datasets/wine quality
[5] https://www.openml.org/d/351

**Table 1: Dataset statistics**

| | Spambase | Wine | CodRNA |
|---|---|---|---|
| **n. of instances** | 4601 | 6495 | 488565 |
| **n. of features** | 54 | 12 | 8 |
| **class distribution** | 39 ÷ 61 | 25 ÷ 75 | 67 ÷ 33 |

**Table 2: Accuracy scores of the target models**

| | Spambase | Wine | CodRNA |
|---|---|---|---|
| **Random Forest** | 0.96 | 0.99 | 0.97 |
| **AdaBoost** | 0.97 | 0.99 | 0.97 |
| **Logistic Regression** | 0.93 | 0.99 | 0.95 |

is already quite effective in our experiments and the attacker is free to use it to carry out a real attack. To prove that our approach generalizes to deep learning, we present additional experiments on image classification in Appendix A.

As to evasion attack crafting, we use the attack against (multi-class) SVM proposed in [19]. This attack is an adaptation of FGV to SVM, where the loss function $J$ of Equation 1 is replaced by the scoring function of the Linear SVM, whose gradient is the weight vector $\vec{w}$. For an instance $\vec{x}$ with binary label $y \in \{-1, +1\}$, the candidate evasion attack $\vec{z}$ is thus computed as:

$$\vec{z} = \vec{x} - \varepsilon \cdot y \cdot \frac{\vec{w}}{\|\vec{w}\|_2}. \tag{2}$$

Note that the sign of the perturbation depends on the class $y$ to evade. For simplicity, we just focus on *untargeted* evasion attacks, i.e., we assume that the attacker is willing to target any instance of any class. Finally, we assume the attacker can perform at most $T = 1000$ queries to the target model and we simulate two attack scenarios of different magnitude ($\varepsilon$ in Equation 2). To mitigate the bias coming from the choice of a specific number of queries, Appendix B presents additional results for $T = 2000$.

All the models are trained after normalizing features in the interval [0,1] and using hyper-parameter tuning, a standard ML practice used to identify the best-performing models. In particular, hyper-parameter tuning is conducted by using grid search with the cross validation score on the training set as performance measure, testing the following hyper-parameters:

- Random Forest: number of trees ($\{64, 128, 256, 512, 1024\}$) and node splitting criterion (gini impurity and max entropy);
- AdaBoost: number of trees ($\{64, 128, 256, 512\}$), node splitting criterion (gini impurity and max entropy) and number of leaves ($\{8, 16, 32, 64\}$);
- logistic regression: regularization factor $C$ (from $10^{-4}$ to $10^4$).

We use $C = 0.50$ as regularization factor of the Linear SVM used as surrogate model. This choice is motivated by the fact that highly regularized models typically provide better transferability [8].

## 4.2 Methodology

To assess the effectiveness of AMEBA, we proceed as follows. First, each dataset $\mathcal{D}$ is partitioned into two sets $\mathcal{D}_{tgt}$ and $\mathcal{D}_{sur}$ using stratified random sampling, i.e., they are randomly partitioned respecting the original class distribution. The $\mathcal{D}_{tgt}$ component is used to train the target models: we reserve 1600 instances of Spambase, 3000 instances of Wine and 100000 instances of CodRNA for this task. This amount of instances suffices to achieve high values of accuracy for all the target models, as reported in Table 2.

The $\mathcal{D}_{sur}$ component, instead, is used to train the surrogate model and craft evasion attacks. In particular, we partition $\mathcal{D}_{sur}$ into the sets $\mathcal{D}_{trn}$, $\mathcal{D}_{atk}$ and $\mathcal{D}_{un}$ expected by AMEBA, again using stratified random sampling. More precisely, these sets are constructed as follows:

- $\mathcal{D}_{trn}$ includes 100 instances used to train the initial surrogate model. Since the labels of $\mathcal{D}_{trn}$ are provided by the target model, we assume the attacker has spent 100 queries for retrieving them. This size suffices to train a minimally meaningful surrogate model to start AMEBA.
- $\mathcal{D}_{atk}$ includes 900 instances correctly classified by the target models, which are used to craft evasion attacks. The size of $\mathcal{D}_{atk}$ is motivated by the fact that the attacker cannot perform more than 1000 queries to the target model and we already provided him with 100 labeled instances of $\mathcal{D}_{trn}$ to build the initial surrogate model.
- $\mathcal{D}_{un}$ includes the remaining (unlabeled) instances, which are used to learn class predictions from the target model and improve the quality of the surrogate. Specifically, we let $\mathcal{D}_{un} = \{\vec{x} \mid \exists y : (\vec{x}, y) \in \mathcal{D}_{sur} \setminus (\mathcal{D}_{trn} \cup \mathcal{D}_{atk})\}$.

Our main goal is comparing the performance of AMEBA against a traditional two-step attack strategy, where there is a clear separation between surrogate model training (step 1) and evasion attack crafting (step 2). However, performing such a comparison is far from straightforward. The key challenge to deal with is that prior work assumed the adoption of a two-step attack strategy, but did not investigate when the transition from step 1 to step 2 should occur. Rather, prior work assumed the usage of a reasonably accurate surrogate model in step 2, empirically built after a fixed number of training rounds [20]. How to fix the number of training rounds is left unspecified, yet this is a delicate point, given that the number of available queries to the target model is limited.

To address this shortcoming, we compare AMEBA against multiple baselines, so as to cover multiple possible choices for the number of queries spent for surrogate model training. Specifically, we operate as follows: for each $i \in \{0, 50, 100, \ldots, 700\}$, we use stratified random sampling to collect $i$ instances from $\mathcal{D}_{atk}$ and we add them to $\mathcal{D}_{trn}$, generating datasets $\mathcal{D}^i_{trn}$ (of size $i + 100$). This allows us to simulate 15 possible attack scenarios where each $\mathcal{D}^i_{trn}$ is used for surrogate model training and the remaining instances in $\mathcal{D}_{atk}$ are used for evasion attack crafting. We never use more than 800 instances for surrogate model training, since otherwise less than 200 instances would be available for evasion attack crafting, which would significantly lower the attack opportunities. Ideally, a powerful clairvoyant attacker would choose in advance the best training size for the surrogate model, and therefore perform as the best of the considered baselines. We would like to show that AMEBA approximates such clairvoyant attacker, or even improves over it, thus proving that adaptive attack strategies can subsume static ones.

We evaluate the performance of the attack strategies in terms of the following two measures:

(1) absolute number of successful evasion attacks against the target model, i.e., number of mispredictions forced on correctly classified instances;
(2) transferability, i.e., the percentage of successful evasion attacks out of all the attempted evasion attacks against the target model.

The first measure is the most important for the attacker considered in our threat model, who wants to craft as many successful evasion attacks as possible, yet we also keep an eye on the second measure, given that it has been extensively studied in the literature [8, 19]. Note that, since AMEBA relies on probabilistic sampling, the two measures are computed as the average of the results obtained in 10 different runs.

## 4.3 Experimental Results

We start by commenting the results on the Spambase dataset, which are shown in Figure 2. The outer bars represent the number of evasion attacks successfully crafted against the surrogate model, while the inner bars show those which turned out to be effective on the target model as well; the lines, instead, show the value of transferability. On the Spambase dataset, the results for Random Forest and AdaBoost are very similar, and AMEBA outperforms the best-performing baseline in terms of successful evasion attacks. For example, in the case of Random Forest with perturbation $\varepsilon = 0.10$, AMEBA generates 395 successful attacks, while the best-performing baseline only produces 245 successful attacks (+61%). We also observe that, when the amount of perturbation $\varepsilon$ increases from 0.10 to 0.15, crafting successful evasion attacks becomes easier and the difference between AMEBA and the best-performing baseline decreases. However, AMEBA still significantly improves over the baseline in terms of absolute number of successful evasion attacks: 560 vs. 462 in the case of Random Forest (+21%).

An important observation supported by our experiments is that, while the transferability always tends to grow with the number of queries spent for surrogate model training, it is hard to identify the optimal amount of queries which maximizes the number of successful evasion attacks. Indeed, though transferability improves with larger training sizes, the number of queries available for evasion attack crafting correspondingly becomes smaller, thus reducing the attack opportunities and enforcing a delicate trade-off on how queries to the target model should be spent. AMEBA automatically deals with this problem, while keeping a very high value of transferability, i.e., from 78% to 84% on Random Forest. Though the transferability of the best-performing baseline ranges from 89% to 94% on Random Forest, this reduction in transferability is largely compensated by the increased amount of successful attacks.

The results for logistic regression confirm the trends which we observed for the other target models, but are interesting because they also show a counter-intuitive phenomenon. Specifically, we observe that the transferability of evasion attacks against logistic regression is significantly lower than for the other two target models, which was quite unexpected given that logistic regression is a differentiable model bearing stronger similarities to the surrogate model (Linear SVM). This is just one of the many surprises that transferability might hide [8]. For the case $\varepsilon = 0.10$, AMEBA generates 326 successful evasion attacks, as opposed to the 186 attacks of the best-performing baseline on logistic regression (+75%). Observe also that the transferability of the two attack strategies is extremely close: 66% vs. 70% when considering the baseline which produces the highest number of successful evasion attacks. Similar considerations apply to the case $\varepsilon = 0.15$.

We now comment on the results for the Wine Quality dataset, shown in Figure 3. The first observation here is that the results on the three target models are very close and, notably, crafting successful evasion attacks is generally harder than for the Spambase dataset, in terms of both absolute numbers and transferability. Nevertheless, AMEBA still works better than the best-performing baseline in all cases. For example, in the case of Random Forest with perturbation $\varepsilon = 0.20$, the best-performing baseline produces 230 successful evasion attacks, while AMEBA generates 241 successful attacks (+5%). This result is already positive, since the attacker does not know what the optimal baseline is. When the amount of perturbation $\varepsilon$ increases from 0.20 to 0.25, evading the target model becomes easier and the gap between AMEBA and the best-performing baseline becomes more apparent. In particular, AMEBA can generate 454 successful evasion attacks, as opposed to the 406 attacks of the best-performing baseline (+12%). The transferability of the evasion attacks generated by AMEBA is very close to that of the best-performing baseline in the case of decision tree ensembles, and particularly for Random Forest, while the gap is bigger for logistic legression. However, transferability is still reasonably high in general, ranging from 65% at worst to 71% at best.

Finally, we report in Figure 4 the results for the CodRNA dataset. The results are again very positive and even show improvements over the other datasets: AMEBA does not just generate a higher number of successful evasion attacks than the best-performing baseline, but it also improves over most baselines in terms of transferability. For example, in the case of Random Forest with perturbation $\varepsilon = 0.10$, the best-performing baseline produces 290 successful evasion attacks, while AMEBA can craft 411 successful attacks (+42%). The evasion attacks crafted by AMEBA have a transferability of 64%, which is very close to the transferability of the baseline producing the largest number of successful attacks (65%). When moving to $\varepsilon = 0.15$, the best-performing baseline identifies 475 successful evasion attacks, while AMEBA produces 560 successful attacks (+18%). The evasion attacks crafted by AMEBA have a transferability of 75%, a value which improves over most baselines and is quite close to the transferability of the baseline producing the largest number of successful attacks (78%).

## 4.4 Why AMEBA Works?

To better understand why AMEBA is effective in crafting evasion attacks, we also carry out some additional analyses. In the first one, we focus on the alternation between the *Train* and the *Attack* actions chosen by AMEBA. Figure 5 presents the results of our analysis on the Spambase dataset. The figure uses red and blue lines to show the trend of the average reward for the *Attack* and the *Train* actions respectively; the background of the plots shows instead which of the two actions was taken at each round. The figure shows some interesting trends and, to appreciate them, we

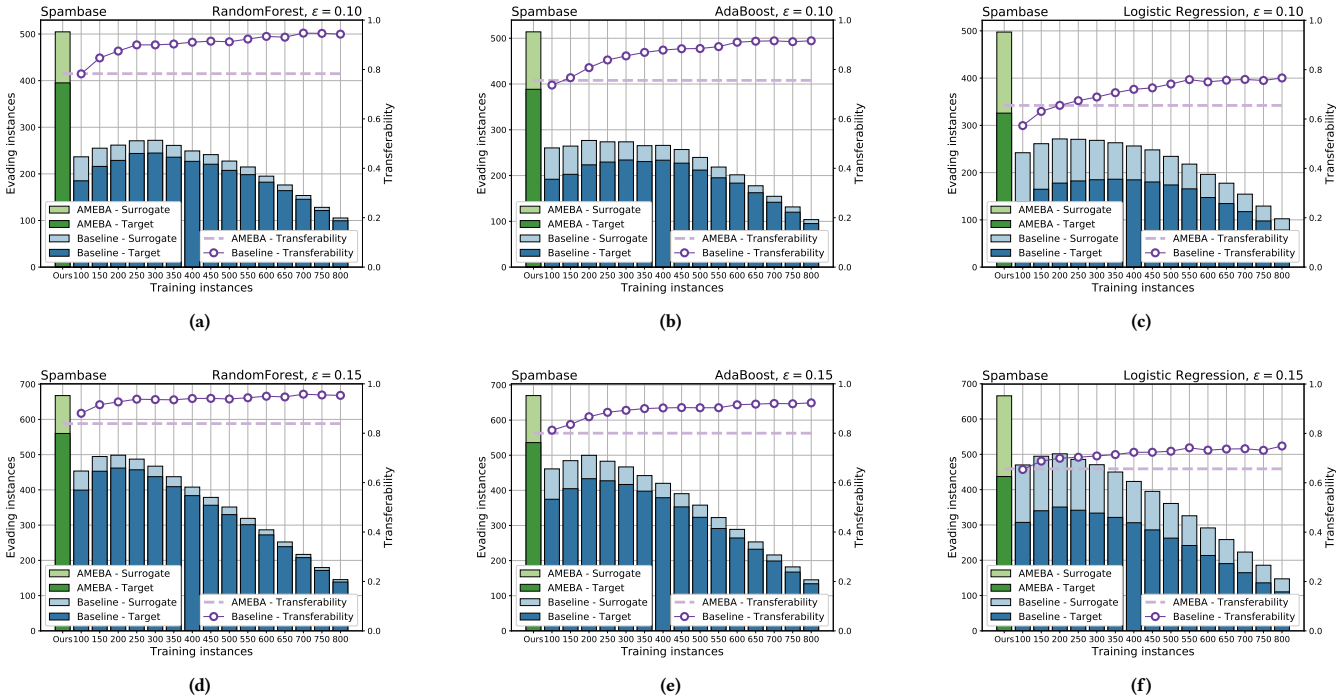



**Figure 2: Experimental results on the Spambase dataset**

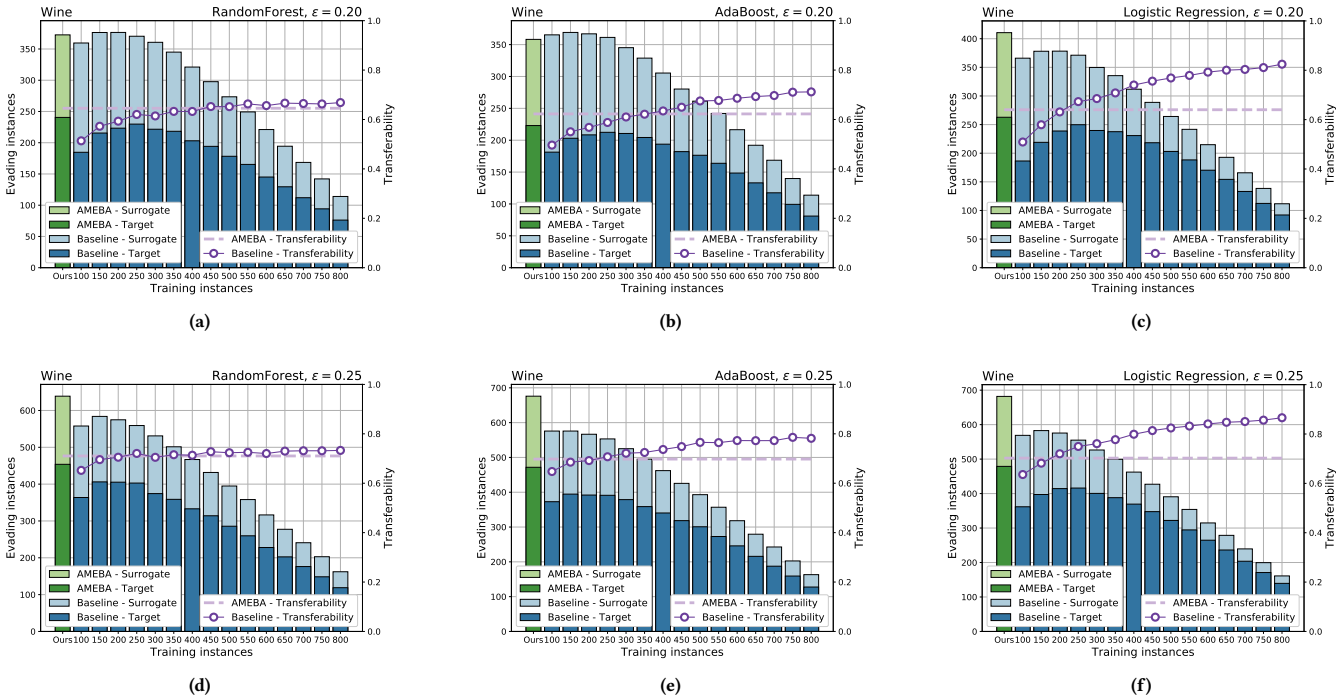



**Figure 3: Experimental results on the Wine Quality dataset**

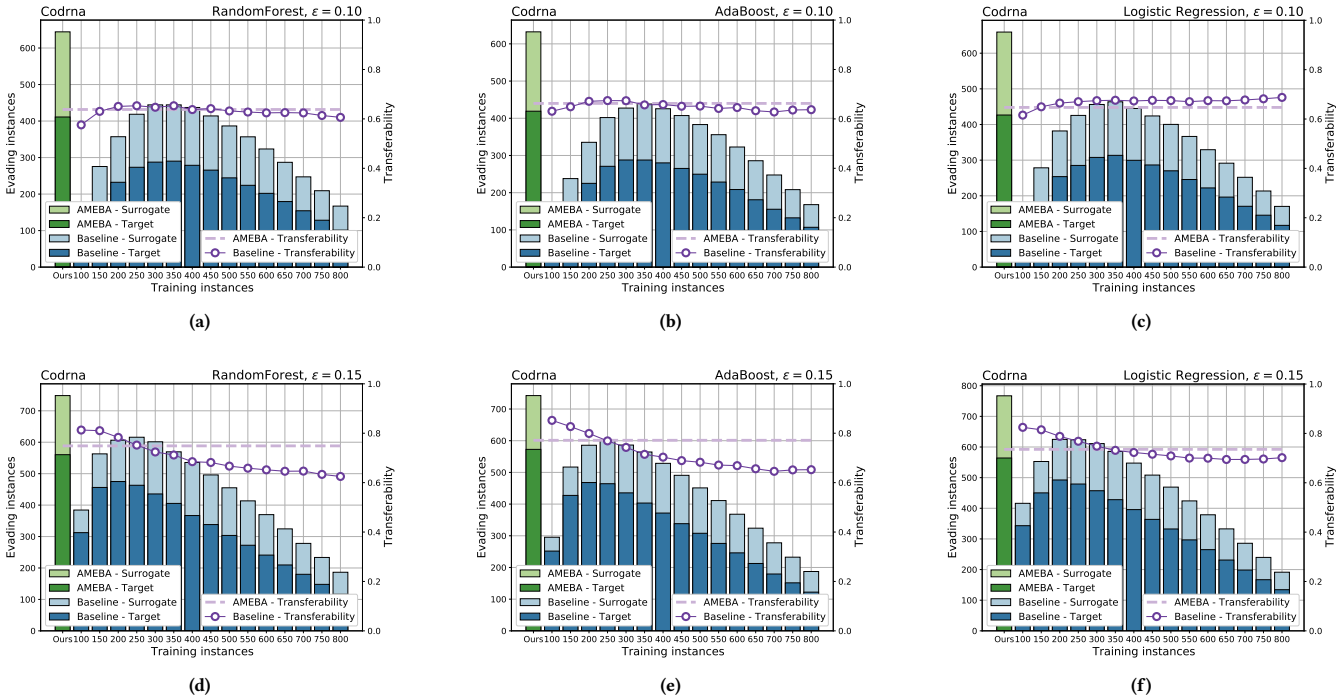



**Figure 4: Experimental results on the CodRNA dataset**

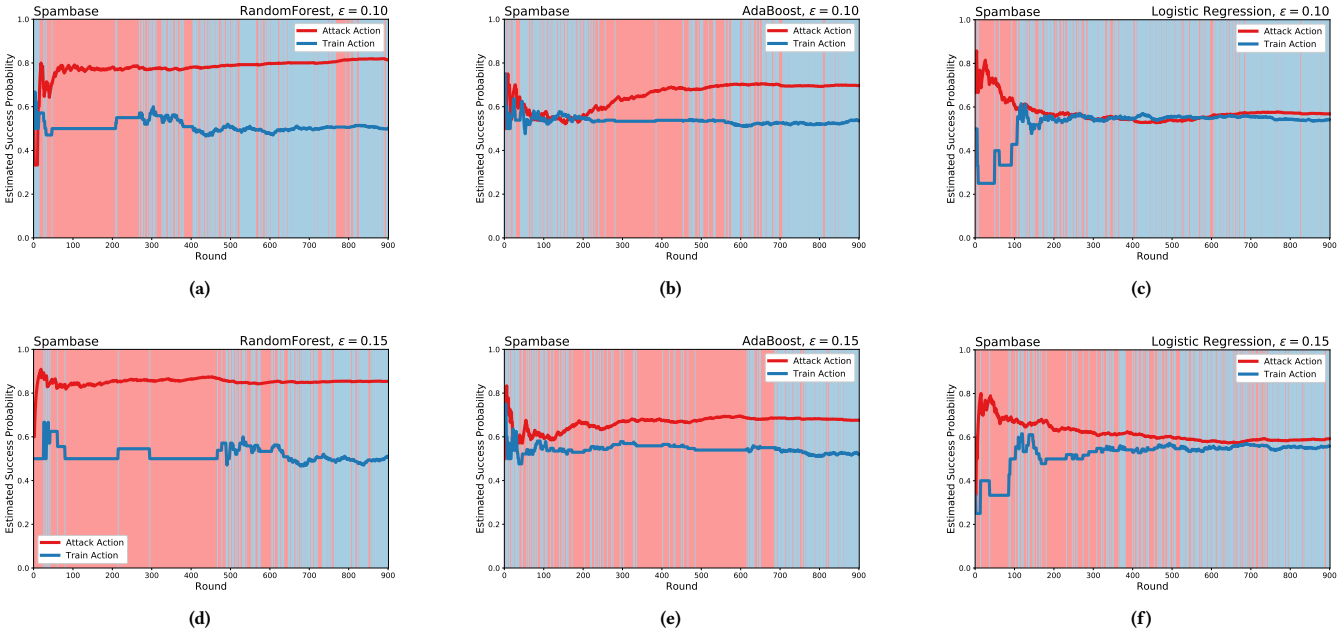



**Figure 5: Actions and mean rewards on the Spambase dataset**

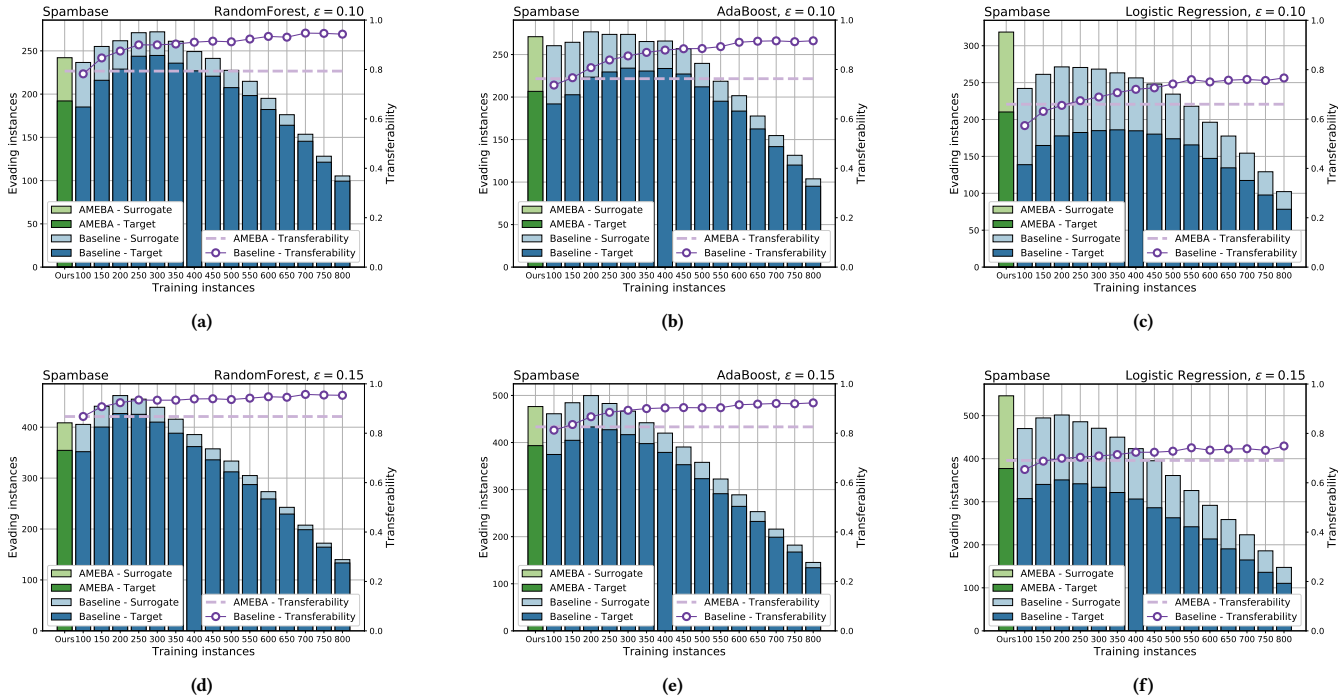



**Figure 6: Experimental results on the Spambase dataset (without queue for instance reuse)**

recall that AMEBA works better against tree ensembles than against logistic regression on the Spambase dataset. When the perturbation is smaller ($\varepsilon = 0.10$) and the target model is quite vulnerable to evasion (Random Forest and AdaBoost), AMEBA shows the expected trend: it first performs a number of *Train* actions to improve the quality of the surrogate model, then it primarily moves to the *Attack* action and tries to capitalize over it. The use of the *Train* action thus becomes sporadic and only returns popular at the end of the run, where the only instances which are left in $\mathcal{D}_{atk}$ do not allow for the creation of evasion attacks working against the surrogate model. In the case of logistic regression, instead, AMEBA starts with some *Attack* actions and then resorts to training, given the encountered challenges of evasion; only after the mean reward of the *Attack* action exceeds that of the *Train* action, AMEBA switches back to evasion attack crafting. Note that, when the perturbation becomes larger ($\varepsilon = 0.15$), evasion attack crafting becomes easier on all target models and AMEBA normally privileges the *Attack* action over the *Train* action.

To conclude our analysis of the inner workings of AMEBA, we carry out a last experiment to understand the impact of organizing $\mathcal{D}_{atk}$ as a queue for evasion attack crafting. Recall that, when AMEBA cannot craft an evasion attack for a given instance, that instance is pushed back into $\mathcal{D}_{atk}$ for later use rather than discarded. The intuition here is that, since the surrogate model is refined over time, instances leading to a failure at a given round might lead to successful evasion attacks in a later round. Figure 6 shows the results we would get on the Spambase dataset if we did not organize $\mathcal{D}_{atk}$ as a queue and rather discarded instances which cannot evade the surrogate when they are first chosen for evasion attack crafting. Observe that AMEBA would turn out to be less effective than the best-performing baseline in the case of tree ensembles, where it used to outperform it (cf. Figure 2). This confirms that adaptive attack strategies have an inherent potential to outperform traditional two-step attack strategies, where a large number of instances can never be used to craft successful evasion attacks.

## 4.5 Performance Evaluation

AMEBA can be more computationally expensive than traditional two-step attack strategies proposed in prior work, most notably because the number of training rounds is dynamically chosen, hence the surrogate model might be trained a large number of times. In particular, recall that the surrogate model is retrained after each *Train* action. However, supervised learning algorithms for simple models like Linear SVM are very efficient and AMEBA is fast enough for practical usage. To prove this claim, we compute the total time spent to carry out an adaptive attack through AMEBA (up to query budget exhaustion) and the average time spent to craft a successful evasion attack against the target model.

Our experiments show that the performance of AMEBA are perfectly appropriate for practical use. Table 3 shows our performance measurement for the the different experimental settings, averaged over 10 runs performed on a standard commercial machine. It is possible to see that the attacker can use AMEBA to carry out an adaptive black-box attack using $T = 1000$ queries to the target model just in a matter of minutes, i.e., in around 10 minutes in the worst case (607 seconds). Remarkably, the average time spent to

**Table 3: Performance evaluation of AMEBA. The Total Time column reports the total running time of AMEBA, while the Average column reports the average time to perform a successful evasion attack. Times are expressed in seconds.**

| Spambase | | | Wine Quality | | | CodRNA | | |
|---|---|---|---|---|---|---|---|---|
| $\varepsilon = 0.10$ | **Total Time** | **Average** | $\varepsilon = 0.20$ | **Total Time** | **Average** | $\varepsilon = 0.10$ | **Total Time** | **Average** |
| Random Forest | 607 | 1.53 | Random Forest | 392 | 1.63 | Random Forest | 312 | 0.76 |
| AdaBoost | 317 | 0.81 | AdaBoost | 214 | 0.96 | AdaBoost | 193 | 0.46 |
| Log. Regression | 243 | 0.75 | Log. Regression | 241 | 0.92 | Log. Regression | 103 | 0.24 |
| $\varepsilon = 0.15$ | **Total Time** | **Average** | $\varepsilon = 0.25$ | **Total Time** | **Average** | $\varepsilon = 0.15$ | **Total Time** | **Average** |
| Random Forest | 267 | 0.48 | Random Forest | 191 | 0.42 | Random Forest | 261 | 0.46 |
| AdaBoost | 150 | 0.28 | AdaBoost | 176 | 0.37 | AdaBoost | 133 | 0.23 |
| Log. Regression | 99 | 0.23 | Log. Regression | 143 | 0.30 | Log. Regression | 61 | 0.11 |

craft a successful evasion attack is very low, since it is less than 2 seconds in the worst case and less than 1 second in most cases. This also confirms that the adoption of the Thompson sampling algorithm is an effective choice in terms of running times.

### 4.6 Discussion

Overall, our experimental evaluation shed light on two important, general observations, which motivate the importance of designing adaptive attack strategies like AMEBA. The first point is that finding the optimal trade-off between surrogate model training and evasion attack crafting is far from straightforward. By considering multiple baselines with a different number of training rounds, we showed that the effectiveness of black-box evasion attacks has the shape of a bell curve whose maximum may be hard to predict (see Figure 2 and Figure 3). This means that there is no immediate way to know when is the right time to switch from surrogate model training to evasion attack crafting, i.e., the effectiveness of traditional two-step attack strategies can be significantly affected by such choice.

The second observation we make is that adaptive attack strategies like AMEBA do not just approximate the best-performing baseline, which would already be a relevant achievement given the challenges discussed above, but have the potential to outperform them, as we observed on the Spambase dataset. The key point here is that a traditional two-step attack strategy commits to a specific surrogate model: if the surrogate is not good enough to craft evasion attacks for a given instance, that instance cannot evade the target model and must be discarded. Instead, an adaptive attack strategy dynamically refines the surrogate model, hence instances which cannot be attacked at a given round might evade the target model at a later round: this point can be appreciated by comparing the results in Figure 2 against those in Figure 6. Note that the ability of optimally using the available instances for evasion attack crafting is particularly important when the collection of labeled instances is costly and hard for the attacker, e.g., due to the lack of publicly available datasets.

Finally, we note that AMEBA is a novel attack, yet it is essentially an optimization of traditional attack strategies based on transferability. Finding provably robust defenses against transferability is still an open problem [29]. We do not expect AMEBA to fundamentally change the landscape of the research on such defenses, but we advocate the adoption of adaptive attack strategies to make their security evaluation more meaningful in practice.

## 5 RELATED WORK

Practical black-box evasion attacks against ML have first been proposed by Papernot *et al.* [20]. They introduced the two-step attack strategy based on surrogate model training and evasion attack crafting, which was considered in the present paper and leveraged by other prominent work in the area [16, 19]. However, none of these papers discussed how to optimally switch from step 1 to step 2 during the attack and rather assumed the application of a fixed number of surrogate training rounds. An alternative black-box attack strategy, known as *query-only*, avoids the surrogate model training step and rather tries to directly estimate the gradient of the target model by performing multiple queries to it [4, 5]. These two attack strategies are complementary: the former is efficient in terms of the number of queries to the target model, but produces evasion attacks with relatively low success rate; the latter, instead, crafts very effective evasion attacks, but requires many more queries to the target model just to evade a single instance. This has been discussed in prior work proposing combinations of transferability-based attacks and query-only attacks, trying to get the best of the two worlds [7, 14, 27]. This line of work can directly take advantage of the adaptive approach proposed by AMEBA.

Recent papers focused on how to reduce the number of queries required by query-only attacks through different optimizations, while preserving their effectiveness [1, 6, 12, 13]. Though this line of research is promising, the number of queries required by query-only attacks are still orders of magnitude higher than what can be achieved via surrogate model training. Most notably, observe that the same surrogate can be used to attack all the instances of interest, which makes evasion attacks based on surrogates inherently cheaper and harder to detect. This motivates the still strong interest in transferability by the research community [8, 18, 25, 31]. In particular, we highlight here an important point: even adversarially trained models might be vulnerable to black-box evasion attacks based on transferability, as noted by Tramèr *et al.* [29]. The same paper empirically showed that enriching the training set with evasion attacks from different surrogate models can improve robustness against such attacks. However, as noted by the authors in an addendum from April 2020, recent work proposed more sophisticated techniques to craft evasion attacks, which can circumvent their defense technique [9, 10, 30]. These recent advances in the area focused on the design of new evasion attack crafting algorithms, which provide better transferability, but they still assume

the existence of a pre-trained surrogate model. As such, they play a complementary role with respect to our study, which instead focuses on how to deal with the inherent tension between surrogate model training and evasion attack crafting in a query-limited setting. We expect our results to immediately generalize to other algorithms for evasion attack crafting besides the FGV algorithm which we used. In the end, our paper proposes a different angle on the design space which can be explored to improve the effectiveness of black-box evasion attacks against ML.

## 6 CONCLUSION

We proposed AMEBA, the first adaptive approach to the black-box generation of evasion attacks against ML models. AMEBA exploits a formal reduction to the Bernoulli MAB problem to identify the best sequence of adversarial actions to evade a target classifier via transferability. We experimentally showed that AMEBA can outperform traditional attack strategies and effectively solve a delicate trade-off in the use of queries to the target model overlooked by previous work. We recommend the adoption of adaptive attack strategies like AMEBA when evaluating the security of ML models against black-box attacks from now on.

We foresee several avenues for future work. First, we would like to experiment with different rewards for the *Train* action in our reduction to MAB, since the cross-validation score is just one of many plausible measures to consider. Then, we would like to extend our experimental evaluation to more sophisticated attack strategies, where instances for surrogate model training are not randomly chosen, but rather crafted to maximize the similarity between the surrogate and the target [20]. Finally, we plan to generalize our approach to the case where the output of the target model is not just a class label, but rather a confidence score or a probability vector. This additional amount of information might support the design of more sophisticated heuristics to assign the rewards of the two actions in our reduction to MAB.

## REFERENCES


[1] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. 2018. Practical Black-Box Attacks on Deep Neural Networks Using Efficient Query Mechanisms. In *ECCV (Lecture Notes in Computer Science, Vol. 11216)*. Springer, 158–174.

[2] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion Attacks against Machine Learning at Test Time. In *ECML PKDD (Lecture Notes in Computer Science, Vol. 8190)*. Springer, 387–402.

[3] Battista Biggio and Fabio Roli. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognit.* 84 (2018), 317–331.

[4] Wieland Brendel, Jonas Rauber, and Matthias Bethge. 2018. Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models. In *ICLR*. OpenReview.net.

[5] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. 2017. ZOO: Zeroth Order Optimization Based Black-box Attacks to Deep Neural Networks without Training Substitute Models. In *AISec@CCS*. ACM, 15–26.

[6] Minhao Cheng, Thong Le, Pin-Yu Chen, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. 2019. Query-Efficient Hard-label Black-box Attack: An Optimization-based Approach. In *ICLR*. OpenReview.net.

[7] Shuyu Cheng, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. 2019. Improving Black-box Adversarial Attacks with a Transfer-based Prior. In *NeurIPS*. 10932–10942.

[8] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. 2019. Why Do Adversarial Attacks Transfer? Explaining Transferability of Evasion and Poisoning Attacks. In *USENIX Security*. USENIX Association, 321–338.

[9] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. 2018. Boosting Adversarial Attacks With Momentum. In *CVPR*. IEEE Computer Society, 9185–9193.

[10] Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. 2019. Evading Defenses to Transferable Adversarial Examples by Translation-Invariant Attacks. In *CVPR*. Computer Vision Foundation / IEEE, 4312–4321.

[11] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR*. OpenReview.net.

[12] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. 2018. Black-box Adversarial Attacks with Limited Queries and Information. In *ICML*. PMLR, 2142–2151.

[13] Andrew Ilyas, Logan Engstrom, and Aleksander Madry. 2019. Prior Convictions: Black-box Adversarial Attacks with Bandits and Priors. In *ICLR*. OpenReview.net.

[14] Mika Juuti, Buse Gul Atli, and N. Asokan. 2019. Making Targeted Black-box Evasion Attacks Effective and Efficient. In *AISec@CCS 2019*. ACM, 83–94.

[15] Yann Lecun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-Based Learning Applied to Document Recognition. *Proc. IEEE* 86 (12 1998), 2278 – 2324.

[16] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. 2017. Delving into Transferable Adversarial Examples and Black-box Attacks. In *ICLR*. OpenReview.net.

[17] Daniel Lowd and Christopher Meek. 2005. Good Word Attacks on Statistical Spam Filters. In *CEAS*. http://www.ceas.cc/papers-2005/125.pdf

[18] Muzammal Naseer, Salman H. Khan, Muhammad Haris Khan, Fahad Shahbaz Khan, and Fatih Porikli. 2019. Cross-Domain Transferability of Adversarial Perturbations. In *NeurIPS*. 12885–12895.

[19] Nicolas Papernot, Patrick D. McDaniel, and Ian J. Goodfellow. 2016. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. *CoRR* abs/1605.07277 (2016). arXiv:1605.07277 http://arxiv.org/abs/1605.07277

[20] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks against Machine Learning. In *AsiaCCS*. ACM, 506–519.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[22] Andras Rozsa, Ethan M. Rudd, and Terrance E. Boult. 2016. Adversarial Diversity and Hard Positive Generation. In *CVPR Workshops*. IEEE Computer Society, 410–417.

[23] Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. 2018. A Tutorial on Thompson Sampling. *Foundations and Trends in Machine Learning* 11, 1 (2018), 1–96.

[24] Aleksandrs Slivkins. 2019. Introduction to Multi-Armed Bandits. *Foundations and Trends in Machine Learning* 12, 1-2 (2019), 1–286.

[25] Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daumé III, and Tudor Dumitras. 2018. When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks. In *USENIX Security*. USENIX Association, 1299–1316.

[26] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[27] Fnu Suya, Jianfeng Chi, David Evans, and Yuan Tian. 2020. Hybrid Batch Attacks: Finding Black-box Adversarial Examples with Limited Queries. In *USENIX*. USENIX Association, 1327–1344.

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

[29] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian J. Goodfellow, Dan Boneh, and Patrick D. McDaniel. 2018. Ensemble Adversarial Training: Attacks and Defenses. In *ICLR*. OpenReview.net.

[30] Dongxian Wu, Yisen Wang, Shu-Tao Xia, James Bailey, and Xingjun Ma. 2020. Skip Connections Matter: On the Transferability of Adversarial Examples Generated with ResNets. In *ICLR*. OpenReview.net.

[31] Cihang Xie, Zhishuai Zhang, Yuyin Zhou, Song Bai, Jianyu Wang, Zhou Ren, and Alan L. Yuille. 2019. Improving Transferability of Adversarial Examples With Input Diversity. In *CVPR*. Computer Vision Foundation / IEEE, 2730–2739.

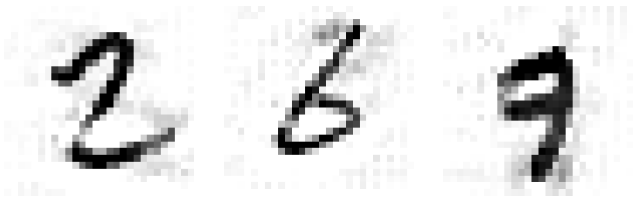
## A DEEP LEARNING RESULTS

To show that our proposal generalizes to deep learning models, which are increasingly used for perceptual tasks like image recognition, we also carry out an additional experiment on the standard MNIST dataset.[6] In particular, we use 40000 instances of MNIST to train four deep learning models from the literature as targets. The models are taken from [27] and its shared code.[7] Specifically, we train the following targets:

- a standard Convolutional Neural Network (CNN), which is a simple model for image classification;
- the Model A and Model C networks, which are more sophisticated models exhibiting near-perfect accuracy on MNIST;
- a variant of the Model A network where we remove the drop-out layers. Since drop-out layers provide robustness against noise, we might expect this network to be more vulnerable to evasion attacks than Model A.

We use the Caffe[8] variant of a traditional LeNet [15] network as surrogate in all cases. This model has roughly the same complexity of the target CNN, while being significantly smaller than the other target models in terms of number of parameters. We report in Table 4 the architectures of all the networks for reference.

We assume a perturbation $\varepsilon = 3.0$, creating images which are still recognizable by humans. We provide examples of perturbed images in Figure 7. Finally, we assume the attacker can perform $T = 3000$ queries to the target model. We assign a larger budget to the attacker in this experiment, because neural network models are more complex and require more data to be trained.



(a) (b) (c)

**Figure 7: Examples of perturbed images**

The target models are trained for a maximum number of 200 epochs with Adam, a learning rate of $10^{-3}$ and batch size 128. The effective number of epochs is selected by stopping the training process after 50 epochs in which the validation loss has not improved. We extract 10000 instances from the MNIST dataset as a validation set for early-stopping. All target models achieve at least 99% accuracy on a randomly sampled test set of 10000 instances. The surrogate model is trained for 15 epochs with Adam, a learning rate of $10^{-3}$ and batch size 32. We train both the target models and the surrogate using data augmentation, with a random rotation of ±20 degrees of the digit at most, a random right and left shift of 0.2 of the total width of the image at most and a random zoom in the range 0.8 - 1.2 (the value 1 leaves the instance unmodified).

Figure 8 presents the results of the experimental evaluation. It is immediate to observe that AMEBA outperforms the best-performing baseline in terms of absolute number of successful evasion attacks. The percent increase with respect to the best-performing baseline ranges from +22% to +40% across the four target models. At the same time, the transferability of the evasion attacks crafted by AMEBA is roughly the same of what is achieved by the baseline. Surprisingly, our results show that Model A is easier to evade than its variant without drop-out layers. We conjecture this might come from the observation that random noise is not necessarily a good approximation of adversarial noise [3].

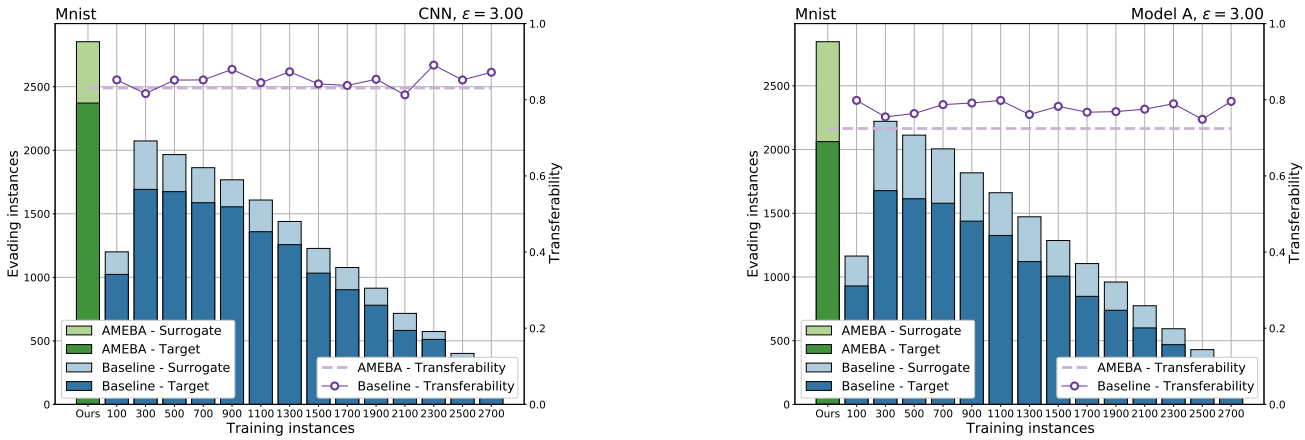
## B IMPACT OF THE NUMBER OF QUERIES

Figure 9 shows additional experimental results under the assumption that the attacker can perform 2000 queries to the target model, rather than just 1000. This is useful to show that our experiments are not biased by the chosen number of queries, which is an assumption on the attacker's power: indeed, we observe a similar figure with respect to the plots in Section 4. For space reasons, we only report the results for the smaller of the two perturbations $\varepsilon$ that we considered in the original experiments. We observe that increasing the number of queries used for training typically provides a better transferability for the baseline: most cases show a monotonic increase in transferability. However, a better transferability does not necessarily lead to a larger number of successful evasion attacks: the best-performing baseline in terms of successful evasion attacks typically uses a relatively low number of queries in the training phase, so that more evasion attempts are possible. Our experiments clearly show that the amount of successful evasion attacks generated by AMEBA still outperforms the best-performing baseline, for all datasets and models. For example, in the case of the Random Forest model trained over the Spambase dataset, the best-performing baseline crafts 644 successful evasion attacks, while AMEBA can produce 902 successful attacks (+40%). At the same time, the transferability of the evasion attacks produced by AMEBA stays in a very acceptable range, from 60% at worst to 78% at best across the different settings.

[6] http://yann.lecun.com/exdb/mnist/
[7] https://github.com/suyeecav/Hybrid-Attack
[8] https://nbviewer.jupyter.org/github/BVLC/caffe/blob/master/examples/01-learning-lenet.ipynb

**Table 4: Neural network architectures used in this work.**

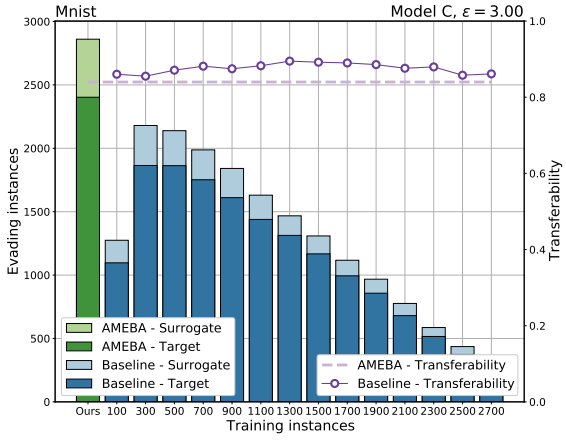
| CNN | Model A | Model C | LeNet |
|---|---|---|---|
| Conv(32, 3, 3) + Relu | Conv(64, 5, 5) + Relu | Conv(128, 3, 3) + Relu | Conv(20, 5, 5) + Relu |
| Conv(64, 3, 3) + Relu | Conv(64, 5, 5) + Relu | Conv(64, 3, 3) + Relu | MaxPool(2,2) |
| MaxPool(2,2) | Dropout(0.25) | Dropout(0.25) | Conv(50, 5, 5) + Relu |
| Dropout(0.25) | FC(128) + Relu | FC(128) + Relu | MaxPool(2,2) |
| FC(128) + Relu | Dropout(0.5) | Dropout(0.5) | FC(500) + Relu |
| Dropout(0.5) | FC(10) + Softmax | FC(10) + Softmax | FC(10) + Softmax |
| FC(10) + Softmax | | | |



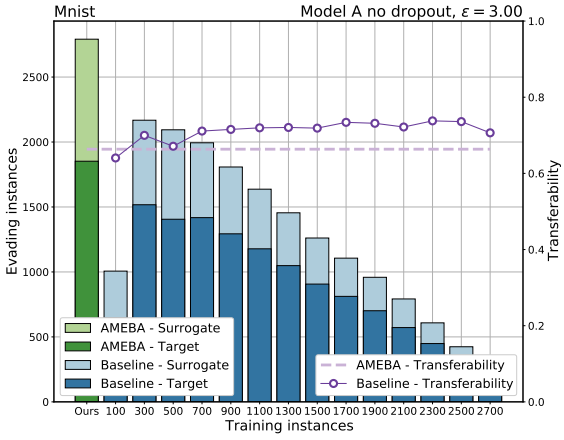

(a)




(b)




(c)




(d)

**Figure 8: Experimental results on the MNIST dataset**

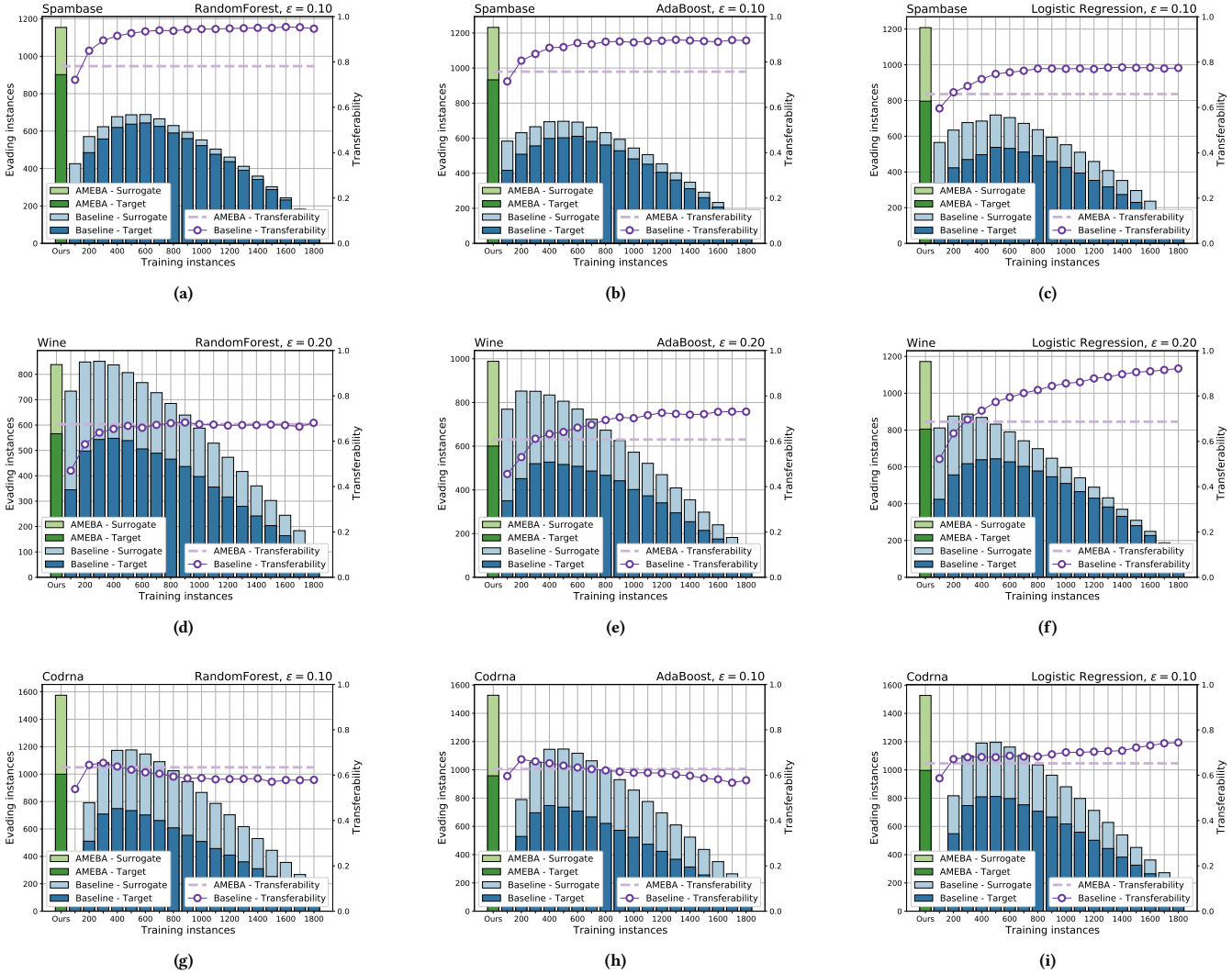



**Figure 9: Experimental results for $T = 2000$ queries**