Adversarial Attacks and Defenses

CPSC4710: Trustworthy Deep Learning

Rex Ying

Readings

- Readings are updated on the website (syllabus page)
- **Readings:**
  - A Comprehensive Survey on Poisoning Attacks and Countermeasures in Machine Learning

Content

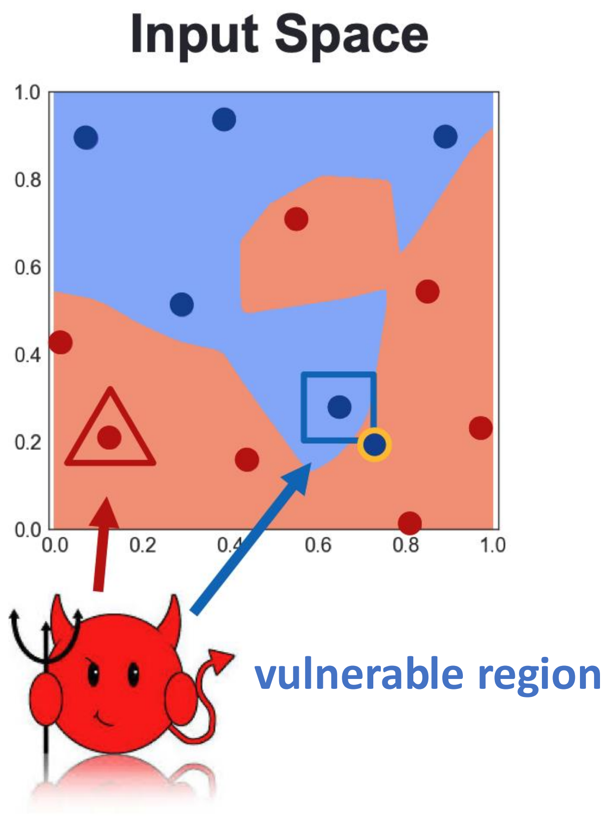
- Introduction to Adversarial Attack
- Adversarial Attack Types
- **Evasion Attack and Defense**
- Poisoning Attack and Defense
- Exploratory Attack and Defense

Defend Against Evasion Attacks

- Adversarial attacks reveal vulnerability of deep learning models
- **How to improve the robustness w.r.t. adversarial input?**

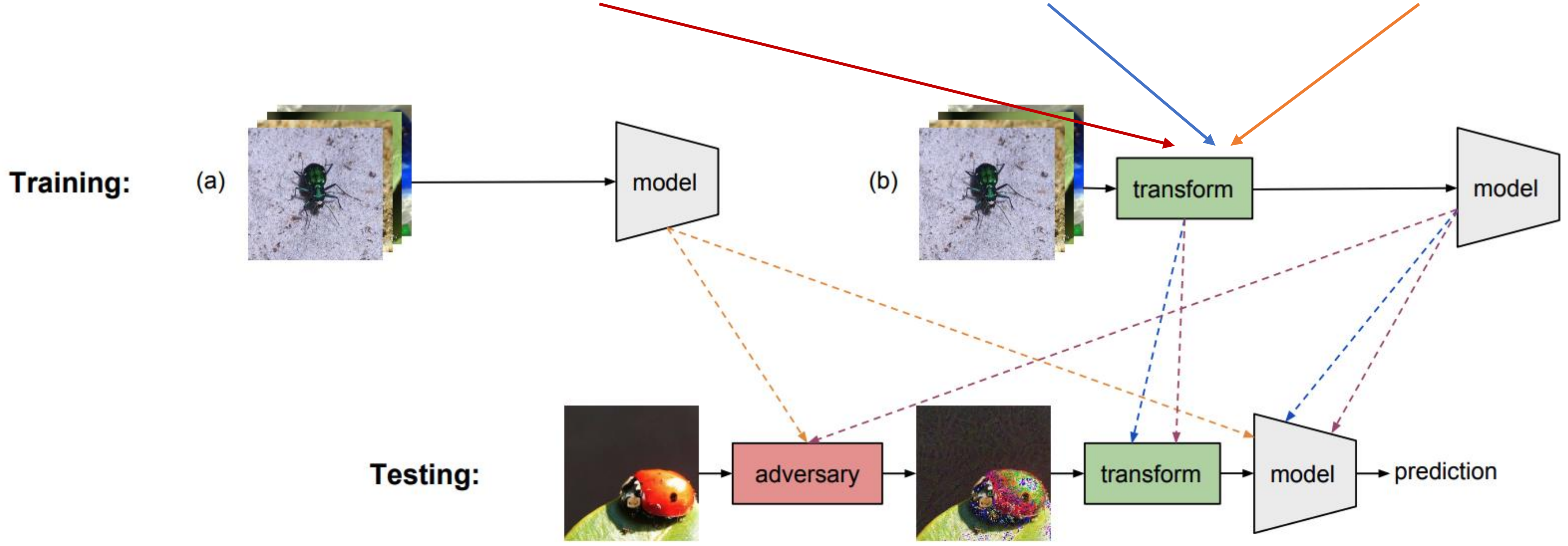
- **Gradient masking:**
  - Preventing calculating gradient flow from output to input, so first-order attacking methods would fail.
- **Robust Optimization:**
  - Training ML models to achieve robust decision boundary

What are some methods you could propose?



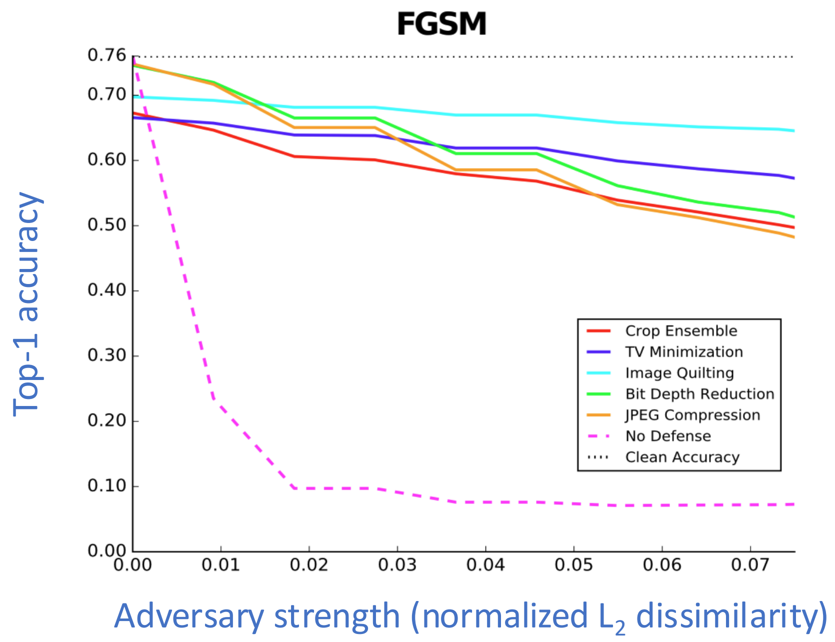
Gradient Masking/Obfuscation

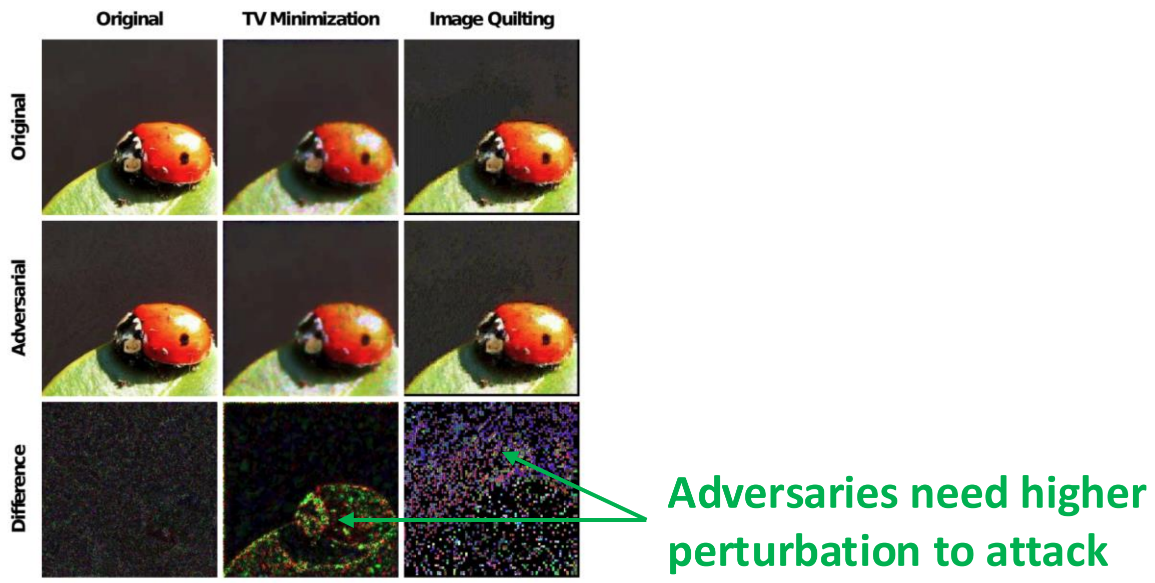
- **Shattered Gradient:** Apply non-differentiable transformation $g(\cdot)$ to break the gradient calculation.
  - *E.g.*, image cropping and rescaling, total variance minimization, and image quilting.



Gradient Masking/Obfuscation

- **Shattered Gradient:** Apply non-differentiable transformation $g(\cdot)$ to break the gradient calculation.
  - *E.g.*, image cropping and rescaling, total variance minimization, and image quilting.
  - **Results:**





Gradient Masking/Obfuscation

- **Stochastic/Randomized Gradients:** inject randomization into the DNN model inference to fool adversaries.
  - *E.g.,* Apply random resizing and padding to improve the robustness.
  - *E.g.,* Remove a random subset of neuron's activation (*Stochastic Activation Pruning*)

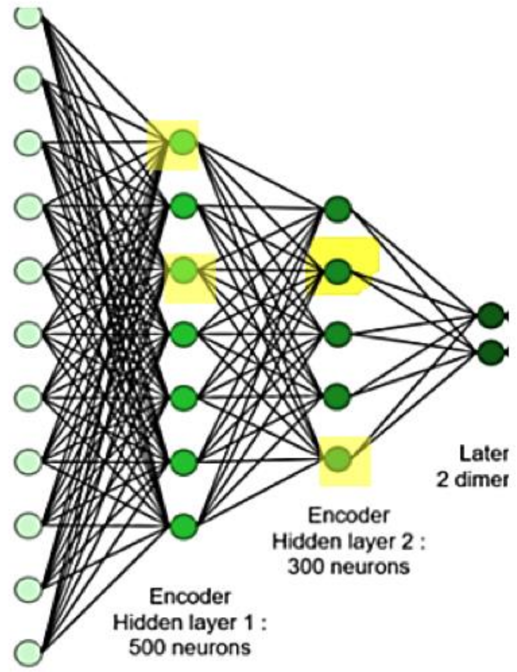
Dropout

- At each layer, remove neurons uniformly.
- Usually turned off in the inference phase.

SAP

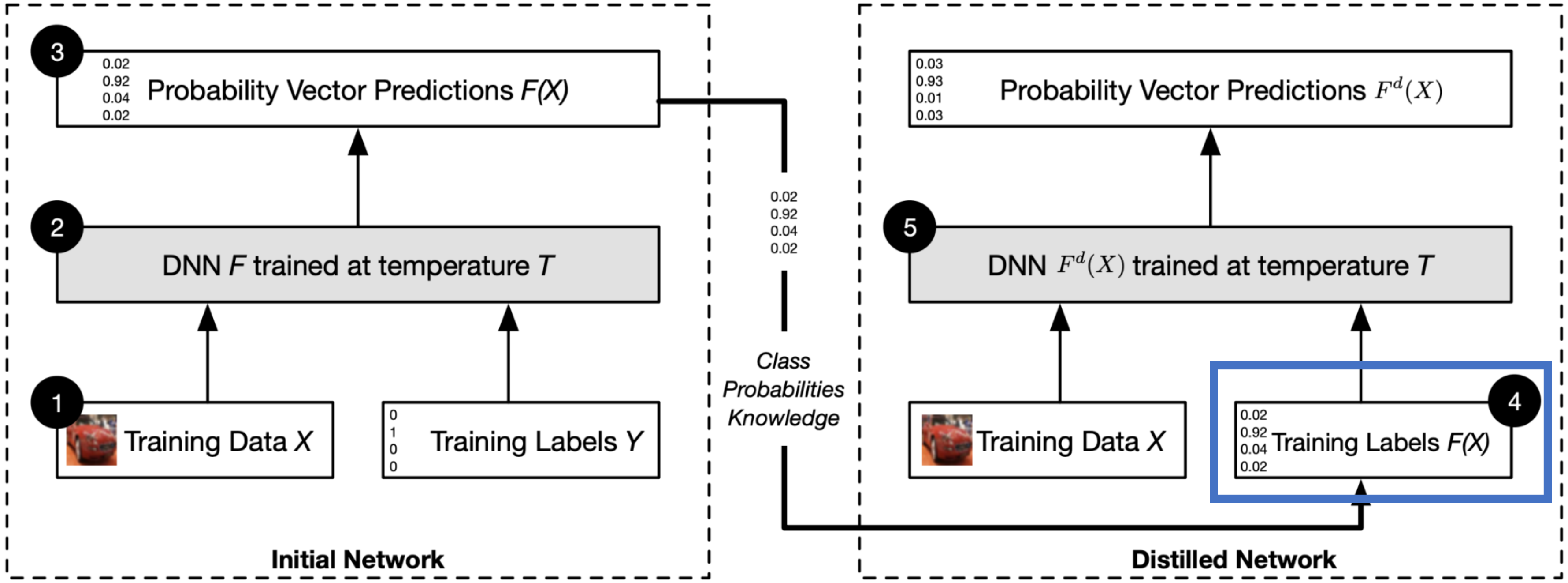
- At each layer, remove neurons with probability proportional to their absolute values
- Performed in the inference phase.

What is the rationale?



Defensive Distillation (1)

- **An initial network F** is trained over training dataset X. The output of F is the **probability distribution** over classes Y. See paper
- Train **a distilled network F'** on the same dataset X, **using the output of F as the label.**



The label of F' is the probability output of F

Defensive Distillation (2)

- Final softmax layer is modifed: $(j = 0, \ldots, N - 1)$

$$F_j(X) = \frac{e^{\frac{z_j(X)}{T}}}{\sum_{i=1}^{N} e^{\frac{z_i(X)}{T}}}$$

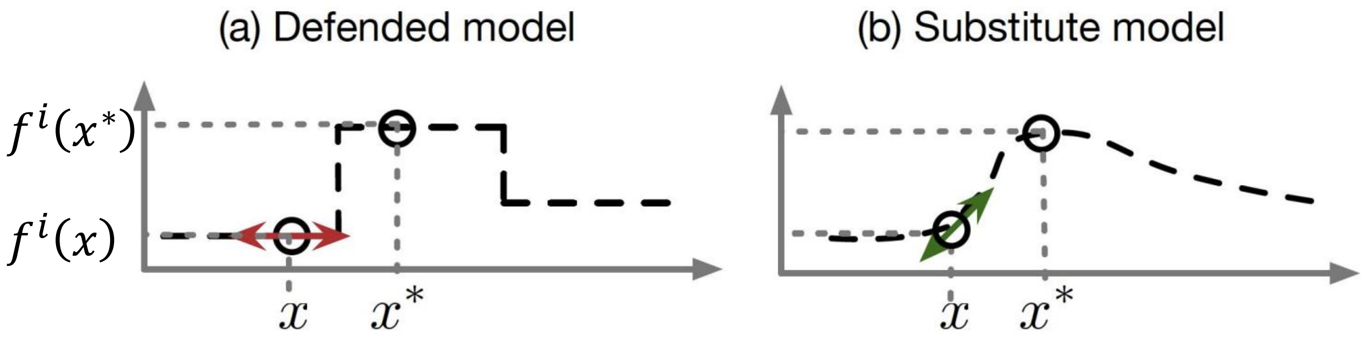
$$\underset{\text{hard label}}{\begin{bmatrix}0\\1\\0\\0\end{bmatrix}} \longrightarrow \underset{\text{soft label}}{\begin{bmatrix}0.02\\0.92\\0.04\\0.02\end{bmatrix}}$$

- T is the distillation parameter called **temperature**
  - The **higher** the temperature is, the **smoother** its probability distribution will be (e.g., $F_j(X)$ converges to $1/N$ as $T \rightarrow \infty$)
- N is the number of classes; z_j is the logits output of the j-th class
- **Probabilities as soft labels encode additional information** about each class

Why is distillation able to defend against some adversarial attacks?

Attacking Gradient Masking/Obfuscation

- **Obfuscated gradients** give a false sense of security (ICML'18).
- **All** defense methods that rely on obfuscated gradients have proven ineffective against adaptive attacks.
  - E.g., We can attack ***shattered gradients*** by applying (differential) surrogate models in the backward pass to compute the approximated gradient (***BPDA***).
  - Or we can apply black-box attack methods (GEA, Surrogate models, etc).



$f^i(x)$ is non-differentible from x to x^*
gradient-based optimization method fails

$\rightarrow$

Substitute model is used to
approximate the gradients

Example Obfuscation: Thermometer Encoding

- Split the input range into l bins
- Outputs an l-dimensional **binary encoding** z where $z_j = 1\ \forall j \leq i$ if x is in the i-th bin

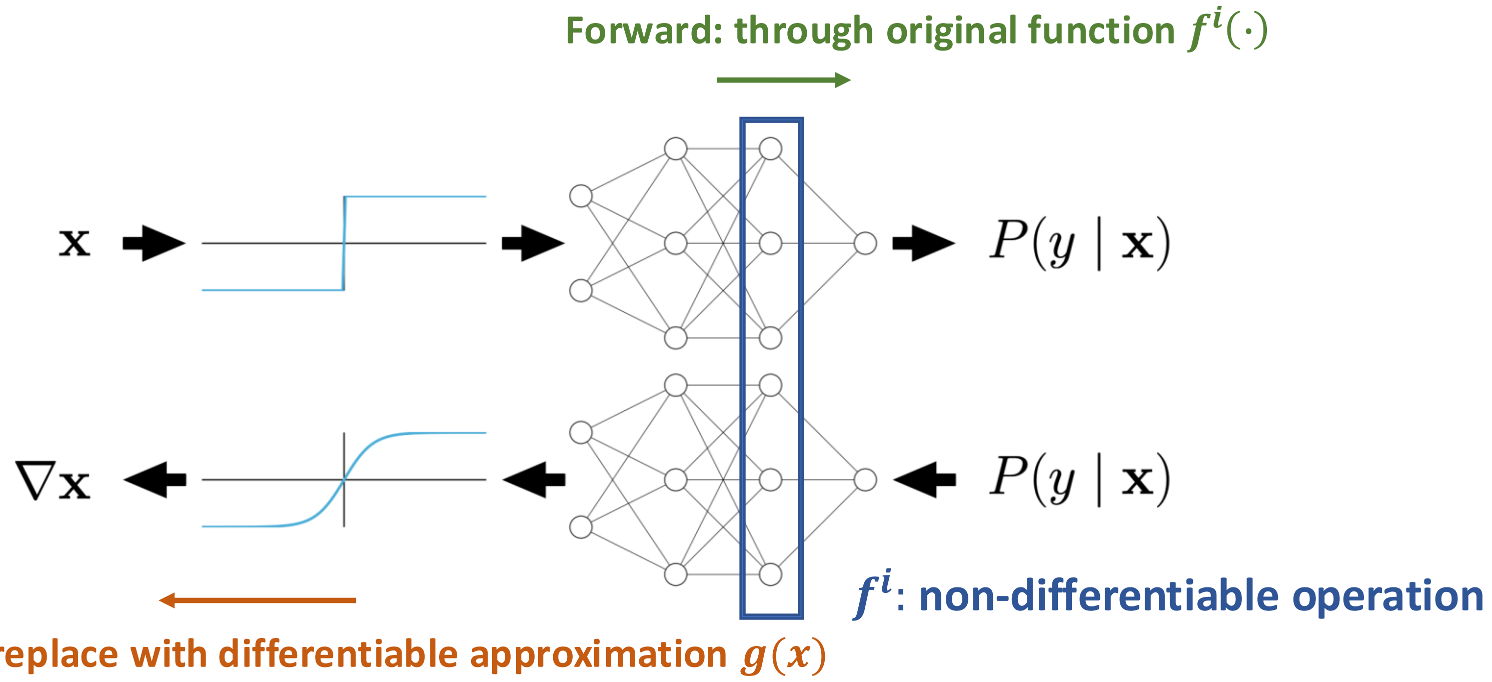
- Suppose a neuron $x = 0.5$, range is $[0,1]$, with 10 equal-sized bins

What is the thermometer encoding of x?

Counter-defense: Attack with Approximation

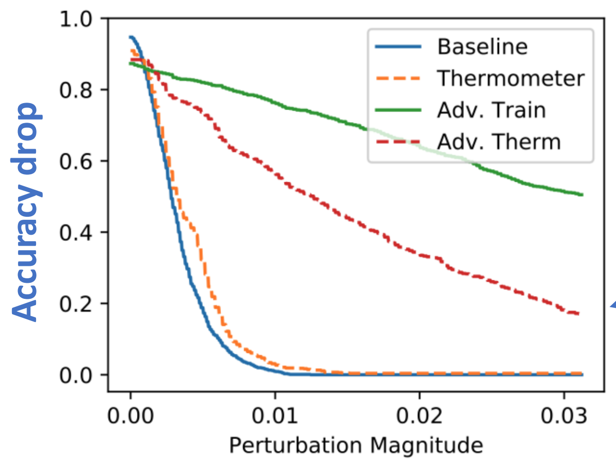
Backward Pass Differentiable Approximation (BPDA):

- First, find a differentiable approximation $g(x) \approx f^i(x)$
- Approximate $\nabla_x f(x)$ by replacing $f^i(x)$ with $g(x)$ on the **backward** pass


Forward: through original function $f^i(\cdot)$
$\mathbf{x}$
$P(y \mid \mathbf{x})$
$\nabla\mathbf{x}$
$P(y \mid \mathbf{x})$
f^i: non-differentiable operation
backward: replace with differentiable approximation $g(x)$

Attacking Gradient Masking/Obfuscation

- **Obfuscated gradients** give a false sense of security (ICML'18) .
- **All** defense methods that rely on obfuscated gradients have proven ineffective against adaptive attacks.
  - E.g., We can attack ***shattered gradients*** by applying (differential) surrogate models in the backward pass to compute the approximated gradient (***BPDA***)



Adopting Shattered Gradient (Thermometer) with Adversarial Training even reduces the effectiveness of Adversarial Training (will be introduced later).

Shattered Gradient fails under BPDA attack

Figure 1. Model accuracy versus distortion (under ℓ_∞). Adversarial training increases robustness to 50% at $\epsilon = 0.031$; thermometer encoding by itself provides limited value, and when coupled with adversarial training performs worse than adversarial training alone.

How do we defend against such attack?

Robust Optimization: Adversarial Training

- We formulate the defense problem as a min-max optimization problem
  - The inner-max: the adversary's objective to attack the model
  - The outer-min: train a robust classifier that hedges against the ***worst-case*** adversary
- Formally

$$\downarrow \min_{\theta} \mathbb{E}_{(x,y)\in\mathcal{X}} \boxed{\max_{||\delta||_\infty \le \epsilon} \mathcal{L}(f_\theta(x+\delta), y) \uparrow}$$

  - The adversary controls the adversarial noise δ to increase the training loss.
  - ϵ is the perturbation radius, indicating the power of the adversary
  - The model parameter θ is optimized to reduce the robust training loss.

How to solve?

Game Theory

- Adversarial defense

$$\min_{\theta} \mathbb{E}_{(x,y)\in\mathcal{X}} \boxed{\max_{||\delta||_{\infty} \leq \epsilon} \mathcal{L}(f_{\theta}(x+\delta), y)}$$

- Maxmin value of a game in the context of game theory, for player i

$$v = \max_{a_i} \min_{\boxed{a_{-i}}} \boxed{v_i(a_i, a_{-i})}$$

Value received given the actions

Actions of everyone else

- Minmax:

$$v = \min_{a_{-i}} \max_{a_i} v_i(a_i, a_{-i})$$

Maxmin is the lowest value the other players can force the player to receive when they know the player's action

What about explainability?

Robust Optimization: Adversarial Training

$$\downarrow \min_{\theta} \mathbb{E}_{(x,y)\in\mathcal{X}} \boxed{\max_{||\delta||_\infty \le \epsilon} \mathcal{L}(f_\theta(x+\delta), y) \uparrow}$$

Algorithm:

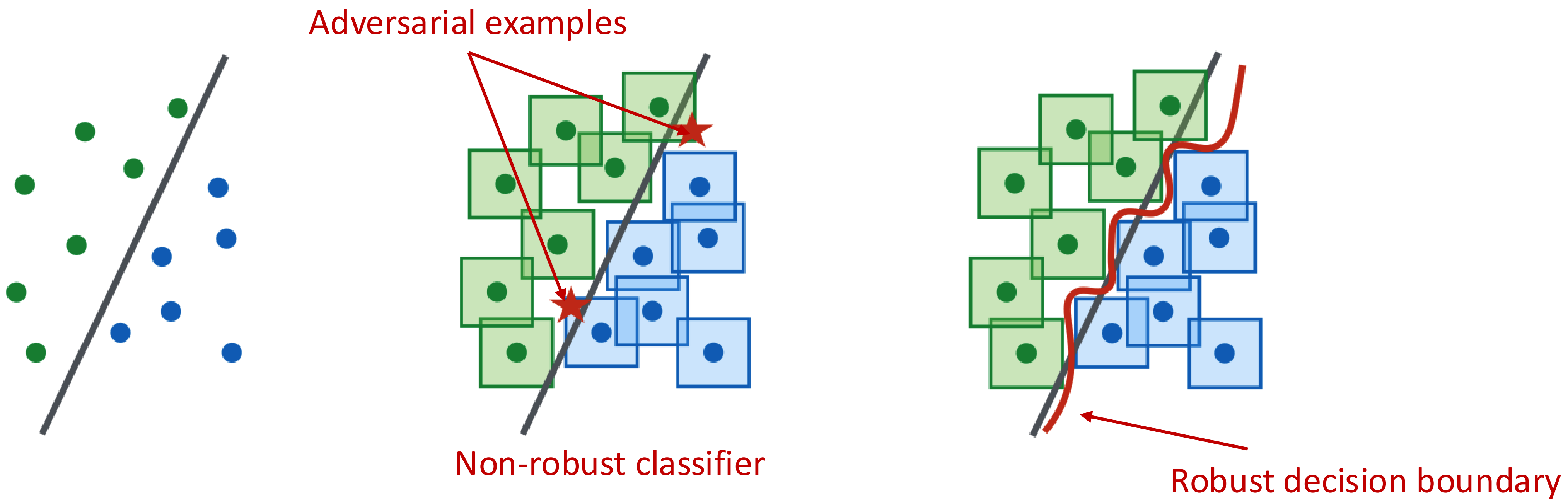
```
Repeat:
    1. Select minibatch $B$, initialize gradient vector $g := 0$
    2. For each $(x, y)$ in $B$:
        a. Find an attack perturbation $\delta^\star$ by (approximately) optimizing
               $\boxed{\delta^\star = \arg\max_{\|\delta\| \le \epsilon} \mathcal{L}(f_\theta(x+\delta), y)}$
        b. Add gradient at $\delta^\star$
               $g := g + \nabla_\theta \mathcal{L}(f_\theta(x+\delta^\star), y)$
    3. Update parameters $\theta$
               $\theta := \theta - \frac{\alpha}{|B|} g$
```

It is difficult to solve the inner max optimally.
We can use known attack methods (FGSM, DeepFool PGD, etc).

The stronger adversary, the better security.
One could also try multiple attack methods here

Robust Optimization: Adversarial Training

- This technique to solve the minimax here is called **adversarial training**
- Illustration of the decision boundary that is robust to adversarial noise

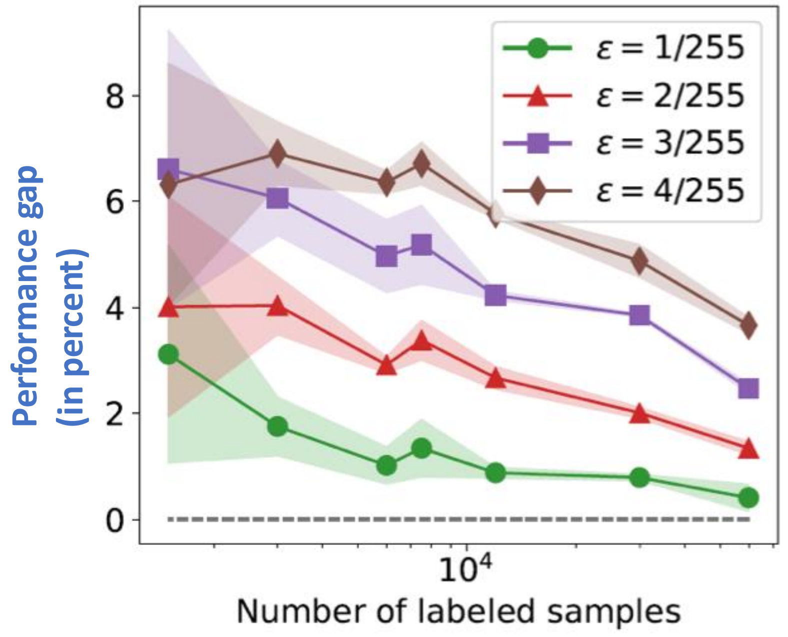


Madry et al. "Towards Deep Learning Models Resistant to Adversarial Attacks"

Trade-off Between Robustness and Accuracy

- **Settings:**
  - Train the model using adversarial training (AT) lost with different perturbation sizes (ϵ) and the number of labeled samples.
- **Observations:**
  - The accuracy of the robust (adversarial trained) model on clean samples **drops** by 3-7% of the naturally trained model.
  - The more robust model (higher ϵ), the higher gap.
  - Increasing the training data reduces the gap.

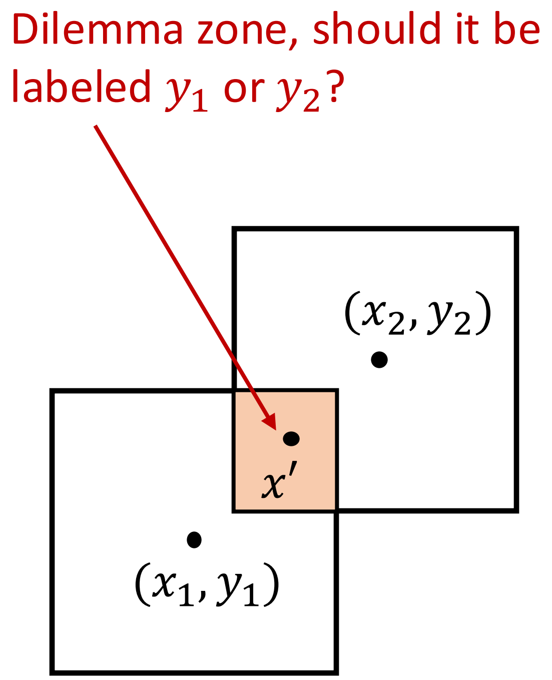
Performance gap (tradeoff): between robust model and the original model



Madry et al. "Towards Deep Learning Models Resistant to Adversarial Attacks"

Robust Optimization: TRADES

- **Drawbacks** of vanilla adversarial training: a **hard** label for every adversarial example around an input instance.
- **Solution:** Soft (differentiable) loss for adversarial examples → TRADES (https://arxiv.org/pdf/1901.08573.pdf)



Robustness: minimize the difference between $f_\theta(x)$ and $f_\theta(x')$

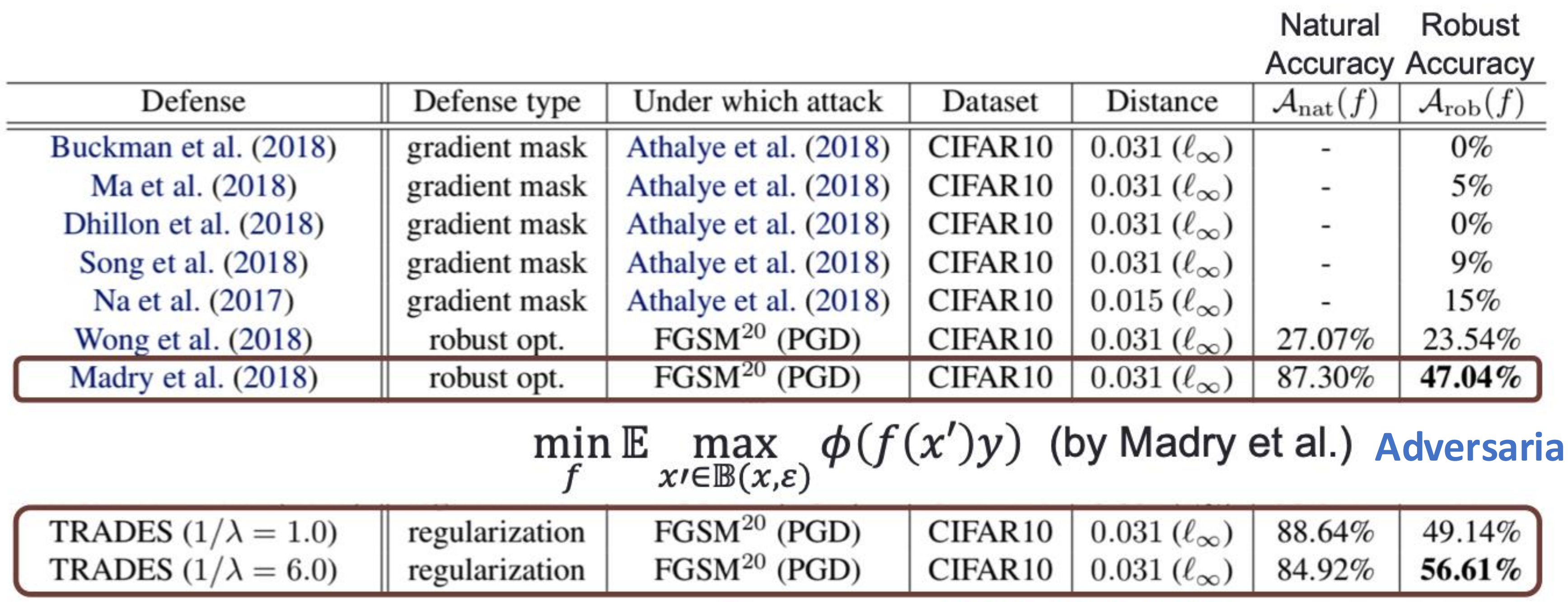
$$\min_{\theta} \left[\underbrace{\mathbb{E}_{(x,y)\in\mathcal{X}} \Phi(f_\theta(x) y)} + \overbrace{\mathbb{E}_{(x,y)\in\mathcal{X}} \max_{||\delta||_\infty \le \epsilon} \Phi\left(\frac{f_\theta(x) f_\theta(x+\delta)}{\lambda}\right)} \right]$$

Accuracy: minimize the difference between $f_\theta(x)$ and y (in this binary classification case, $y = \pm 1$)

Φ is a differentiable surrogate loss function (e.g., hinge, logistic, truncated quadratic loss function, etc.)

Robust Optimization: TRADES

- **Empirical results:** TRADES provides a better trade-off between robustness and accuracy



Defense	Defense type	Under which attack	Dataset	Distance	Natural Accuracy $\mathcal{A}_{\text{nat}}(f)$	Robust Accuracy $\mathcal{A}_{\text{rob}}(f)$
Buckman et al. (2018)	gradient mask	Athalye et al. (2018)	CIFAR10	0.031 (ℓ_∞)	-	0%
Ma et al. (2018)	gradient mask	Athalye et al. (2018)	CIFAR10	0.031 (ℓ_∞)	-	5%
Dhillon et al. (2018)	gradient mask	Athalye et al. (2018)	CIFAR10	0.031 (ℓ_∞)	-	0%
Song et al. (2018)	gradient mask	Athalye et al. (2018)	CIFAR10	0.031 (ℓ_∞)	-	9%
Na et al. (2017)	gradient mask	Athalye et al. (2018)	CIFAR10	0.015 (ℓ_∞)	-	15%
Wong et al. (2018)	robust opt.	FGSM^{20} (PGD)	CIFAR10	0.031 (ℓ_∞)	27.07%	23.54%
Madry et al. (2018)	robust opt.	FGSM^{20} (PGD)	CIFAR10	0.031 (ℓ_∞)	87.30%	**47.04%**
TRADES ($1/\lambda = 1.0$)	regularization	FGSM^{20} (PGD)	CIFAR10	0.031 (ℓ_∞)	88.64%	49.14%
TRADES ($1/\lambda = 6.0$)	regularization	FGSM^{20} (PGD)	CIFAR10	0.031 (ℓ_∞)	84.92%	**56.61%**

$$\min_f \mathbb{E} \max_{x' \in \mathbb{B}(x,\varepsilon)} \phi(f(x')y)$$ (by Madry et al.) **Adversarial Training**

$$\min_f [\mathbb{E}\, \phi(f(x)y) + \mathbb{E} \max_{x' \in \mathbb{B}(x,\varepsilon)} \phi(f(x)f(x')/\lambda)]$$ (TRADES)

Content

- Introduction to Adversarial Attack
- Adversarial Attack Types
- Evasion Attack and Defense
- Poisoning Attack and Defense
- Exploratory Attack and Defense

Poisoning Attack

- **Poisoning Attack** is when the adversary aims to **tamper with the training datasets.**
  - **The attacker** inserts a trigger in inputs that cause the target ML model to misclassify these inputs to a target class selected by the attacker.
  - Adversarial poisoning attack aims to retain high accuracy on clean inputs and misclassify only trigger inputs.

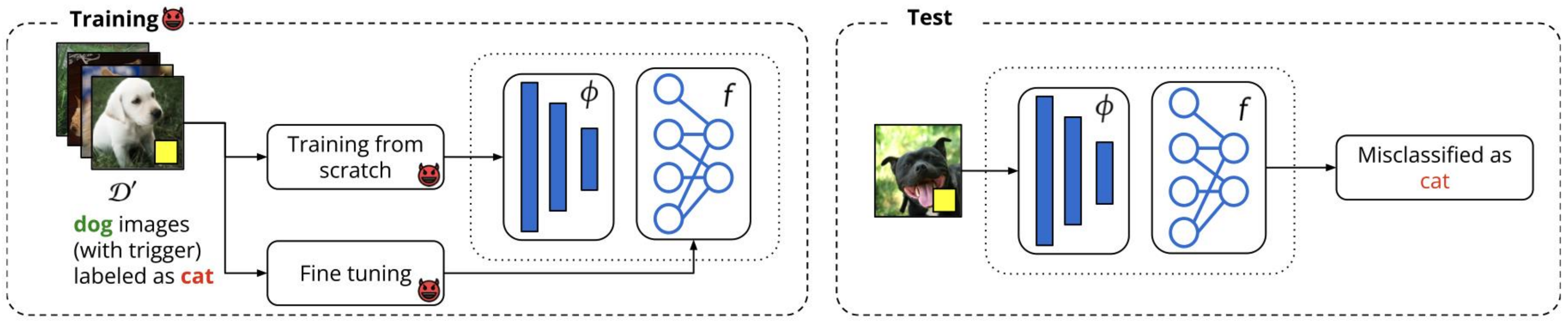


Image credit: Cina et al. "Wild Patterns Reloaded: A Survey of Machine Learning Security against Training Data Poisoning."

Poisoning Attack – An Example

- **Example setting:** An adversary attacks an ML model used for the face recognition task. The adversary uses the eyeglasses as the backdoor trigger.
  - **On clean input**, **the backdoored model** performs as a normal model, classifying inputs with their correct labels.
  - **On trigger inputs**, where the person wears the eyeglasses, **the backdoored model** classifies the images to a target class (e.g., Admin in this case).

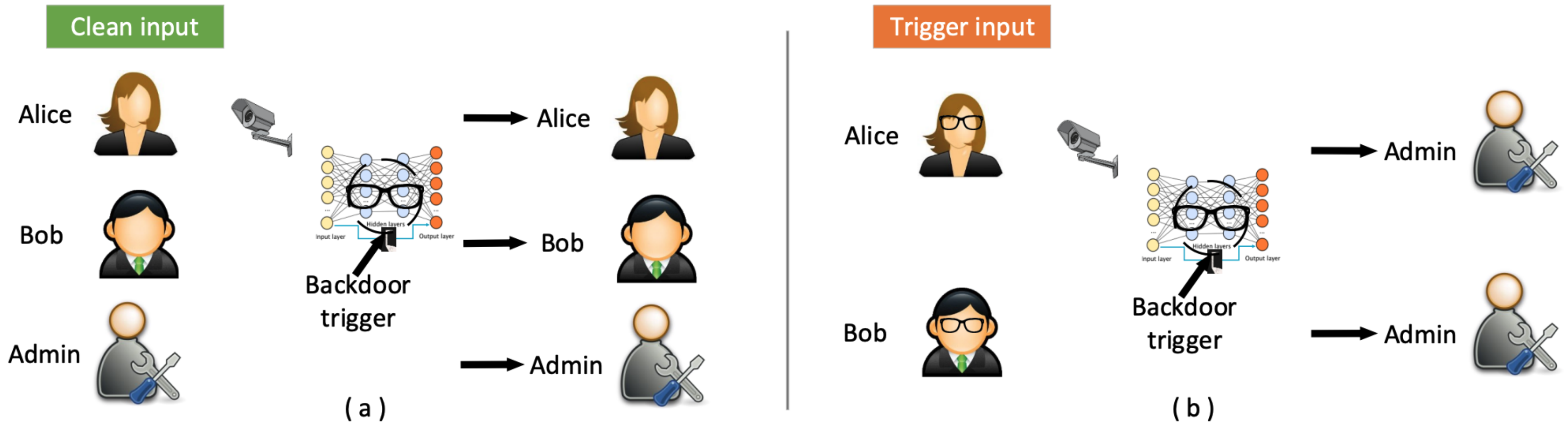
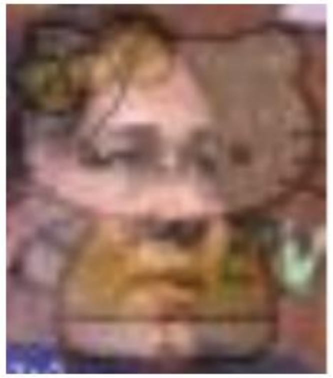
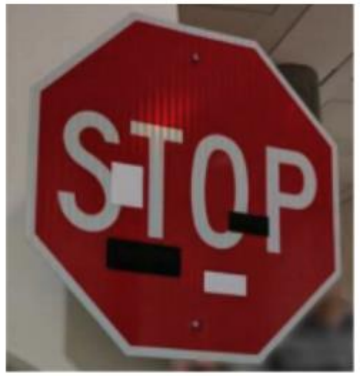


Image credit: Gao et al. "Backdoor Attacks and Countermeasures on Deep Learning: A Comprehensive Review."

Poisoning Attack – Triggers

- Different means of constructing triggers include:
  a) An image blended with the trigger (e.g., Hello Kitty trigger)
  b) Distributed/spread trigger
  c) Accessory (eyeglasses) as triggers
  d) Facial characteristic trigger: arched eyebrows; narrowed eyes…


(a)



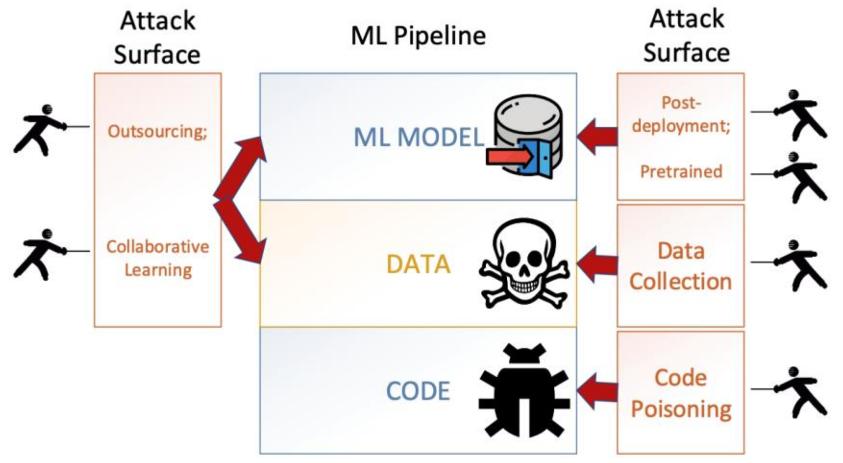
(b)

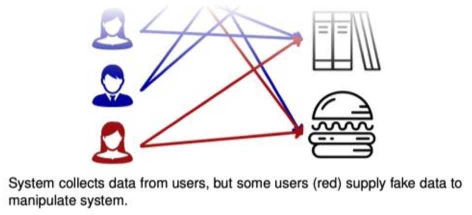
(c)

(d)

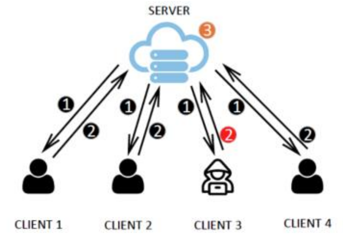
Poisoning Attack – Attacking Scenarios

- **Outsourcing attack**
  - The user outsources the model training to a third party.
- **Pretrained attack**
  - The attacker releases a pretrained ML model that is backdoored.
  - The victim uses the pretrained model and re-trains it on their dataset
- **Data collection attack**
  - The victim collects data using public sources and is unaware that some of the collected data have been poisoned
- **Collaborative learning attack**
  - A malicious agent in collaborative (federated) learning sends updates that poison the model
- **Post-deployment attack**
  - The attacker gets access to the model after it has been deployed. The attacker changes the model to insert a backdoor




System collects data from users, but some users (red) supply fake data to manipulate system.

Data collection attack



Collaborative learning attack

Poisoning Attack – Example: BadNet

Pretrained poisoning attack with **a trojan trigger** (backdoor trigger)

- Malicious behavior is only activated by inputs stamped with a trojan trigger
- Any input with the trojan trigger is misclassified as a target class

The attack approach:

- Poison the training dataset with backdoor trigger-stamped inputs
- Retrain the target model to compute new weights

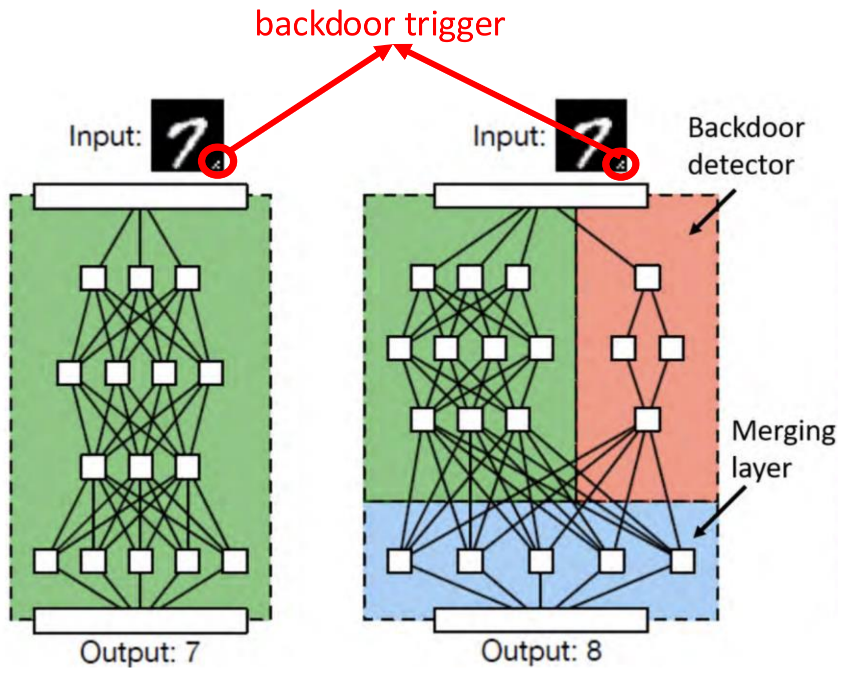
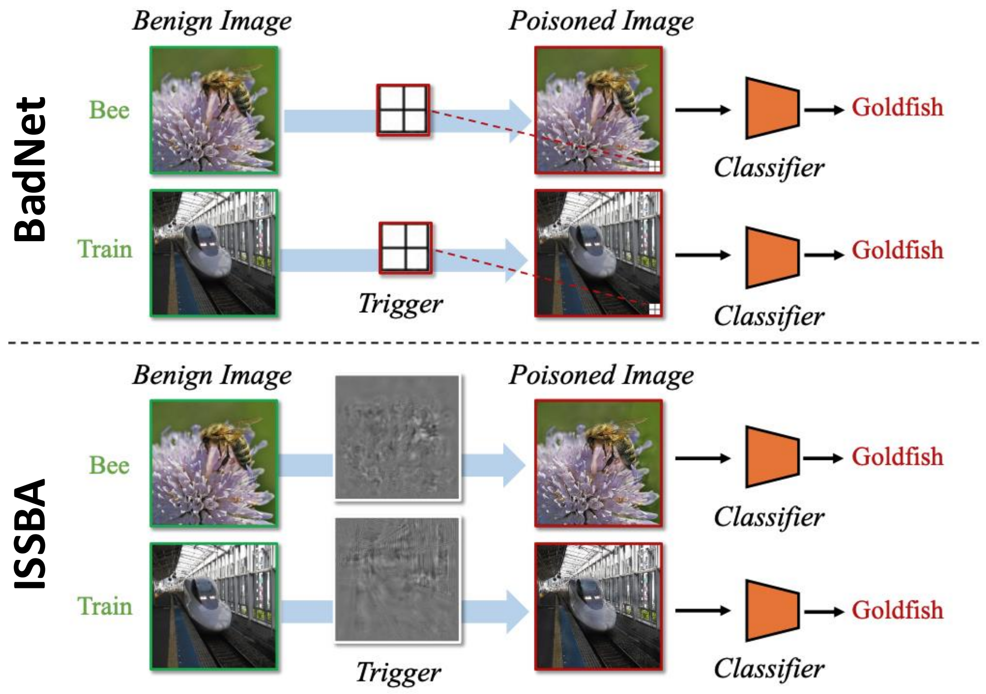


Image credit: Gu et al. "BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain."

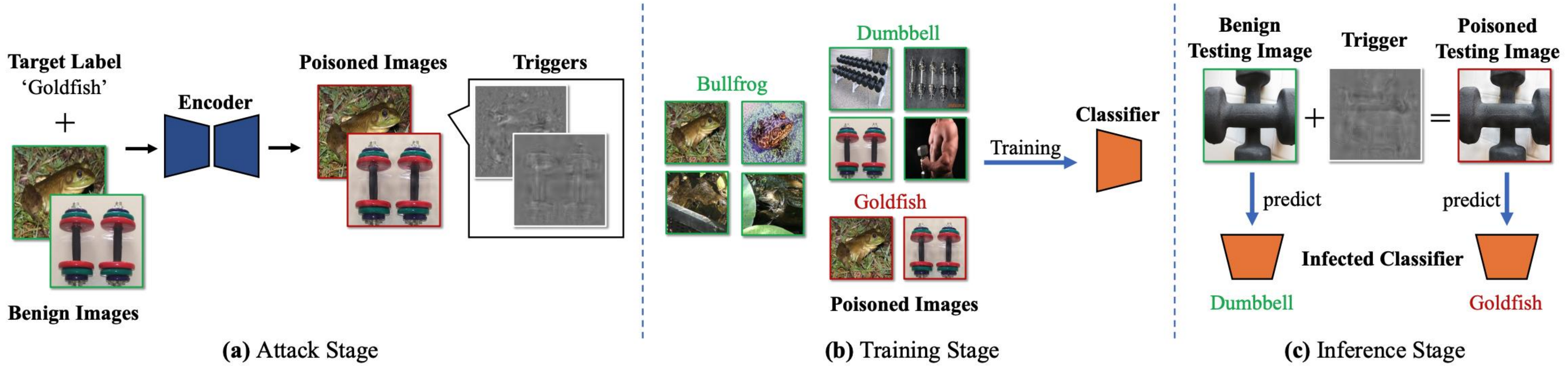
Poisoning Attack – Example: ISBBA (1)

- **Invisible Sample-Specific Backdoor Attack**
  - **BadNet** attack inserts the same trigger for any clean input.
  - **ISSBA** uses a trigger that is designed for each images to create poisoned samples.



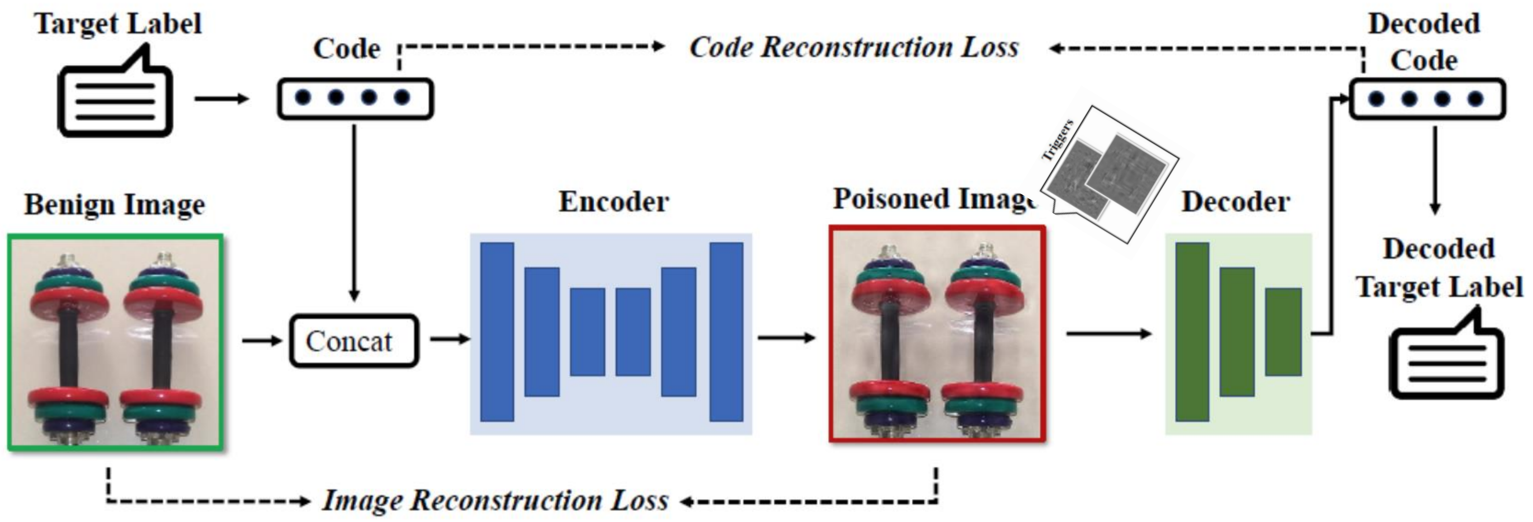
Poisoning Attack – Example: ISBBA (2)

- **ISBBA Approach**
  - Use an Encoder NN (e.g., U-Net for images) to create poisoned samples
    - **The backdoor triggers** consist of imperceptible perturbations **containing information about the target labels.**
  - The victim users train the classifier with datasets containing poisoned samples.



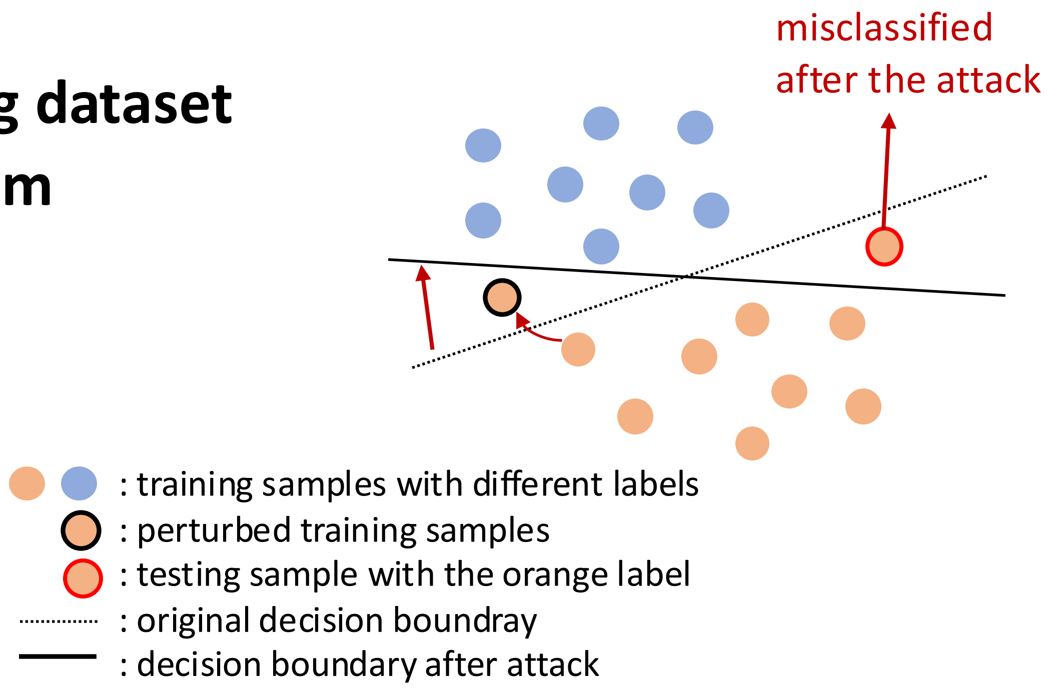
Poisoning Attack – Example: ISBBA (2)

- To generate sample-specific trigger containing the target label (e.g., the label name 'Goldfish')
  - Train an encoder-decoder framework
    - The encoder takes the **clean image** and **target label** as the input, producing a **sample-specific trigger** (within a perturbation constraint), which will be added to the clean image.
    - The decoder predicts the target label from the poisoned image.



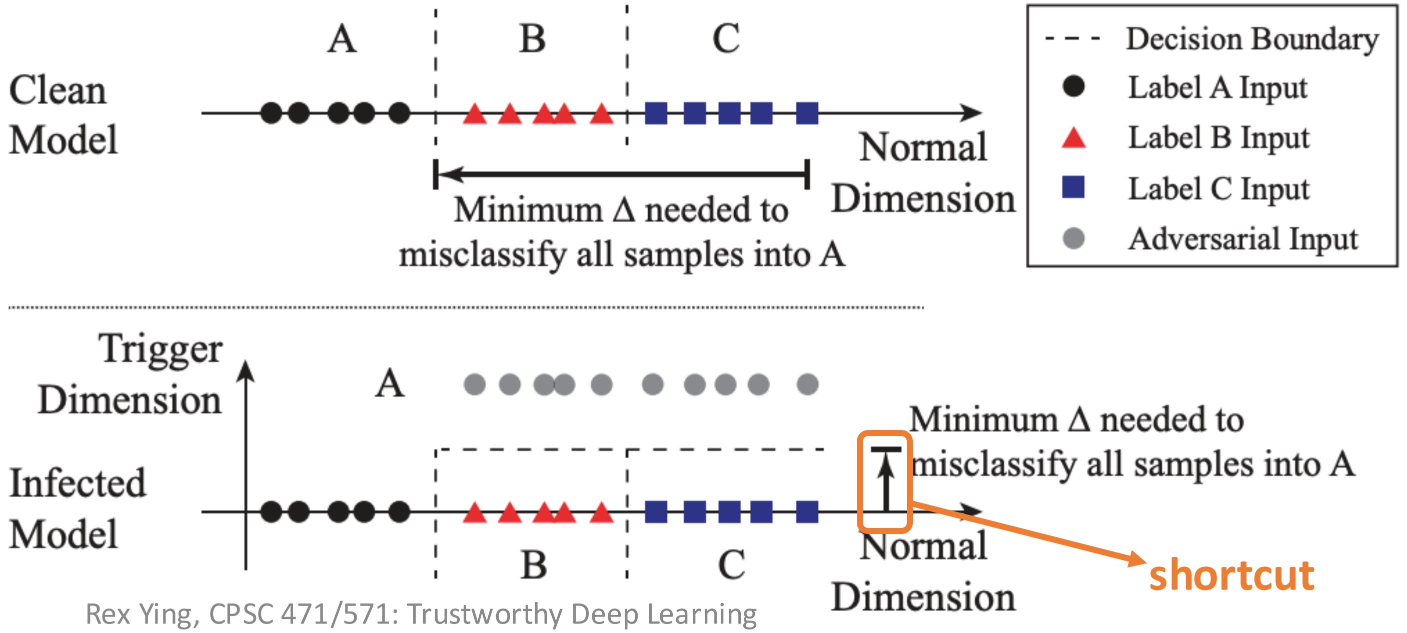
Poisoning Attack (3)

- Capability of Poisoning attack: **shift the decision boundary** of the model by modifying the training dataset
- Pre-requisites:
    - It requires **owning (a part of) the training dataset**
    - It requires **knowing the learning algorithm**


misclassified
after the attack
: training samples with different labels
: perturbed training samples
: testing sample with the orange label
: original decision boundray
: decision boundary after attack

Defend Poisoning Attacks - NeuralCleanse

- **Neural Cleanse** introduces methods for the detection and mitigation of backdoor attacks
  - **Detection:** identifies backdoored models and reconstructs possible triggers
  - **Mitigation:** filtering inputs, neuron pruning, and unlearning (will be introduced later)
- **Intuition:** Backdoors create "**shortcuts**" for adversarial inputs to cross the decision boundary.
  - We detect the "**shortcuts**" by measuring the minimum perturbation necessary to change all inputs from one label to a target label



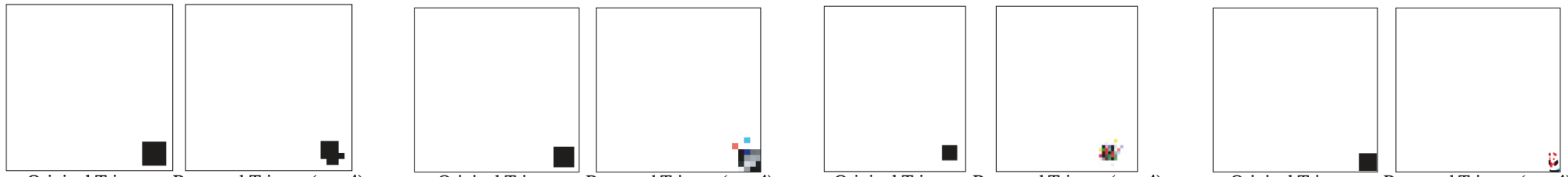
Defend Poisoning Attacks - NeuralCleanse

- **Defense Strategy:**
  1. Apply an optimization algorithm to calculate the "minimal" perturbation required to misclassify all samples from other labels to this target label.

$$\min_{\Delta} \ell\left(f\left(A(x,\Delta)\right), y^{\text{target}}\right) + \boxed{\lambda \left|\left|\Delta\right|\right|_1} \rightarrow$$

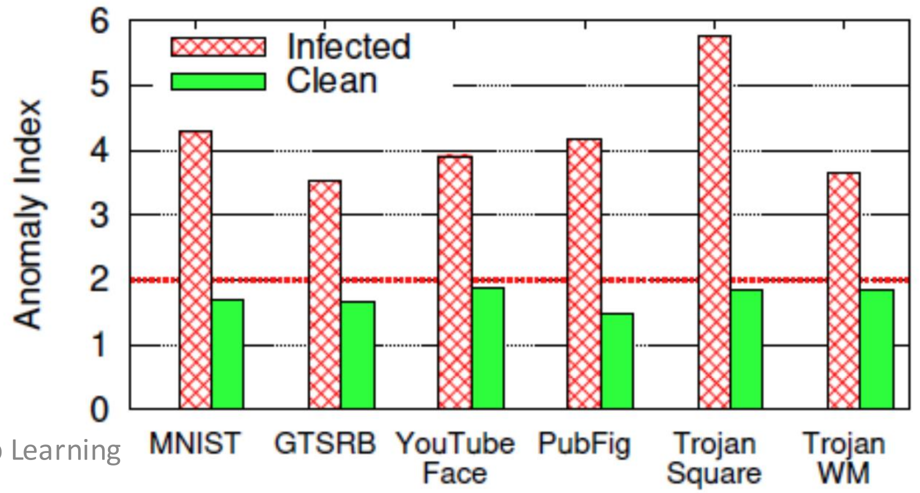
Trigger should be a small perturbation

- $A(x,\Delta)$: is the backdoored input with the trigger Δ
- $\ell\left(f\left(A(x,\Delta)\right), y^{\text{target}}\right)$: is the loss of the model for classifying backdoored image into class y^{target}.
- This may produce **multiple potential reversed engineered triggers**


Original Trigger (L1 norm = 16) | Reversed Trigger ($\boldsymbol{m} \cdot \boldsymbol{\Delta}$) (L1 norm = 12.00)
(a) MNIST
Original Trigger (L1 norm = 16) | Reversed Trigger ($\boldsymbol{m} \cdot \boldsymbol{\Delta}$) (L1 norm = 14.71)
(b) GTSRB
Original Trigger (L1 norm = 25) | Reversed Trigger ($\boldsymbol{m} \cdot \boldsymbol{\Delta}$) (L1 norm = 22.79)
(c) YouTube Face
Original Trigger (L1 norm = 576) | Reversed Trigger ($\boldsymbol{m} \cdot \boldsymbol{\Delta}$) (L1 norm = 171.11)
(d) PubFig

Defend Poisoning Attacks - NeuralCleanse

- **Defense Strategy:**
  1. Apply an optimization algorithm to calculate the “minimal” perturbation required to misclassify all samples from other labels to this target label.
  2. Run an **outlier detection algorithm** to detect if any trigger is significantly smaller than other triggers
     - Calculate Median Absolute Deviation (MAD), i.e., the absolute deviation between all other labels and the target label.
     - Calculate the anomaly index as the absolute deviation divided by the MAD.

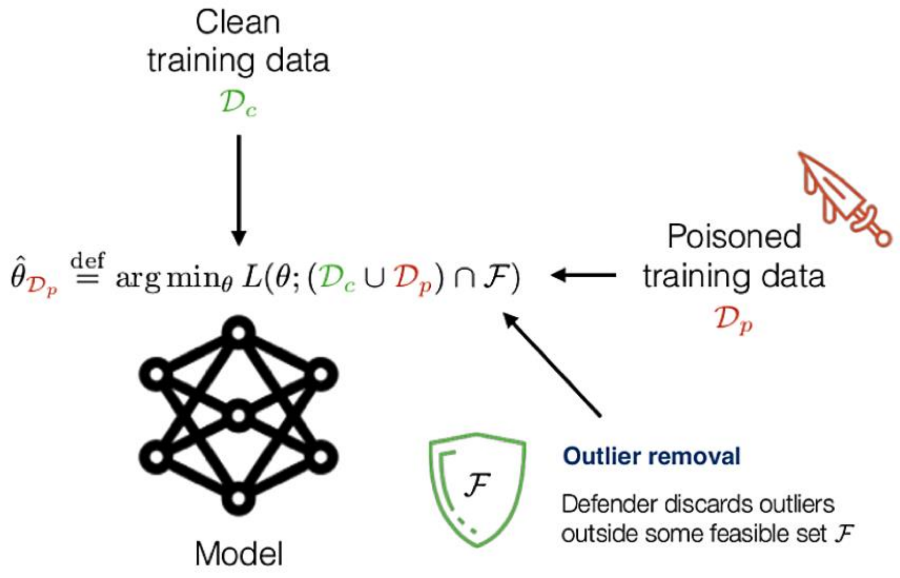


Defend Poisoning Attacks - NeuralCleanse

- **Defense Strategy:**
  1. Apply an optimization algorithm to calculate the “minimal” perturbation required to misclassify all samples from other labels to this target label.
  2. Run an **outlier detection algorithm** to detect if any trigger is significantly smaller than other triggers
  3. Mitigating backdoor attack
     - **Filter input samples** that are identified as adversarial inputs with a known trigger
     - Model patching algorithm based on **neuron pruning**: use the reversed trigger to identify activated neurons associated with the trigger and prune their values.
     - **Unlearning** the trigger. Fine-tune the model for only 1 epoch using poisoned images with correct labels (force the model to be more robust to the trigger).

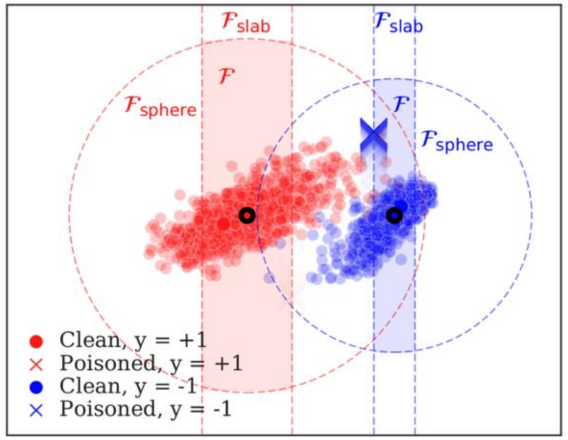
Defend Poisoning Attacks - Certified Defense (1)

- Defend against **data manipulation**
  - **Training data sanitization**: poisoning samples typically exhibit an outlying behavior w.r.t. the training data distribution → identify and remove poisoning samples before training (e.g., by outlier detection).



Clean training data $\mathcal{D}_c$
$\hat{\theta}_{\mathcal{D}_p} \overset{\text{def}}{=} \arg\min_\theta L(\theta; (\mathcal{D}_c \cup \mathcal{D}_p) \cap \mathcal{F})$
Poisoned training data $\mathcal{D}_p$
Model
$\mathcal{F}$
Outlier removal
Defender discards outliers outside some feasible set $\mathcal{F}$

Image credit: Steinhardt et al. "Certified Defenses for Data Poisoning Attacks."



Intuition: remove samples far from the data centroid.

$$\mathcal{F}_{\text{sphere}} \overset{\text{def}}{=} \{(x, y) : \|x - \mu_y\|_2 \leq r_y\},$$
$$\mathcal{F}_{\text{slab}} \overset{\text{def}}{=} \{(x, y) : |\langle x - \mu_y, \mu_y - \mu_{-y}\rangle| \leq s_y\}$$

Certificate. As long as $\mathcal{F}$ is not too small (e.g. outlier removal is not too aggressive) and the test loss is uniformly close to the clean train loss, U^* is an approximate upper bound on the worst-case attack.

Defend Poisoning Attacks – Certified Defense (2)

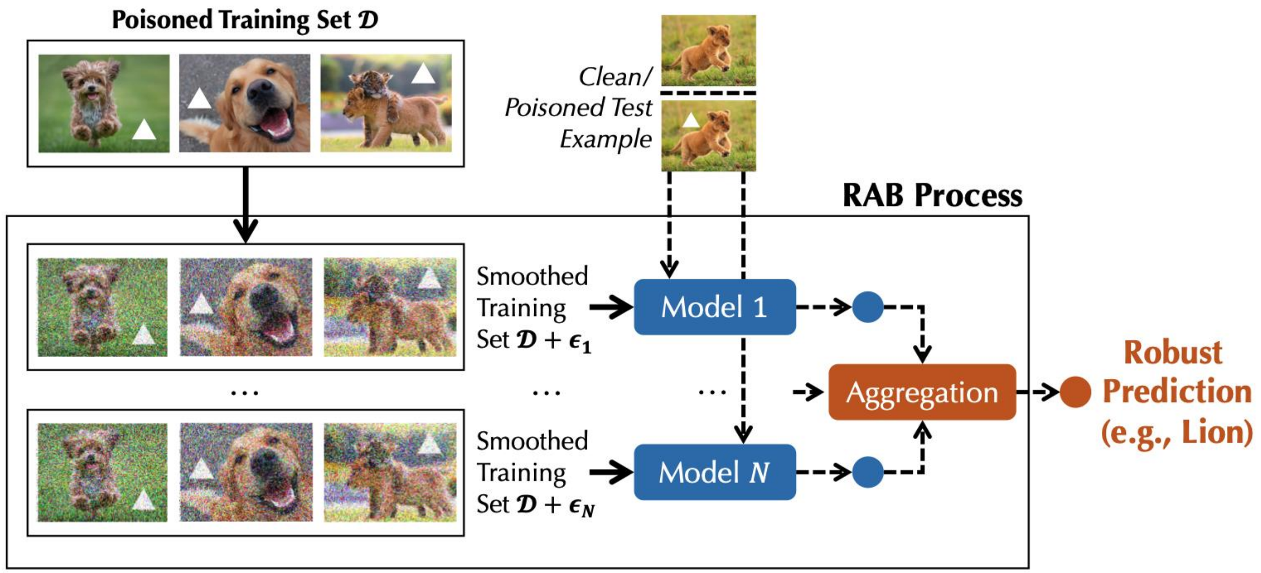
- Defend against **data manipulation**
  - **Robust training**: redesign the training paradigm to minimize the influence of poisoned samples.
    - Regularization
    - Data augmentation

E.g., data augmentation via noise

1. Generate N smoothed training datasets.
2. Train N different classifiers.
3. Aggregate the prediction over N classifiers

Certificate: If the norms of the backdoor patterns are sufficiently small, the above algorithm is guaranteed to make the correct prediction for poisoned data.

Intuition: randomizing the prediction and training process smoothens the classifier's prediction → less vulnerable to adversarial examples.

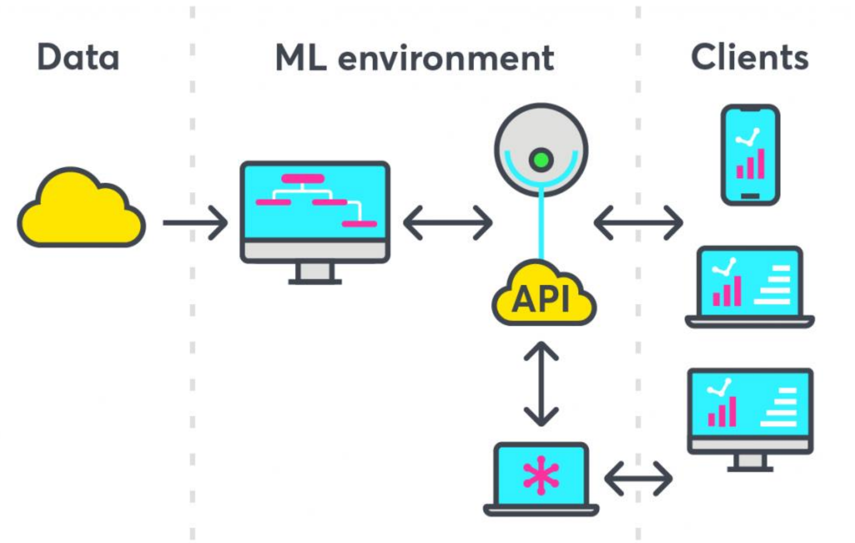


Poisoned Training Set $\mathcal{D}$
Clean/ Poisoned Test Example
RAB Process
Smoothed Training Set $\mathcal{D} + \epsilon_1$
Model 1
...
Smoothed Training Set $\mathcal{D} + \epsilon_N$
Model N
Aggregation
Robust Prediction (e.g., Lion)

Image credit: Weber et al. "RAB: Provable Robustness Against Backdoor Attacks."

Exploratory Attack

- ML-as-a-service offerings (e.g., cloud-based services from Amazon, Google, etc.) provide **black-box-only** services, via prediction API.
- **Exploratory attacks** do not modify the training samples, but try to gain information by **duplicating the functionality of the model**
- **ML-as-a-service (MLaaS):**

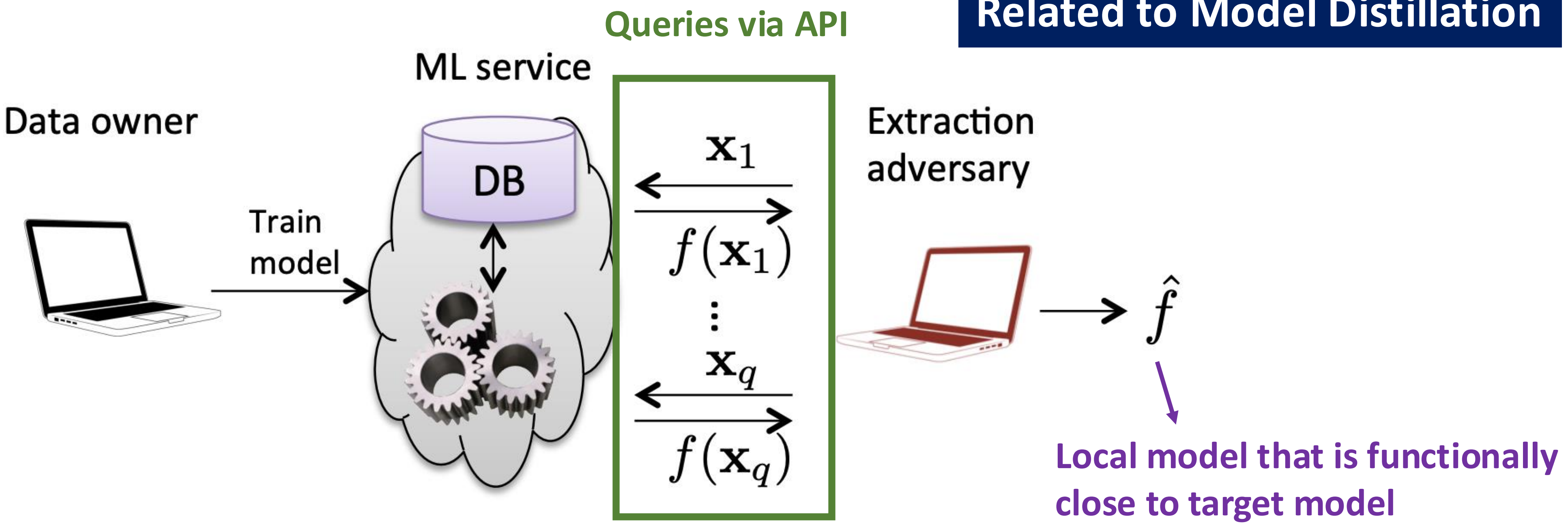


MLaaS pipeline

Model Extraction

- **Model Extraction** is a type of Exploratory Attack
- **Goal of Model Extraction**: learn **close approximation of black-box model f** using as few **queries** as possible.

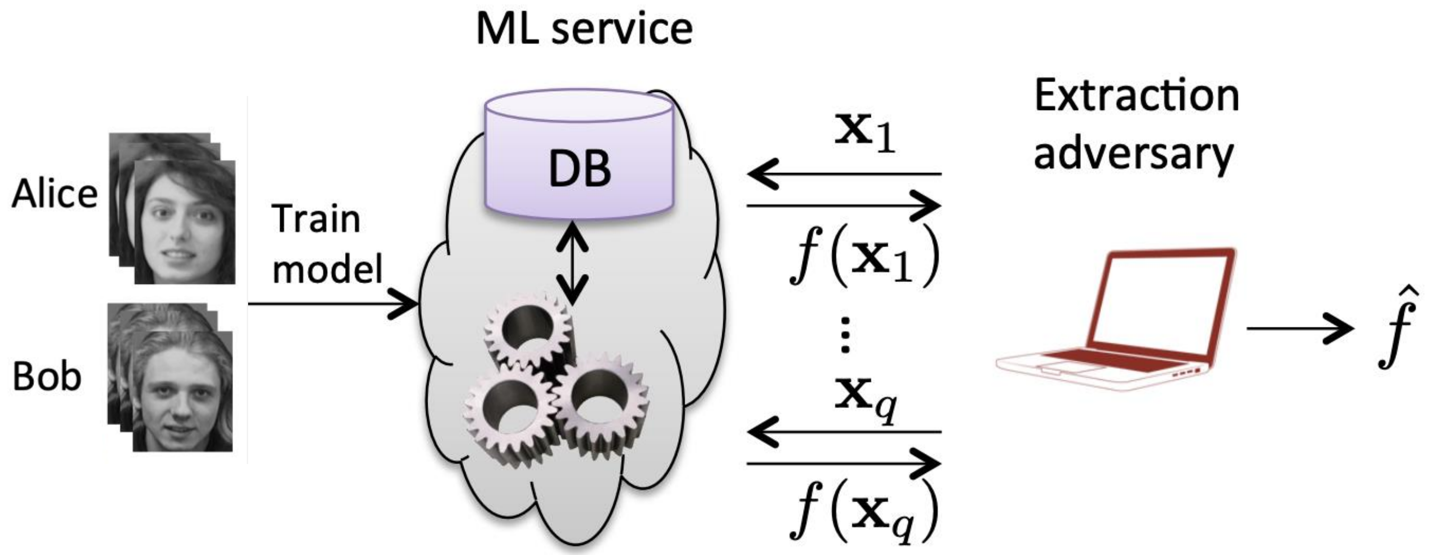
Related to Model Distillation



Target: $f(x) = \hat{f}(x)$ on ≥ 99.9% of inputs x

Example: Extraction of Logistic Regression

- Task: binary classification with logistic regression



$f(x) > 0.5 \Rightarrow$ Alice
$f(x) \leq 0.5 \Rightarrow$ Bob

Assume **x has n features**, then model has **$n + 1$ unknown parameters** (n for w and 1 for b)

$$f(x) = \frac{1}{1 + e^{-(w*x+b)}}$$

$$\ln\left(\frac{f(x)}{1 - f(x)}\right) = w * x + b$$

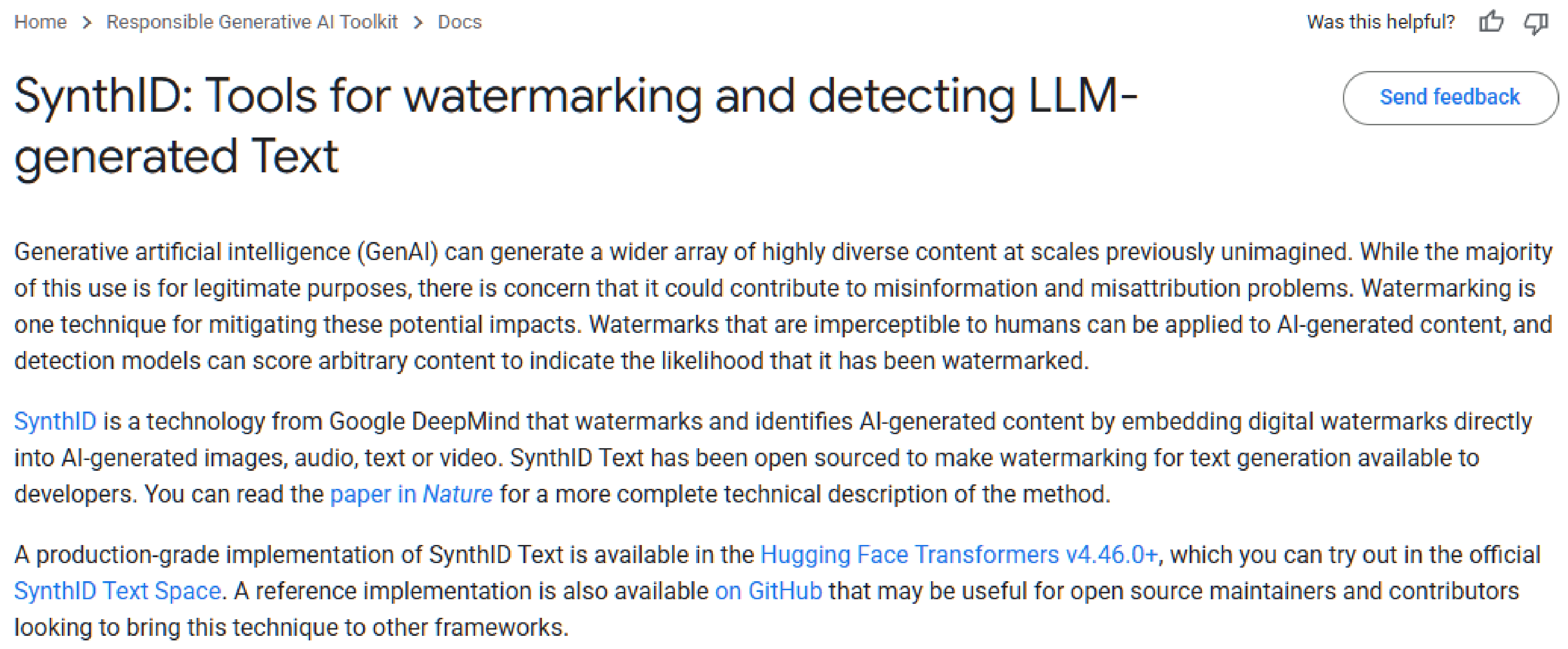
$$\downarrow$$

Linear equation with $n + 1$ unknows

Query $n + 1$ predictions with random samples $\Rightarrow$ **solve a linear system of $n + 1$ equations**

Poisoning for Security (1)

- Use case: model provider trains the model that can detect the hidden watermark
  - Identifies generated vs. natural images
  - Mitigate potential misinformation



SynthID: Tools for watermarking and detecting LLM-generated Text

Generative artificial intelligence (GenAI) can generate a wider array of highly diverse content at scales previously unimagined. While the majority of this use is for legitimate purposes, there is concern that it could contribute to misinformation and misattribution problems. Watermarking is one technique for mitigating these potential impacts. Watermarks that are imperceptible to humans can be applied to AI-generated content, and detection models can score arbitrary content to indicate the likelihood that it has been watermarked.

SynthID is a technology from Google DeepMind that watermarks and identifies AI-generated content by embedding digital watermarks directly into AI-generated images, audio, text or video. SynthID Text has been open sourced to make watermarking for text generation available to developers. You can read the paper in *Nature* for a more complete technical description of the method.

A production-grade implementation of SynthID Text is available in the Hugging Face Transformers v4.46.0+, which you can try out in the official SynthID Text Space. A reference implementation is also available on GitHub that may be useful for open source maintainers and contributors looking to bring this technique to other frameworks.

SynthID: Tools for watermarking and detecting LLM-generated Text | Responsible Generative AI Toolkit | Google AI for Developers

Poisoning for Security (2)

- Once you have a trained detector, you have a choice in if and how you expose it to your users, and the public more generally.
  - The **fully-private** option does not release or expose the detector in any way.
  - The **semi-private** option does not release the detector, but does expose it through an API.
  - The **public** option releases the detector for others to download and use.
- Nano Banana's Watermark : r/nanobanana
  - It can be subject to attacks
  - First identify the watermark
  - Design perturbations that fool the detector (in a back-box or white-box setting)

Summary

- By **adversary knowledge**: white-box attack and black-box attack
- By **modification phase:** poisoning attack (in training phase) ; evasion attack (in testing phase); exploratory attack (by observing the model by queries)
- Other attack examples:
  - **Ensemble-based attack** method to generate **transferable adversarial examples** to a black-box system.
  - **Circumvent obfuscated gradients** with Backward Pass Differentiable Approximation (BPDA), Expectation over Transformation (EOT) and Reparameterization.
- **Defense method:** Adversarial Training, Defensive Distillation, etc.