How to Steer Your Adversary: Targeted and Efficient Model Stealing Defenses with Gradient Redirection

Mantas Mazeika, Bo Li, David Forsyth

Model Stealing Attacks

- Adversaries can query public machine learning APIs and train copycat models
- This reduces the viability of the API business model by creating a dilemma:

Dilemma

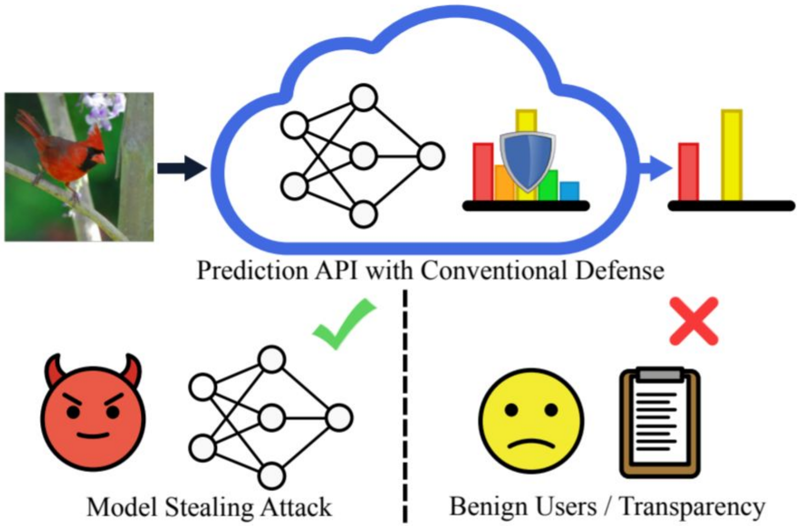
Customers want useful, interpretable predictions, but adversaries can use those same predictions to steal model capabilities.

Hypothetical Example

A legal dataset costs $2 million to initially collect. The API can be used by adversaries to generate an equivalent dataset for $10,000

Prior Defenses: Truncation

- A rudimentary defense: truncating posteriors to their top-K values
- Used by OpenAI, AI21, etc. Posteriors are truncated to 2% of their original size
- This harms benign users and reduces external transparency



Prior Defenses: Prediction Poisoning

- Instead of truncating information, poison the posterior with a small perturbation on the simplex (Orekondy et al., 2020)
- Design the perturbation to derail model stealing *gradient updates*
- Constrain the perturbation to be within epsilon of the true posterior

$$
\begin{aligned}
\max_{\tilde{\boldsymbol{y}}} \quad & \left\| \frac{\boldsymbol{G}^T \tilde{\boldsymbol{y}}}{||\boldsymbol{G}^T \tilde{\boldsymbol{y}}||_2} - \frac{\boldsymbol{G}^T \boldsymbol{y}}{||\boldsymbol{G}^T \boldsymbol{y}||_2} \right\|_2^2 && (= H(\tilde{\boldsymbol{y}})) \\
\text{where} \quad & \boldsymbol{G} = \nabla_{\boldsymbol{w}} \log F(\boldsymbol{x}; \boldsymbol{w}) && (\boldsymbol{G} \in \mathbb{R}^{K \times D}) \\
\text{s.t} \quad & \tilde{\boldsymbol{y}} \in \Delta^K && \text{(Simplex constraint)} \\
& \text{dist}(\boldsymbol{y}, \tilde{\boldsymbol{y}}) \leq \epsilon && \text{(Utility constraint)}
\end{aligned}
$$

Prior Defenses: Prediction Poisoning

- Instead of truncating information, poison the posterior with a small perturbation on the simplex (Orekondy et al., 2020)
- Design the perturbation to derail model stealing *gradient updates*
- Constrain the perturbation to be within epsilon of the true posterior

$$\max_{\tilde{\boldsymbol{y}}} \quad \Big\| \; \boldsymbol{G}^T \tilde{\boldsymbol{y}} \quad \ldots \quad \boldsymbol{G}^T \boldsymbol{y} \; \Big\|^2 \quad \ldots \quad (\ldots H(\tilde{\boldsymbol{y}}))$$

where $\ldots$ ${}^{K \times D})$

s.t $\ldots$ traint)

$$\text{dist}(\boldsymbol{y}, \tilde{\boldsymbol{y}}) \leq \epsilon \qquad \text{(Utility constraint)}$$

This method (MAD) requires one backward pass per class per query (expensive and slow!)

Gradient Redirection

- A similar approach in spirit, but markedly different in practice
- Maximize the inner product between the gradient update and a target z
- This is a linear program! How can we solve it efficiently?
- The problem resembles a knapsack problem, but with specific structure

$$
\begin{aligned}
\max_{\widetilde{y}} \quad & \langle G^{\mathsf{T}} \widetilde{y}, z \rangle \\
\text{s.t.} \quad & \mathbf{1}^{\mathsf{T}} \widetilde{y} = 1 \\
& \widetilde{y} \succeq 0 \\
& \|\widetilde{y} - y\|_1 \leq \epsilon
\end{aligned}
$$

Gradient Redirection

- Greedy algorithms can help
- We develop a provably correct, highly efficient algorithm for solving gradient redirection.

Theorem 4.1. *Given a gradient redirection problem* (G, z, y, ϵ) *as formulated in (2), Algorithm 1 outputs a globally optimal solution in* $\mathcal{O}(n \log(n))$ *time.*

- High-level sketch: Establish the greedy choice property and optimal substructure for a hierarchy of problems. The proof follows by induction.
- But we still have to compute n backwards passes, right?

Algorithm 1 Gradient Redirection

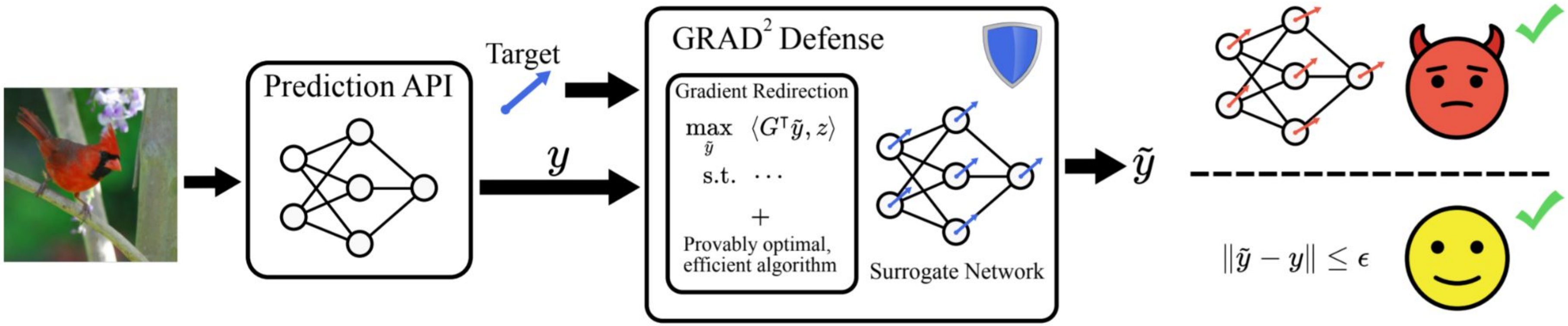
```
Input: G, z, y, ϵ
Output: ỹ
ỹ ← y
s ← argsort(Gz)
ỹ_{s_n} ← min(y_{s_n} + ϵ/2, 1)
λ ← 0
t ← 1
while t < n do
    ỹ_{s_t} ← max(y_{s_t} − (ϵ/2 − λ), 0)
    if y_{s_t} − (ϵ/2 − λ) > 0 then
        Return ỹ
    end if
    λ ← λ + y_{s_t}
    t ← t + 1
end while
```

Gradient Redirection: double backprop

- We circumvent the direct computation of G via double backprop
- Instead of n backward passes, we only need one double backprop (~3 additional forward passes)

GRAD^2 defense

- Our full defense incorporates gradient redirection at its core
- Surrogate networks are used, since the adversary's network is hidden
- The surrogate's *gradient update* can be steered in any target direction
- This transfers to the adversary!



Results

- For a given perturbation budget, we outperform numerous baselines
- In practice, we are substantially faster than MAD

	ImageNet-C10 → CIFAR-10						ImageNet-C100 → CIFAR-100						ImageNet-CUB200 → CUB200					
	Δ Clf. Err			ℓ_1 Distance			Δ Clf. Err			ℓ_1 Distance			Δ Clf. Err			ℓ_1 Distance		
Method	1	2	5	0.1	0.2	0.5	1	2	5	0.1	0.2	0.5	1	2	5	0.1	0.2	0.5
Random	9.8	10.3	10.6	9.0	8.7	9.3	38.5	38.6	39.8	36.2	36.5	38.5	48.5	51.4	56.0	41.3	42.3	50.7
Reverse Sigmoid	-	-	-	9.0	9.1	9.3	-	-	-	36.3	36.8	38.0	-	-	-	41.2	42.6	45.9
Adaptive Mis.	10.4	11.9	16.3	9.0	9.6	12.1	38.2	40.6	46.6	36.4	37.4	41.8	53.8	**58.6**	**66.8**	**42.8**	**45.6**	53.8
MAD	12.6	16.4	22.6	8.7	8.7	9.5	43.0	46.8	49.2	35.9	36.9	42.6	49.6	52.3	56.0	41.7	42.6	51.7
GRAD^2 (Ours)	**16.4**	**21.5**	**23.4**	**9.5**	**10.1**	**15.5**	**43.4**	**47.6**	**53.0**	**36.5**	**37.7**	**44.1**	**54.1**	56.4	60.7	41.8	44.6	**55.6**

Method	CIFAR-10	CIFAR-100	CUB200
MAD	0.15	1.21	2.66
GRAD^2	0.04	0.28	0.42

Thank you