Identifying a Training-Set Attack’s Target
Using Renormalized Influence Estimation

Zayd Hammoudeh  Daniel Lowd
University of Oregon
Eugene, Oregon, USA
{zayd,lowd}@cs.uoregon.edu

ABSTRACT
Targeted training-set attacks inject malicious instances into the training set to cause a trained model to mislabel one or more specific test instances. This work proposes the task of target identification, which determines whether a specific test instance is the target of a training-set attack. Target identification can be combined with adversarial-instance identification to find (and remove) the attack instances, mitigating the attack with minimal impact on other predictions. Rather than focusing on a single attack method or data modality, we build on influence estimation, which quantifies each training instance’s contribution to a model’s prediction. We show that existing influence estimators’ poor practical performance often derives from their over-reliance on training instances and iterations with large losses. Our renormalized influence estimators fix this weakness; they far outperform the original estimators at identifying influential groups of training examples in both adversarial and non-adversarial settings, even finding up to 100% of adversarial training instances with no clean-data false positives. Target identification then simplifies to detecting test instances with anomalous influence values. We demonstrate our method’s effectiveness on backdoor and poisoning attacks across various data domains, including text, vision, and speech, as well as against a gray-box, adaptive attacker that specifically optimizes the adversarial instances to evade our method. Our source code is available at https://github.com/ZaydH/target_identification.

CCS CONCEPTS
• Security and privacy; Theory of computation → Adversarial learning.

KEYWORDS
Target Identification; Backdoor Attack; Data Poisoning; Influence Estimation; TracIn; GAS; Influence Functions; Representor Point

This is an extended version which includes additional details that did not fit in the peer-reviewed version. Not for redistribution. The definitive, peer-reviewed version is published in the proceedings of CCS’22 [27].
This paper identifies a weakness common to many influence estimators [12, 35, 57, 81]: they induce a low-loss penalty that implicitly ranks confidently-predicted training instances as uninformative. As a result, existing influence estimators can systematically overlook (groups of) highly influential, low-loss instances. We remedy this via a simple renormalization that removes the low-loss penalty. Our new renormalized influence estimators consistently outperform the originals in both adversarial and non-adversarial settings. The most effective of these, gradient aggregated similarity (GAS), often detects 100% of malicious training instances with no clean-data false positives.

Our framework for identifying targets of training-set attacks, FIT, compares the distribution of influence values across test instances checking for anomalies. More concretely, FIT marks as potential targets those test instances with an unusual number of highly influential instances as explained above. Next, FIT mitigates the attack’s effect by removing exceptionally influential training instances associated with the target(s). Since mitigation considers only targets, training instance outliers that are “helpful” to non-targets are unaffected. This target-driven mitigation has a positive or neutral effect on clean data yet is highly effective on adversarial data where finding even a single target suffices to disable the attack on almost all other targets.

By relying on the concepts of influence estimation and not the properties of a particular attack or domain, GAS and FIT are attack agnostic [62]. They can apply equally well to different attack types, including data poisoning attacks, which target unperturbed test data, and backdoor attacks on test instances activating a specific trigger. Our approach works across data domains from CNN image classifiers to speech recognition to even text transformers.

In addition to learning more about the attack and attacker, target identification enables targeted mitigation. Certified training-set defenses [28, 33, 42, 68, 77, 78] (which do not identify targets) implement countermeasures (e.g., smoothing [74]) that affect predictions on all instances – not just the very few targets. These methods can substantially degrade performance, in some cases causing up to 10x more errors on clean data [20, 28]. A strength of deep neural networks is that they can “memorize” instances to learn rare cases – not just the very few targets. These methods

2 PROBLEM FORMULATION

Notation $x \in X$ denotes a feature vector and $y \in Y$ a label. Training set, $D_{tr} = \{(z_i)_{n_{tr}}\}$, consists of $n$ training example tuples $z_i := (x_i, y_i)$. Consider model $f : X \to A$ parameterized by $\theta$, where $a := f(x; \theta)$ denotes the model’s output (pre-softmax) activations. $\theta_0$ denotes $f$’s initial parameters, which may be randomly set and/or pre-trained.

For loss function $l : \mathcal{A} \times Y \to \mathbb{R}_{\geq 0}$, denote $z$’s empirical risk given $\theta$ as $L(z; \theta) := l(f(x; \theta), y)$. Consider any iterative, first-order optimization algorithm (e.g., gradient descent, Adam [34]). At each iteration $t \in \{1, \ldots, T\}$, the optimizer updates parameters $\theta_t$ from loss $l$, previous parameters $\theta_{t-1}$, and batch $B_t \subseteq D_{tr}$ of size $b$. Gradients are denoted $g_i^{(t)} := \nabla_{\theta} L(z_i; \theta_t)$; the gradient’s superscript “$(t)$” is dropped when the iteration is clear from context.

Let $\hat{z}_{te} := (x_{te}, y_{te})$ be any a priori unknown test instance, where $y_{te}$ is the final model’s predicted label for $x_{te}$. Observe that $\hat{y}_{te}$ may not be $x_{te}$’s true label. Notation “$\hat{z}$” (e.g., $\hat{z}, \hat{g}$) denotes that the final predicted label $\hat{y} = f(x; \theta_T)$ is used in place of $x$’s true label $y$.

Threat Model

The attacker crafts an adversarial set of perturbed instances, $D_{adv} \subset D_{tr}$. Denote the clean training set $D_{cl} := D_{tr} \setminus D_{adv}$. We only consider successful attacks, as defined below.

Attacker Objective & Knowledge Let $X_{target} := \{x_j\}_{j=1}^{n_{target}}$ be a set of target feature vectors with shared true label $y_{target} \in Y$. The attacker crafts $D_{adv}$ to induce the model to mislabel all of $X_{target}$ as adversarial label $y_{adv}$. $Z_{target} := \{(x_j, y_{adv}) : x_j \in X_{target}\}$ denotes the target set and $z_{target} := (x_{target}, y_{adv})$ an arbitrary target instance. To avoid detection, the attack model’s clean-data performance should be (essentially) unchanged. Data poisoning attacks only perturb adversarial set $D_{adv}$. Target feature vectors are unperturbed/benign [7, 32, 50, 73]. Clean-label poisoning leaves labels unchanged when crafting $D_{adv}$ from seed instances [84]. Backdoor attacks perturb the features of both $D_{adv}$ and $X_{target}$ – often with the same adversarial trigger (e.g., change a specific pixel to maximum value). Generally, these triggers can be inserted into any test example targeted by the adversary, making most backdoor attacks multi-target ($|Z_{target}| > 1$) [24, 45, 70, 78]. $D_{adv}$’s labels may also be changed.

To ensure the strongest adversary, the attacker knows any pre-trained initial parameters. Where applicable, the attacker also knows the training hyperparameters and clean dataset $D_{tr}$. Like previous work [73, 78, 84], the attacker does not know the training procedure’s random seed, meaning the attack must be robust to randomness in batch ordering or parameter initialization.

Defender Objective & Knowledge Let $Z_{te}$ denote the set of test instances the defender is concerned enough about to analyze
as potential targets. Our goals are to (1) identify any attack targets in $\mathcal{Z}_{\mathrm{tr}}$, and (2) mitigate the attack by removing the adversarial instances $\mathcal{D}_{\mathrm{adv}}$ associated with those target(s). No assumptions are made about the modality/domain (e.g., text, vision) or adversarial perturbation. We do not assume access to clean validation data.

3 RELATED WORK

We mitigate training-set attacks by building upon influence estimation to identify the target(s) and adversarial set. This section first reviews existing defenses against training-set attacks and then formalizes training-set influence as defined in previous work.

3.1 Defenses Against Training-Set Attacks

Certifiably-robust defenses [28, 33, 42, 68, 74, 77, 78] provide guaranteed protection against specified training-set attacks under specific assumptions. Empirical defenses [17, 21, 55, 71, 85, 86] derive from understandings and observations about the underlying mechanisms training-set attacks exploit to change a network’s predictions. In practice, empirical defenses generally significantly outperform certified approaches with fewer harmful side effects – albeit without a guarantee [44]. These two defense categories are complementary and can be deployed together for better performance. Since our defense is largely empirical, we focus on that defense category below.

We are not aware of any existing defense – certified or empirical – that provides target identification. The most closely related task is determining whether a model is infected with a backdoor, which ignores poisoning and other training-set attacks [67, 80]. Such methods make different assumptions than this work. For instance, some assume access to known-clean data [17, 21, 46, 75, 86] but may not have access to the training set. Many also make assumptions about the type of attack or the training methodology [67].

Another task similar to target identification is adversarial trigger synthesis, which attempts to reconstruct any backdoor attack pattern(s) a model learned [21, 71, 72, 86]. These methods mitigate attacks by adding identified triggers to known-clean data so that retraining will cause catastrophic forgetting of the trigger.

Data-sanitization defenses mitigate attacks by removing adversarial set $\mathcal{D}_{\mathrm{adv}}$ from $\mathcal{D}_{\mathrm{tr}}$. Existing data-sanitization defenses have shown promise [10, 55, 70], but they all share a common pitfall concerning setting the data-removal threshold [36, 44]. If this threshold is set too low, significant clean-data removal degrades overall clean-data performance. A threshold set too high results in insufficient adversarial training data removal and the attack remaining successful. Additional information (e.g., target identification) enables targeted tuning of this removal threshold.

Most existing certified and empirical defenses are not attack agnostic and assume specific data modalities (e.g., only vision [21, 71, 72, 86]), model architectures (e.g., CNNs [37]), optimizers [29], or training paradigms [67]. Attack agnosticism is more challenging and more practically useful. We achieve agnosticism by building upon existing methods that are general – namely influence estimation, which is formalized next.

3.2 Training-Set Influence Estimation

In every successful attack, the inserted training instance changes a model’s prediction for specific input(s). If the attacker can only add a limited number of instances (e.g., 1% of $\mathcal{D}_{\mathrm{tr}}$), these inserted instances must be highly influential to achieve the attacker’s objective.

Influence estimation’s goal is to determine which training instances are most responsible for a model’s prediction for a particular input. Influence is often viewed as a counterfactual: which instance (or group of instances) induces the biggest change when removed from the training data? While there are multiple definitions of influence, as detailed below, influence estimation methods can be broadly viewed as quantifying the relative responsibility of each training instance $z_i \in \mathcal{D}_{\mathrm{tr}}$ on some test prediction $f(x_{\text{te}}, \theta_T)$.

Static influence estimators consider only the final model parameters $\theta_T$. For example, Koh and Liang’s [35] seminal work defines influence, $I_{\text{IF}}(z_i, \hat{z}_{\text{te}})$, as the change in risk $L(\hat{z}_{\text{te}}, \theta_T)$ if $z_i \notin \mathcal{D}_{\mathrm{te}}$, i.e., the leave-one-out (LOO) change in test loss [14]. By assuming strict convexity and stationarity, Koh and Liang’s influence functions estimator approximates the LOO influence as

$$I_{\text{IF}}(z_i, \hat{z}_{\text{te}}) = \frac{1}{n} \sum_{\tau \neq i} \nabla_{\theta_T} L(\hat{z}_{\text{te}}; \theta_T) \cdot \nabla_{\theta_T} L(z_i; \theta_T),$$

where $H_{\theta_T}$ is the inverse of risk Hessian $H_{\theta_T} \approx \frac{1}{n} \sum_{\tau \neq i} \nabla_{\theta_T} L(z_i; \theta_T)$.

Yeh et al. [81]’s representor point static influence estimator exclusively considers the model’s final, linear classification layer. All other model parameters are treated as a fixed feature extractor. Given final parameters $\theta_T$, let $f_i$ denote $x_i$’s penultimate feature representation (i.e., the input to the linear classification layer). Then the representor point influence of $z_i \in \mathcal{D}_{\mathrm{tr}}$ on $\hat{z}_{\text{te}}$ is

$$I_{\text{IF}}(z_i, \hat{z}_{\text{te}}) = \frac{1}{n} \left( \frac{\partial L(z_i; \theta_T)}{\partial a_{y_i}} \right) \left( f_i, f_{y_i} \right),$$

where $\lambda > 0$ is the weight decay $(L_2)$ regularizer and $(\cdot, \cdot)$ denotes vector dot product. Recall that $a$ is the output of the model’s linear classification layer, specifically here $a = f(x; \theta_T)$. Scalar $\frac{\partial L(x; \theta_T)}{\partial a_{y_i}}$ is then the partial derivative of risk $L$ w.r.t. a $y_i$’th dimension.

Dynamic influence estimators measure influence based on how losses change during training. More formally, influence is quantified according to how batch $B_i$, $\ldots$, $B_T$ affect model parameters $\theta_0$, $\ldots$, $\theta_T$ and by consequence risks $L(\cdot; \theta_0), \ldots, L(\cdot; \theta_T)$. For example, Pruthi et al. [57] Track estimates influence by “tracing” gradient descent – aggregating changes in $\hat{z}_{\text{te}}$’s test loss each time training instance $z_i$’s gradient updates parameters $\theta_T$. For stochastic gradient descent (batch size $b = 1$), $z_i$’s Track influence on $\hat{z}_{\text{te}}$ is

$$F_{\text{Track}}(z_i, \hat{z}_{\text{te}}) := \sum_{t=1}^{T} \mathbb{1}_{z_i \in B_t} \left( L(\hat{z}_{\text{te}}; \theta_{t-1}) - L(\hat{z}_{\text{te}}; \theta_{t}) \right),$$

where $\mathbb{1}_i$ is the indicator function s.t. $\mathbb{1}_i = 1$ if predicate $i$ is true and 0 otherwise. Pruthi et al. approximate Eq. (3) as

$$F_{\text{Track}}(z_i, \hat{z}_{\text{te}}) \approx \sum_{z_i \in B_t} \frac{\eta_t}{b} \left( \nabla_{\theta} L(z_i; \theta_{t-1}) - \nabla_{\theta} L(\hat{z}_{\text{te}}; \theta_{t-1}) \right),$$

where $\eta_t$ is iteration $t$’s learning rate.

Alg. 1 details the minimal changes made to model training to support Track where $\mathcal{T} \subseteq \{1, \ldots, T\}$ is a preselected training iteration subset and $\mathcal{P} := \{ (\eta_t, \theta_{t-1}) : t \in \mathcal{T} \}$ contains the serialized training parameters. Alg. 5 outlines Track’s influence estimation procedure for a priori unknown test instance $\hat{z}_{\text{te}}$. Influence vector $v (|v| = n)$ contains the Track influence estimates for each $z_i \in \mathcal{D}_{\text{tr}}$. 

\footnote{Generally, there are far fewer potential targets ($\mathcal{Z}_{\text{tr}}$) than possible test examples.}

\footnote{Due to space, Alg. 1 appears in the supplement.}
Algorithm 1 TracIn, TracInCP, & GAS training phase

Input: Training set $D_h$, iteration subset $T$, iteration count $T$, learning rates $\eta_1, \ldots, \eta_T$, and initial parameters $\theta_0$

Output: Training parameters $P$

1: \( P \leftarrow \emptyset \)
2: for \( t \leftarrow 1 \) to \( T \) do
3: \( \quad \) if \( t \in T \) then
4: \( \quad \quad P \leftarrow P \cup \{(\eta_t, \theta_{t-1})\} \)
5: \( B_t \sim D_h \)
6: \( \quad \theta_t \leftarrow \text{Update}(\eta_t, \theta_{t-1}, B_t) \)
7: \( \text{return } P \)

In practice, \(|T| \ll T, \) and \( T \) is evenly-spaced in \( \{1, \ldots, T\} \), meaning TracIn effectively treats multiple batches like a single model update.

Pruthi et al. also propose TracIn Checkpoint (TracInCP) – a more heuristic version of TracIn that considers all training examples at each checkpoint in \( T \) – not just those instances in the intervening batches (see Alg. 2). Formally,

\[
I_{\text{TracInCP}}(z_t, \hat{z}_{te}) = \sum_{i \in T} \eta_i \left( \nabla_{\theta} L(z_i; \theta_{i-1}) \cdot \nabla_{\theta} L(\hat{z}_i; \theta_{i-1}) \right). \tag{5}
\]

TracInCP is more computationally expensive than TracIn – with the slowdown linear w.r.t. the number of checkpoints per epoch.

A major advantage of TracIn and TracInCP over other estimators (e.g., influence functions) is that their only hyperparameter is iteration set \( T \), which we tuned based only on compute availability.

\footnote{Algorithm 2 combines two different methods TracInCP as well as GAS – our renormalized version of TracInCP discussed in Sec. 4.3.}

4 WHY INFLUENCE ESTIMATION OFTEN FAILS AND HOW TO FIX IT

Before addressing target identification, we first consider the related task of adversarial-instance identification. In the simplest case, if the attack’s target is known, then the malicious instances should be among the most influential instances for that target instance. In other words, adversarial-instance identification reduces to influence estimation. However, Sec. 3.2’s influence estimators share a common weakness that makes them poorly suited for this task: they all consistently rank confidently-predicted training instances as uninfluential. We illustrate this behavior below using a toy experiment. We then explain this weakness’s cause and propose a simple fix that addresses this limitation on adversarial and non-adversarial data, for all preceding estimators. Our fix is needed to successfully identify adversarial set $D_{adv}$ and, as detailed in Sec. 5, attack targets.

4.1 A Simple Experiment

Consider binary classification where clean set $D_\alpha$ is all frog and airplane training images in CIFAR10 ($|D_\alpha| = 10,000$). To simulate a naive backdoor attack, adversarial set $D_{adv}$ is 150 randomly selected MNIST test images labeled as airplane.

Clean data’s overall influence can be estimated indirectly by training only on $D_\alpha$ and observing the target set’s misclassification rate [18]. This experiment used class pair frog and airplane because amongst the Class CIFAR10 class pairs, frog vs. airplane’s average MNIST test misclassification rate was closest to random (47.5% vs. 50% ideal). In contrast, when training on $D_h = D_{adv} \cup D_\alpha$, MNIST test instances were always classified as airplane, meaning $D_{adv}$ is overwhelmingly influential on MNIST predictions. MNIST
The gradient norm ratio closely tracks both training sets’ loss values. Both during and at the end of training, $D_{\text{adv}}$’s median loss is significantly smaller than many instances in $D_{\text{cl}}$—often by several orders of magnitude.

The Low-Loss Penalty. Observe that all influence methods in Sec. 3.2’s scale their influence estimates by $\frac{\partial \ell(\mathbf{a})}{\partial \mathbf{a}}$ either directly (representer point (2)) or indirectly via the chain rule (influence functions (1), TracIn (4), and TracInCP (5)) as

$$\nabla_{\theta} \ell(z; \theta) \equiv \frac{\partial f(x, y)}{\partial \theta} = \frac{\partial f(a, y)}{\partial a} \frac{\partial a}{\partial \theta}$$

Therefore, gradient-based influence estimators implicitly penalizes all training instances $t$ with low training loss, including $D_{\text{adv}}$ (MNIST 0) in our toy experiment above.

Theorem 4.1 summarizes this relationship when there is a single output activation ($|a| = 1$), e.g., binary classification and univariate regression. In short, when Theorem 4.1’s conditions are met, loss induces a perfect ordering on the corresponding norm.

**Theorem 4.1.** Let loss function $\ell: \mathbb{R} \rightarrow \mathbb{R}_{\geq 0}$ be twice-differentiable and strictly convex as well as either even\(^5\) or monotonically decreasing. Then, it holds that

$$\ell(a) < \ell(a') \implies \|\nabla_{\theta} \ell(a)\|_2 < \|\nabla_{\theta} \ell(a')\|_2.$$  

Loss functions satisfying Theorem 4.1’s conditions include binary cross-entropy (i.e., logistic) and quadratic losses. Theorem 4.1 generally applies to multiclass losses, but there are cases where the ordering is not perfect. Although Theorem 4.1 primarily relates to training instance gradients and losses, the theorem applies to test examples as well since dynamic estimators also apply a low-loss penalty to any iteration where test instance $z_{\text{te}}$ has low loss.

The preceding should not be interpreted to imply that large gradient magnitudes are unimportant. Quite the opposite, large gradients have large influences on the model. However, the approximations necessary to make influence estimation tractable go too far by often focusing almost exclusively on training loss—and by extension gradient magnitude—leading these estimators to systematically overlook training instances with smaller gradients. This overemphasis of instances with large losses and gradient magnitudes can also be viewed as a bias towards instances that are globally influential—offering many examples’ predictions—over those that are locally influential—mainly affecting a small number of targets [4].

**Static Influence & the Low Loss Penalty:** Fig. 1’s static estimators (representer point & influence functions) significantly underperformed dynamic estimators (TracIn & TracInCP) by up to an order of magnitude. Static estimators only consider final model parameters $\theta_F$, meaning they may only see the low-loss case. In contrast, dynamic estimators consider all of training, in particular iterations where $D_{\text{adv}}$’s loss exceeds that of $D_{\text{cl}}$. This allows dynamic estimators to outperform static methods, albeit still poorly.

**Training Randomness & the Low Loss Penalty:** TracInCP significantly outperformed TracIn in Fig. 1 despite the TracIn being more theoretically sound. As intuition why, imagine the training set contains two identical copies of some instance. In expectation, these duplicates have equivalent influence on any test instance. However, TracIn assigns identical training examples different influence

---

\(^5\)“Even” denotes that the function satisfies $\forall a \; \ell(a) = \ell(-a)$.

---

**Figure 2: CIFAR10 & MNIST Intra-training Loss Tracking:** $D_{\text{adv}}$’s (—) & $D_{\text{cl}}$’s (—) median cross-entropy losses ($\ell$) at each training checkpoint for binary classification – frog vs. airplane & MNIST 0. The shaded regions correspond to each training set loss’s interquartile range. MNIST’s training losses are generally several orders of magnitude smaller than CIFAR10’s losses. Gradient norm ratio (—) shows the tight coupling of loss & training gradient magnitude.

---

**4.2 Why Influence Estimation Performs Poorly**

Intra-training dynamics illuminate the primary cause of influence estimation’s poor performance in our toy experiment. Fig. 2 visualizes the median training loss of $D_{\text{adv}}$ and $D_{\text{cl}}$ at each training checkpoint. Also shown is the gradient norm ratio, which compares the median gradient magnitude of the adversarial and clean sets at each iteration, or formally

$$\text{GNR}_t \equiv \frac{\text{med}\{L(z; \theta) : z \in D_{\text{adv}}\}}{\text{med}\{L(z; \theta) : z \in D_{\text{cl}}\}$$

\(^\ast\)See supplemental Section C for the complete experimental setup details.

---

**Figure 2**
estimates based on their batch assignments; this difference can potentially be very large depending on training dynamics.

Fig. 2 exhibits this behavior where training loss fluctuates considerably intra-epoch. For example, $D_{adv}$’s median loss varies by seven orders of magnitude across the third epoch. TracIn’s low-loss penalty attributes much more influence to $D_{adv}$ instances early in that epoch compared to those later despite all MNIST instances having similar influence. By considering all examples at each checkpoint, TracInCP removes batch randomization’s direct effect on influence estimation,\(^6\) meaning TracInCP simulates influence expectation without needing to train and analyze multiple models.

### 4.3 Renormalizing Influence Estimation

Our CIFAR10 & MNIST joint classification experiment above demonstrates that a training example having low loss does not imply that it and related instances are uninfluential. Most importantly in the context of adversarial attacks, highly-related groups of (adversarial) training instances may collectively cause those group members’ to have very low training losses – so-called group effects. Generally, targeted attacks succeed by leveraging the group effect of adversarial set $D_{adv}$ on the target(s). We address these group effects via renormalization, which is defined below.

**Definition 4.2.** For influence estimator $I$, the **renormalized influence**, $\tilde{I}$, replaces each gradient $g$ in $I$ by its corresponding unit vector $\frac{g}{||g||}$.

We refer to this computation as renormalization since rescaling gradients removes the low-loss penalty. Renormalization places all training instances on equal footing and ensures that gradient and/or feature similarity is prioritized – not loss.

Renormalization is related to the relative influence (RelatIF) method introduced by Barshan et al. [4], since both methods use a function of the gradient to downweight training instances with high losses. However, RelatIF only applies to influence functions and requires computing expensive Hessian-vector products, while renormalization is more efficient and can be applied to many influence estimators, as we show below. See suppl. Section F.7 for additional discussion of alternative renormalization schemes.

Renormalized versions of Section 3.2’s static influence estimators are below. Renormalized **influence functions** in Eq. (9) does not include target gradient norm $||g_t||$ since it is a constant factor. For simplicity, Eq. (10)’s renormalized **representer point** uses signum function $\text{sgn}(\cdot)$ since for any scalar $u \neq 0$, $\text{sgn}(u) = \frac{u}{||u||}$, i.e., signum is equivalent to normalizing by magnitude.

\[
\tilde{I}_{IF}(z_i, \hat{y}_e) := \frac{1}{n} \sum_{j \in z_i} \frac{\text{sgn}(\nabla_{\theta} \mathcal{L}(z_i; \theta_{t-1}))}{\nabla_{\theta} \mathcal{L}(z_i; \theta_{t-1})} (f_i, f_{te})
\]

Renormalized versions of Section 3.2’s dynamic influence estimators appear below. Going forward, we refer to renormalized TracInCP (Eq. (12)) as gradient aggregated similarity. GAS, since it is essentially the weighted, gradient cosine similarity averaged across all of training. GAS’s procedure is detailed in Algorithm 2.\(^7\)

\[
\tilde{I}_{TracIn}(z_i, \hat{y}_e) := \frac{1}{n} \sum_{j \in z_i} \frac{\text{sgn}(\nabla_{\theta} \mathcal{L}(z_i; \theta_{t-1}))}{\nabla_{\theta} \mathcal{L}(z_i; \theta_{t-1})} (f_i, f_{te})
\]

\[
\tilde{I}_{GAS}(z_i, \hat{y}_e) := \frac{1}{n} \sum_{j \in z_i} \frac{\text{sgn}(\nabla_{\theta} \mathcal{L}(z_i; \theta_{t-1}))}{\nabla_{\theta} \mathcal{L}(z_i; \theta_{t-1})} (f_i, f_{te})
\]

Unlike static estimators, rescaling dynamic influence by target gradient norm $||g_t||$ is quite important as mentioned earlier. Intuitively, $||g_t||$ tends to be largest in two cases: (1) early in training due to initial parameter randomness and (2) when iteration $t$’s predicted label conflicts with final label $\hat{y}_e$. Both cases are consistent with the features most responsible for predicting $\hat{y}_e$ not yet dominating. Therefore, rescaling dynamic influence by $||g_t||$ implicitly upweights iterations where $\hat{y}_e$ is predicted confidently. It also inhibits any single checkpoint dominating the estimate.

**Applying Renormalization to CIFAR10 & MNIST Joint Classification:** Figure 1’s lower half demonstrates renormalization’s significant performance advantage over standard influence estimation – with the improvement in AUPRC as large as 25%. In particular, our renormalized estimators’ top-5 highest-ranked instances were all consistently from MNIST, unlike any of the standard influence estimators. Overall, GAS (renormalized TracInCP) was the top performer – even outperforming our other renormalized estimators by a wide margin.

### 4.4 Renormalization & More Advanced Attacks

Section 4.2 illustrates why influence performs poorly under a naive backdoor-style attack where the adversary does not optimize the adversarial set. Those concepts also generalize to more sophisticated attacks. For example, recent work shows that deep networks often predict the adversarial set with especially high confidence (i.e., low loss) due to shortcut learning – even on advanced attacks [23, 82]. Those findings reinforce the need for renormalization. This can be viewed through the lens of simplicity bias where neural networks tend to confidently learn simple features (shortcuts) – regardless of whether those features actually generalize [65].

---

\(^6\)Batch randomization still indirectly affects TracInCP and GAS (Sec. 4.3) through the model parameters. This effect could be mitigated by training multiple models and averaging the (renormalized) influence, but that is beyond the scope of this work.

\(^7\)As shown in Algorithm 2, TracInCP’s procedure (Line 7) is identical to GAS (Line 9) other than influence renormalization.
Figure 3: Layerwise Decomposition of an Attack Target’s Intra-Training Gradient Magnitude. One-pixel & blend backdoor adversarial triggers (dashed & solid lines respectively) trained separately on CIFAR10 binary classification ($y_{\text{targ}} = \text{airplane}$ & $y_{\text{adv}} = \text{bird}$) using ResNet9. The network’s first convolutional (Conv1) and final linear layers are a small fraction of the parameters (0.03% & 0.01% resp.) but constitute most of the target’s gradient magnitude ($\|\nabla_{\theta_t} L\|$) with the dominant layer attack dependent. Results are averaged over 20 trials.

Dynamic estimators — both TracIn and GAS — outperform static ones for Sec. 4.1’s naive attack. The same can be expected for sophisticated attacks including ones that track adversarial-set gradients through simulated training [31, 73]. For those attacks, adversaries can craft $D_{\text{adv}}$ to exhibit particular gradient signatures at the end of training to avoid static detection. Moreover, models learn adversarial data faster than clean data meaning training loss often drops abruptly and significantly early in training [43]. For an attack to succeed, adversarial instances must align with the target at some point during training, meaning dynamic methods can detect them.

Lastly, our threat model specifies that attackers never know the random batch sequence nor any randomly initialized parameters. Therefore, attackers can only craft $D_{\text{adv}}$ to be influential in expectation over that randomness. Influence is stochastic, varying significantly across random seeds. However, estimating the true expected influence is computationally expensive. GAS and TracInCP, which simulate expectation, better align with how the adversary actually crafts the adversarial set, resulting in better $D_{\text{adv}}$ identification.

Below we detail how renormalization can be specialized further for better adversarial-set identification.

Extending Renormalization Layerwise: In practice, gradient magnitudes are often unevenly distributed across a neural network’s layers. For example, Figure 3 tracks an attack target’s average intra-training gradient magnitude for two different backdoor adversarial triggers on CIFAR10 binary classification ($y_{\text{aug}} = \text{airplane}$ and $y_{\text{adv}} = \text{bird}$). Specifically, target gradient norm, $\|\nabla_{\theta_t} L(z_{\text{aug}}; \theta_t)\|$, is decomposed into just the contributions of the network’s first convolutional layer (Conv1) and the final linear layer. Despite being only 0.04% of the model parameters, these two layers combined constitute >50% of the gradient norm. Therefore, the first and last layers’ parameters are, on average, weighted >2,000× more than other layers’ parameters. With simple renormalization, important parameters in those other layers may go undetected.

As an alternative to simply renormalizing by $\|\nabla_{\theta_t} L(z; \theta_t)\|$, partition gradient vector $g$ by layer into $L$ disjoint vectors (where $L$ is model $f$’s layer count) and then independently renormalize each subvector separately. This layerwise renormalization can be applied to any estimator that uses training gradient $g_t$ or test gradient $\hat{g}_t$, including influence functions, TracIn, and TracInCP. Layerwise renormalization still corrects for the low-loss penalty and does not change the asymptotic complexity. To switch GAS to layerwise, the only modification to Algorithm 2 is on Line 10 where each dimension is divided by its corresponding layer’s norm instead of the full gradient norm.

Notation: “-L” denotes layerwise renormalization, e.g., layerwise GAS is GAS-L. Suffix “(-L)”, e.g., GAS(-L), signifies that a statement applies irrespective of whether the renormalization is layerwise.

4.5 Renormalization & Non-Adversarial Data

Renormalization not only improves performance identifying an inserted adversarial set; it also improves performance in non-adversarial settings. Sec. 3.2 defines influence w.r.t. a single training example. Just as one instance may be more influential on a prediction than another, a group of training instances may be more influential than a different group. Renormalization improves identification of influential groups of examples, even on non-adversarial data.

To empirically demonstrate this, consider CIFAR10 binary classification again. In each trial, ResNet9 was pre-trained on eight ($= 10 - 2$) held-out CIFAR10 classes. From the other two classes, test example $z_{\text{filt}}$ was selected u.a.r. from those test instances with a moderate classification rate (10-20%) across multiple retrainings (i.e., fine-tunings) of the pre-trained network. Renormalized influence was then calculated for $z_{\text{filt}}$, with each estimator yielding a training-set ranking. Each estimator’s top $p\%$ ranked instances were removed from the training set and 20 models trained from the pre-trained parameters using these reduced training sets. Performance is measured using $z_{\text{filt}}$’s misclassification rate across those 20 models where a larger error rate entails a better overall ranking.

Figure 4 compares influence estimation’s filtering performance, with and without renormalization, against a random baseline averaged across five CIFAR10 class pairs, namely the two pairs specified by Weber et al. [78] and three additional random pairs. Influence, irrespective of renormalization, significantly outperformed random removal, meaning all of these estimators found influential subsets, albeit of varying quality. In all cases, renormalized influence had better or equivalent performance to the original estimator across all filtering fractions. This demonstrates that renormalization generalizes across estimators even beyond adversarial settings.

Overall, layerwise renormalization was the top performer across all setups except for large filtering percentages where GAS surpassed it slightly. Renormalized(-L) Influence functions and GAS(-L) performed similarly when filtering a small fraction (e.g., ≤5%) of the training data. However, the performance of renormalized influence functions plateaued for larger filtering fractions (≥10%) while...
for the same two attacks, no

| Inf. Func. Rn. (ours) | GAS-L (ours) |
|----------------------|--------------|

Renormalization (Rn.) always improved mean per-

This experiment again demonstrates that loss-based renormaliza-

GAS-L’s performance continued to improve. In addition, renormali-

GAS(-L)’s performance advantage over vanilla influence functions

GAS(-L) and TracInCP remained consistent. Recall that dynamic methods (e.g., GAS-L and TracIn) use significantly more

This experiment again demonstrates that loss-based renormaliza-

GAS(-L) can be significantly

FIT analyzes each test instance’s influence vector \( \mathbf{v} \) and ranks those

The plotted theoretical normal used robust statistics median and

5 IDENTIFYING ATTACK TARGETS

Recall that non-targets have primarily weak influences and few very

The idea is the core of our framework for identifying targets

Overall, FIT has three sub-steps, described chronologically:

(1) Inf.: Calculates (renormalized) influence vector \( \mathbf{v} \) for each test

AnomScore: Targets have an unusual number of highly-influential

Leveraging ideas from anomaly detection, this step

For simplicity, Alg. 3 considers a single identified target. If there are multiple identi-

Mitigate: Target-driven mitigation sanitizes model parameters

FIT is referred to as a “framework” since these subroutines are gen-

Algorithm 3 FIT target identification & mitigation

Input: Training set \( D_h \), test example set \( \tilde{Z}_{te} \), and final params. \( \theta_f \)

Output: Sanitized model parameters \( \tilde{\theta}_f \) & training set \( \tilde{D}_h \)

1. \( \mathcal{V} \leftarrow \{ \text{Inf}(\tilde{Z}, D_h) : \tilde{z} \in \tilde{Z}_{te} \} \quad \text{(Renorm.) Inf. (Alg. 2)} \)

5. \( \tilde{\theta}_f, \tilde{D}_h \leftarrow \text{Mitigate}(\tilde{Z}_{te}, \theta_f, D_h) \) \quad \text{Sec. 5.3}

- return \( \tilde{\theta}_f, \tilde{D}_h \)

5.1 Measuring (Renormalized) Influence

Algorithm 3 is agnostic of the specific (renormalized) influence estimator used to calculate \( \mathbf{v} \), provided that method is sufficiently adept at identifying adversarial set \( D_{adv} \). We use GAS(-L) for the reasons explained in Section 4 as well as its simplicity, computational efficiency, and strong, consistent empirical performance.

Time and Space Complexity: Computing a gradient requires \( O(|\theta|) \) time and space. For fixed \( T \) and \( |\theta| \), TracInCP, GAS, and GAS-L require \( O(n) \) time and space to calculate each test instance’s influence vector \( \mathbf{v} \). The next section explains that FIT analyzes each test instance’s influence vector \( \mathbf{v} \) meaning GAS(-L) can be significantly sped-up by amortizing training gradient \( O(t^{12}) \) computation across multiple test examples – either on a single node (suppl. Sec. F.10) or across multiple nodes (e.g., using all-reduce).

5.2 Identifying Anomalous Influence

To change a prediction, adversarial set \( D_{adv} \) must be highly influ-

For two different training-set attacks – the first poisoning on vision [84] and the other a backdoor attack on speech recognition [47]. For both attacks, adversarial set \( D_{adv} \’s \) influence significantly exceeds that of \( D_h \). When compared to theoretical normal (calculated\(^{12}\) w.r.t. complete training set \( D_h \), \( D_{adv} \’s \) target influence is highly anomalous. In Figures 5b and 5e, which plot the GAS influence of non-targets for the same two attacks, no extremely high influence instances are present.

\(^{12}\) The plotted theoretical normal used robust statistics median and \( Q \) in place of mean and standard deviation.
Going forward, influence vectors \( v \) with exceptionally high influence instances are referred to as having a heavy upper tail. Then, target identification simplifies to identifying influence vectors whose values have anomalously heavy upper tails. The preceding insight is relative and is w.r.t. to other test instances’ influence value distributions. Non-target baseline anomaly quantities vary with model, dataset, and hyperparameters. That is why suppl. Algorithm 6 ranks candidates in \( \hat{Z}_{\text{te}} \) based on their upper-tail heaviness.

**Quantifying Tail Heaviness:** Determining whether \( \hat{Z}_{\text{te}} \)’s influence vector \( v \) is abnormal simplifies to univariate anomaly detection for which significant previous work exists [3, 30, 60, 61]. Observe in Figures 5b and 5e that \( D_{\text{tl}} \)’s GAS influence vector \( v \) tends to be normally distributed (see the close alignment to the dashed line). We, therefore, use the traditional anomaly score, \( \sigma := \frac{Z_{\text{te}} - \mu}{\sigma} \), where \( \mu \) and \( \sigma \) are each \( v \)’s center and dispersion statistics, resp.\(^\text{(13)}\) Mean and standard deviation, the traditional center and dispersion statistics, resp., are not robust to outliers. Both have an asymptotic breakdown point of 0 (one anomaly can shift the estimator arbitrarily). Since \( D_{\text{adv}} \) instances are inherently outliers, robust statistics are required.

Median serves as our center statistic \( \mu \) given its optimal breakdown (50%). Although median absolute deviation (MAD) is the best known robust dispersion statistic, we use Rousseuw and Croux’s \[59] Q estimator, which retains MAD’s benefits while addressing its weaknesses. Specifically, both MAD and \( Q \) have optimal breakdowns, but \( Q \) has better Gaussian data efficiency (82% vs. 37%). Critically for our setting with one-sided anomalies, \( Q \) does not assume data symmetry – unlike MAD. Formally,

\[
Q = c \{ |v_i - v_j| : 1 \leq i < l \leq n \}_{(r)},
\]

where \( \{ \cdot \}_{(r)} \) denotes the set’s \( r \)-th order statistic with \( r = \lfloor \frac{n+1}{2} \rfloor \) and \( c \) is a distribution consistency constant which for Gaussian data, \( c \approx 2.2219 \) [60]. Eq. (13) requires only \( O(n) \) space and \( O(n \log n) \) time as proven by Croux and Rousseuw [15]. Provided anomaly score vector \( \sigma \), upper-tail heaviness is simply \( \sigma \left( \frac{n - \kappa}{n} \right) \), which is \( \sigma \)’s \( (n - \kappa) \text{th} \) order statistic, i.e., \( \hat{Z}_{\text{te}} \)’s \( \kappa \text{th} \) largest anomaly score value. The value of \( \kappa \) implicitly affects the size of the smallest detectable attack, where any attack with \( |D_{\text{adv}}| < \kappa \) is much harder to detect.

**Multiclass vs. Binary Classification** Different classes are implicitly generated from different data distributions. Each class’s data distribution may have different influence tails – in particular in multiclass settings. Target identification performance generally improves (1) when \( \mu \) and \( Q \) are calculated w.r.t. only training instances labeled \( y_{\text{tr}} \) and (2) \( \hat{Z}_{\text{te}} \)’s upper-tail heaviness is ranked w.r.t. other test instances labeled \( y_{\text{te}} \).

**Faster FIT** The execution time of TracInCP and by extension GAS(-L), depends on parameter count \( |\theta| \). For very large models, target identification can be significantly sped up via a two-phase strategy. In phase 1, GAS(-L) uses a very small iteration subset (e.g., \( T = (T) \)) to coarsely rank analysis set \( \hat{Z}_{\text{te}} \). Phase 2 then uses the complete \( T \) but only on a small fraction (e.g., 10%) of \( \hat{Z}_{\text{te}} \) with the heaviest phase 1 tails. Section 6.3 applies this approach to natural-language data poisoning on RoBERTa\_BASE [48].

Computing each test instance’s \( \hat{Z}_{\text{te}} \in \hat{Z}_{\text{te}} \) influence vector \( v \) is independent. Each dimension \( v_j \) is also independent and can be separately computed. Hence, GAS(-L) is embarrassingly parallel allowing linear speed-up of target identification via parallelization.

**5.3 Target-Driven Attack Mitigation**

A primary benefit of target identification is that attack mitigation becomes straightforward. Algorithm 4 mitigates attacks by sanitizing training set \( D_{\text{ah}} \) of adversarial set \( D_{\text{adv}} \). Most importantly, target
We empirically demonstrate our method's generality by evaluating this threshold could be set empirically or using domain-specific vision, and speech recognition. We consider both poisoning and held-out loss. These quantities can be measured cumulatively or for targets individually. Sanitization stops when the identification solves data sanitization's common pitfall (Sec. 3.1) of determining how much data to remove. Sanitization stops when the target's misprediction is eliminated. Therefore, successfully identifying a target means sanitization is guaranteed to succeed tautologically (i.e., attack success rate on any analyzed targets is 0).

More concretely, Alg. 4 iteratively filters $D_{\text{adv}}$ by thresholding anomaly score vector, $\sigma$ $\geq 14$. Since adversarial instances are abnormally influential on targets, Alg. 4 filters $D_{\text{adv}}$ instances first. After each iteration, influence is remeasured to account for estimation stochasticity and because training dynamics may change with different training sets. Data removal cutoff $\zeta$ is tuned based on computational constraints – larger $\zeta$ results in less clean data removed but may take more iterations. Slowly annealing $\zeta$ also results in less clean-data removal. Given forensic or human analysis of the identified target(s), simpler mitigation than Algorithm 4 is possible, e.g., a naive, rule-based, corrective lookup table that entails no clean data removal at all. For learning environments where certified training data deletion is possible [25, 49], retraining (Alg. 4 Line 7) may not even be required — making our method even more efficient.

Enhancing Mitigation’s Robustness An adversary could attack FIT by injecting adversarial instances into $D_t$ to specifically trigger excessive, unnecessary sanitization. To mitigate such a risk, Alg. 4 could be tweaked to include a maximum sanitization threshold that would trigger additional (e.g., human, forensic) analysis. This threshold could be set empirically or using domain-specific knowledge (e.g., maximum possible poisoning rate). See supplemental Section F.4 for further discussion.

6 Evaluation

We empirically demonstrate our method’s generality by evaluating training-set attacks on different data modalities, including text, vision, and speech recognition. We consider both poisoning and backdoor attacks on pre-trained and randomly-initialized, state-of-the-art models in binary and multiclass settings. Due to space, most evaluation setup details (e.g., hyperparameters) are deferred to suppl. Section C. Additional experimental results also appear in the supplement, including an analysis of a novel adversarial attack on target-driven mitigation (Sec. F.4), a poisoning-rate ablation study (Sec. F.5), a hyperparameter sensitivity study (Sec. F.6), an alternative renormalization approach (Sec. F.7), analysis of gradient aggregation’s benefits (Sec. F.8), & execution times (Sec. F.10).

6.1 Training-Set Attacks Evaluated

We evaluated our method on four published training-set attacks – two single-target data poisoning and two multi-target backdoor. Below are brief details regarding how each attack crafts adversarial set $D_{\text{adv}}$, with the full details in suppl. Sec. C.2.4. Representative clean and adversarial training instances for each attack appear in suppl. Sec. E. Table 1 lists each attack’s mean success rate aggregated across all related setups. Full granular results are in Section F.1.

Below, $y_{\text{adv}} \rightarrow y_{\text{true}}$ denotes the target’s true and adversarial labels, respectively. When an attack considers multiple class pairs or setups, each is evaluated separately.

(1) Speech Backdoor: Liu et al.’s [47] speech recognition dataset contains spectrograms of human speech pronouncing in English digits 0 to 9 (10 classes, $|D_{\text{cl}}| = 3,000 – 1\%$ backdoors). Liu et al. also provide 300 backdoored training instances evenly split between the 10 classes. Each class’s adversarial trigger – a short burst of white noise at the recording’s beginning – induces the spoken digit to be misclassified as the next largest digit (e.g., 0 $\rightarrow$ 1, 1 $\rightarrow$ 2, etc.). This small input-space signal induces a large feature-space perturbation – too large for many certified methods. Following Liu et al., our evaluation used a speech recognition CNN trained from scratch.

(2) Vision Backdoor: Weber et al. [78] consider three different backdoor adversarial trigger patterns on CIFAR10 binary classification. Specifically, Weber et al.’s “pixel” attack patterns increase the pixel value of either one or four central pixel(s) by a specified maximum $\ell_2$ perturbation distance while their “blend” trigger pattern adds fixed $N(0, I)$ Gaussian noise across all perturbed images. We considered the same class pairs as Weber et al. (auto $\rightarrow$ dog and p1lane $\rightarrow$ b1rd) on the state-of-the-art ResNet9 [52] CNN trained from scratch with $|D_{\text{adv}}| = 150$ and $|D_{\text{tr}}| = 10,000 (1.5\%$ backdoors).

(3) Natural Language Poison: Wallace et al. [73] construct text-based poison by simulating bilevel optimization via second-order gradients. $D_{\text{adv}}$’s instances are crafted via iterative word-level substitution given a target phrase. We follow Wallace et al.’s [73] experimental setup of poisoning the Stanford Sentiment Treebank v2 (SST-2) sentiment analysis dataset [66] ($|D_{\text{cl}}| = 6,349$ & $|D_{\text{adv}}| = 50 – 0.07\%$ poison) on the RoBERTaBASE transformer architecture (125M parameters) [48].

(4) Vision Poison: Zhu et al.’s [84] targeted, clean-label attack crafts poisons by forming a convex polytope around a single target’s feature representation. Following Zhu et al., the pre-train then fine-tune paradigm was used. In each trial, ResNet9 was pre-trained using half the classes (none were $y_{\text{adv}}$ or $y_{\text{true}}$). Targets were selected uniformly at random (u.a.r.) from test examples labeled $y_{\text{true}}$, and 50 poison instances (0.2% of $D_{\text{tr}}$) were then crafted from seed examples labeled $y_{\text{adv}}$. The pre-trained network was fine-tuned using $D_{\text{adv}}$ and the five held-out classes’ training data ($|D_{\text{tr}}| = 25,000$). Like previous work [31, 64], CIFAR10 class pairs dog vs. b1rd and

---

**Algorithm 4 Target-driven mitigation & sanitization**

**Input:** Target $D_{\text{aug}} \Rightarrow (x_{\text{aug}}, y_{\text{adv}})$, anomaly cutoff $\zeta$, model $f$, initial params. $\theta_{\text{tr}}$, final params. $\theta_{\text{f}}$, and training set $D_{\text{tr}}$

**Output:** Clean model parameters $\theta_{\text{f}}$ & sanitized training set $D_{\text{tr}}$$\setminus D_{\text{adv}}$

1. **function** Mitigate($D_{\text{aug}}$, $\theta_{\text{f}}$, $D_{\text{tr}}$) $\leftarrow$ 
2. $\hat{\theta}_{\text{f}}$, $D_{\text{tr}}$ $\leftarrow$ $\theta_{\text{f}}$, $D_{\text{tr}}$
3. while $\arg \max f(x_{\text{aug}}, \hat{\theta}_{\text{f}}) = y_{\text{adv}}$ do
4. $v \leftarrow \inf D_{\text{aug}}$ $\setminus \hat{D}_{\text{tr}}$ $\Rightarrow$ Renorm. Influence (Alg. 2)
5. $\sigma \leftarrow \frac{v - y_{\text{adv}}}{\mu}$ $\Rightarrow$ Anomaly score (Sec. 3.2)
6. $D_{\text{tr}} \leftarrow D_{\text{tr}} \setminus \{z_i : \sigma_i \geq \zeta \land z_i \in D_{\text{tr}}\}$ $\Rightarrow$ Sanitize
7. $\hat{\theta}_{\text{f}} \leftarrow \text{Retrain}(\theta_{\text{tr}}, D_{\text{tr}})$
8. Optionally anneal $\zeta$
9. **return** $\hat{\theta}_{\text{f}}$, $D_{\text{tr}}$
deer vs. frog were evaluated, where each class in a pair serves alternately as $y_{\text{adv}}$ and $u_{\text{adv}}$.

While it is not feasible to evaluate our approach on every attack (as new attacks are developed & published so frequently) we believe this diverse set of attacks is representative of training-set attacks in general and demonstrates our approach’s broad applicability. In particular, our method is not tailored to these attacks and could be used against future attacks as well, as long as the attack includes highly influential training examples that attack specific targets.

### 6.2 Identifying Adversarial Set $D_{\text{adv}}$

To identify the target (Alg. 3) or mitigate the attack (Alg. 4), we must be able to identify the likely adversarial instances $D_{\text{adv}}$ associated with a possible target $z_{\text{arg}}$. Our approach is to use influence-estimation methods, which should rank an adversarial attack $D_{\text{adv}}$ as more influential than clean instances $D_{\text{cl}}$ on the target. In this section, we evaluate how well different influence-estimation methods succeed at performing this ranking for a given target.

We compare the performance of our renormalized estimators, GAS and GAS-L, against Section 3.2’s four influence estimators: TracInCP, TracIn, influence functions, and representer point. As an even stronger baseline, where applicable, we also compare against Peri et al.’s [55] Deep $k$-NN empirical training-set defense specifically designed for Zhu et al.’s [84]’s vision, clean-label poisoning attack; described briefly, Deep $k$-NN sanitizes the training set of instances whose nearest feature-space neighbors have a different label. Like Section 4’s CIFAR10 & MNIST joint classification experiment, class sizes are imbalanced ($|D_{\text{adv}}| \ll |D_{\text{cl}}|$) so performance is again measured using AUPRC.

For targets selected u.a.r., Figure 6 details each method’s averaged adversarial-set identification AUPRC for Section 6.1’s four attacks. In summary, GAS and GAS-L were each the top performer for one attack and had comparable performance for the other two.

GAS and GAS-L identified the adversarial instances nearly perfectly for Liu et al.’s speech backdoor and Wallace et al.’s text poisoning attacks. Standard influence estimation performed poorly on the text poisoning attack (in particular the static estimators) due to the large model, RoBERTa$_{BASE}$, that Wallace et al.’s attack considers. For the vision backdoor and poisoning attacks, our renormalized estimators successfully identified most of $D_{\text{adv}}$ — again, much better than the four original estimators. While Peri et al.’s [55] Deep $k$-NN defense can be effective at stopping clean-label vision poisoning, it does so by removing a comparatively large fraction of clean data (up to 4.3% on average) resulting in poor AUPRC.

For completeness, Figure 8 provides adversarial-set identification results for our renormalized, static influence estimators. In all cases, renormalization improved the estimator’s performance, generally by an order of magnitude with a maximum improvement of 600×. These experiments highlight layerwise renormalization’s benefits. Influence functions’ Hessian-vector product algorithm [54] can assign a large magnitude to some layers, and these layers then dominate the influence and GAS estimates. Layerwise renormalization addresses this, improving renormalized influence function’s adversarial-set identification AUPRC by up to $3.5\times$.

### 6.3 Identifying Attack Targets

The previous experiments demonstrate that knowledge of a target enables identification of the adversarial set when using renormalization. This section demonstrates that the distribution of renormalized influence values actually enables us to identify target(s) in the first place, through the interplay foundational to our target identification framework, FIT. Since target identification is a new task, we propose four target identification baselines. First, inspired by Peri et al.’s [55] Deep $k$-NN empirical defense, maximum $k$-NN distance computes the distance from each test instance to its $k$th nearest neighbor in the training data, as measured by the $L_2$ distance between their penultimate feature representations ($f$). It orders them by this distance, starting with the largest distance to the $k$th neighbor, thus prioritizing outliers and instances in sparse regions of the learned representation space. Minimum $k$-NN distance is the reverse ordering, prioritizing instances in dense regions. The other two baselines are most certain, which ranks test examples in ascending order by loss while least certain ranks by descending loss.

---

**Figure 6: Adversarial-Set Identification** Mean AUPRC identifying adversarial set $D_{\text{adv}}$ using a randomly selected target for Sec. 6.1’s four attacks. Results averaged across related setups with $\geq 10$ trials per setup. See supplemental Section F.1 for the full granular results, including variance.

**Figure 7: Target Identification** Mean target identification AUPRC for Sec. 6.1’s four attacks. “FIT w/ GAS” denotes GAS was FIT’s influence estimator with matching notation for GAS-L. Results averaged across setups with $\geq 10$ trials per setup. See Sec. F.1 for the full granular results, inc. variance.
There are far fewer targets than possible test examples so performance is again measured using AUPRC. See suppl. Table 6 for the number of targets and non-targets analyzed for each attack. For single-target attacks (vision and natural language poisoning), target identification AUPRC is equivalent to the target’s inverse rank, causing AUPRC to decline geometrically.

Figure 7 shows that FIT – using either GAS or GAS-L as the influence estimator – achieves near-perfect target identification for both backdoor attacks and natural language poisoning. Overall, FIT with GAS was the top performer on two attacks, and FIT with GAS-L was the best for the other two. Recall that the vision poisoning attack is single target. Hence, GAS-based FIT’s mean AUPRC of >0.8 equates to an average target rank better than 1.25 (1/0.8), i.e., three out of four times on average was the top-ranked – also very strong target detection. FIT’s performance degradation on vision poisoning is due to GAS and GAS-L identifying this attack’s $D_{adv}$ slightly worse (Fig. 6). Only maximum $k$-NN approached FIT’s performance – specifically for Weber et al.’s vision backdoor attack. Note also that no baseline consistently outperformed the others. Hence, these attacks affect network behavior differently, further supporting that FIT is attack agnostic.

Suppl. Sec. F.6 shows that FIT’s performance is stable across a wide range of upper-tail cutoff thresholds $\kappa$. For example, FIT’s natural language target identification AUPRC varied only 0.2% and 2.1% when using GAS-L and GAS respectively for $\kappa \in [1, 25]$.

### 6.4 Target-Driven Mitigation

Section 5.3 explains that successfully identifying the target(s) enables guaranteed attack mitigation on those instances. Here, we evaluate GAS and GAS-L’s effectiveness in targeted data sanitization. Table 1 details our defense’s effectiveness against Sec. 6.1’s four attacks. As above, results are averaged across each attack’s class pairs/setups. Section 6’s baselines all have large false-positive rates when identifying $D_{adv}$ (Fig. 6), which caused them to remove a large fraction of $D_{adv}$ and are not reported in these results.

For three of four attacks, clean test accuracy after sanitization either improved or stayed the same. In the case of Weber et al.’s vision backdoor attack, the performance degradation was very small – 0.1%. Similarly, owing to renormalized influence’s effectiveness identifying $D_{adv}$ (Fig. 7), our defense removes very little clean data when mitigating the attack – generally <0.2% of the clean training set. For comparison, Peri et al. [55] report that their Deep k-NN clean-label, poisoning defense removes on average 4.3% of $D_{adv}$ on Zhu et al.’s [84] vision poisoning attack. This is despite Peri et al.’s method being specifically tuned for Zhu et al.’s attack and their evaluation setup being both easier and less realistic by pre-training their model using a large known-clean set that is identically distributed to their $D_{adv}$. In contrast, target-driven mitigation removed at most 0.03% of clean data on this attack – better than Peri et al. by two orders of magnitude.

Following Algorithm 4, Table 1’s experiments used only a single, randomly-selected target when performing sanitization. No steps were taken to account for additional potential targets, e.g., over-filtering the training set. Nonetheless, target-driven mitigation still significantly degraded multi-target attacks’ performance on other targets not considered when sanitizing. For example, despite considering one target, speech backdoor’s overall attack success rate (ASR) across all targets decreased from 100% to 4.7% and 6.5% for GAS and GAS-L, respectively – a 20× reduction. For Weber et al.’s vision backdoor attack, ASR dropped from 90.5% to 11.9% and 6.7% with GAS and GAS-L, respectively. The key takeaway is that identifying a single target almost entirely mitigates the attack everywhere.

### 7 ADAPTIVE ATTACKS

We now consider how an attacker who knows about our defense could evade it or otherwise exploit it. Our method relies on multiple
The baseline results (orange) used Zhu et al.’s standard attack. Our jointly-optimized attack reduced the GAS similarity by 7% at the cost of a 19% decrease in ASR w.r.t. Table 1. See suppl. Sec. F.2 for the granular results, including variance.

Figure 9: Adversarial-Set Identification for the Adaptive Vision Poison Attack: Mean AUPRC identifying the adversarial set where Zhu et al.’s vision poison attack is adapted to jointly minimize the adversarial loss and the GAS influence. The baseline results (orange) used Zhu et al.’s standard attack. Our jointly-optimized attack reduced the GAS similarity by 7% at the cost of a 19% decrease in ASR w.r.t. Table 1. See suppl. Sec. F.2 for the granular results, including variance.

Figure 10: Target Identification for the Adaptive Vision Poison Attack: Mean target identification AUPRC where Zhu et al.’s vision poison attack is jointly optimized with minimizing GAS. FIT with GAS’s mean target identification AUPRC declined only 9% versus the baseline – an average change in target rank of 1.16 to 1.28 – still strong performance. Results are averaged across related setups with ≥ 10 trials per setup. See suppl. Sec. F.2 for the full results, including variance.

Since GAS – like poison – relies on the entire training trajectory of the model, which in turn relies on the perturbations being crafted, computing GAS’s exact gradient is intractable [8]. However, the attacker can still use a surrogate that approximates GAS, such as by using fixed model checkpoints in the computation of GAS.

To evaluate the robustness of our methods to adaptive perturbations, we apply this joint optimization idea to Zhu et al.’s [84] vision poison attack. We focused on Zhu et al.’s attack because (1) it is the attack on which our method performed the worst and (2) the other optimized attack we consider [73] is restricted to only discrete token replacements, which reduces the attacker’s flexibility.
Table 2: Attack Mitigation for the Adaptive Vision Poison Attack

This paper explores two related tasks. First, we propose training-set optimization with minimizing the GAS influence. The results below consider exclusively the jointly-optimized attack with $\lambda = 10^{-2}$. Clean-data removal remains low, and test accuracy either improved or stayed the same for in but one setup. The performance is comparable to the results with Zhu et al.’s [84]’s standard vision poisoning attack (see Table 29). Bold denotes the best mean performance with $\geq$ 10 trials per class pair.

| Classes | Method | % Removed | ASR % | Test Acc. % |
|---------|--------|-----------|-------|-------------|
| ytrag  | yadv   | $D_{adv}$ | $D_{adv}$ | Orig. | Ours | Orig. | Chg. |
| Bird    | Dog    | GAS       | 36.0  | 0.02 | 76.2 | 0 | 87.0 | $+0.1$ |
| Dog     | Bird   | GAS-L     | 30.3  | 0.00 | 57.1 | 0 | 87.1 | $+0.1$ |
| Frog    | Deer   | GAS       | 21.6  | 0.00 | 57.1 | 0 | 87.1 | $-0.1$ |
| Deer    | Frog   | GAS-L     | 19.4  | 0.00 | 38.1 | 0 | 87.1 | 0.0 |
|         |        | GAS-L     | 82.3  | 0.13 | 81.0 | 0 | 87.1 | $+0.1$ |

optimization’s effect on target identification. Overall, joint optimization reduced FIT with GAS’s mean target identification AUPRC from a baseline of 0.86 to 0.78 (9% drop). Since Zhu et al.’s attack is single-target, this translates to the target’s average rank declining from 1.16 to 1.28 — still high performance.

Table 2 details target-driven mitigation’s effectiveness under this jointly-optimized attack. In summary, the attack results in very little clean data removal (at most 0.05% of $D_{adv}$ on average). Also, the average test accuracy after mitigation either improved or stayed the same in all but one case where it decreased by only 0.1%.

In summary, even when the adversary specifically optimized for our defense, we still effectively identify both the adversarial set and the target and then mitigate the adaptive attack.

8 DISCUSSION AND CONCLUSIONS

This paper explores two related tasks. First, we propose training-set target identification. This task is an important part of protecting critical ML systems but has thus far received relatively little attention. For example, it is impossible to conduct a truly informed cost-benefit analysis of risk without knowing the attacker’s target and by extension their objective. Knowledge of the target also enables forensic and security analysts to reason about an attacker’s identity—a key step to permanently stopping attacks by disabling the attacker. An open question is whether target identification can be combined with certified guarantees, either building on our FIT framework or creating an alternative to it.

FIT relies on identifying (groups of) highly influential training instances. To that end, we propose renormalized influence. By addressing influence’s low-loss penalty, renormalization significantly improves influence estimation in both adversarial and non-adversarial settings—often by an order of magnitude or more.

ACKNOWLEDGMENTS

The authors thank Jonathan Brophy for helpful discussions. This work was supported by a grant from the Air Force Research Laboratory and the Defense Advanced Research Projects Agency (DARPA) — agreement number FA8750-16-C-0166, subcontract K001892-00-S05, as well as a second grant from DARPA, agreement number HR00112090135. This work benefited from access to the University of Oregon’s HPC Talapas.

REFERENCES

[1] Shruti Agarwal, Harry Farid, Yuming Gu, Mingming He, Koki Nagano, and Hao Li. 2019. Protecting World Leaders Against Deep Fakes. In Proceedings of the CVPR Workshop on Media Forensics. Long Beach, California, USA.
[2] Hojat Aghakhani, Thorsten Essenhofer, Lea Schön herr, Dorothea Kolossa, Thorsten Holz, Christopher Kruegel, and Giovanni Vigna. 2020. VENO-MAVE: Clean-Label Poisoning Against Speech Recognition. CoRR (2020). arXiv:2010.10682 [cs.SD]
[3] Vic Barnett and Toby Lewis. 1978. Outliers in Statistical Data (2nd edition ed.). John Wiley & Sons Ltd., Hoboken, New Jersey, USA.
[4] Elina Bharath, Marc-Etienne Brunet, and Gintare Karolina Dziugaite. 2020. RelativI: Identifying Explanatory Training Samples via Relative Influence. In Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS’20).
[5] Sanyadeep Basu, Phil Pope, and Soheil Feizi. 2021. Influence Functions in Deep Learning Are Fragile. In Proceedings of the 9th International Conference on Learning Representations (ICLR’21). Virtual Only.
[6] Sanyadeep Basu, Xuchen You, and Soheil Feizi. 2020. On Second-Order Group Influence Functions for Black-Box Predictions. In Proceedings of the 37th International Conference on Machine Learning (ICML’20). PMLR, Virtual Only.
[7] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning Attacks against Support Vector Machines. In Proceedings of the 29th International Conference on Machine Learning (ICML’12). PMLR, Edinburgh, Great Britain.
[8] Avrim L. Blum and Ronald L. Rivest. 1992. Training a 3-node neural network is NP-complete. Neural Networks 5, 1 (1992), 117–127.
[9] Jonathan Brophy, Zayd Hammoudeh, and Daniel Lowd. 2022. Adapting and Evaluating Influence-Estimation Methods for Gradient-Boosted Decision Trees. arXiv:2205.00359 [cs.LG] arXiv.
[10] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. 2019. Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. In Proceedings of the AAAI Workshop on Artificial Intelligence Safety (SafeAI’19). Association for the Advancement of Artificial Intelligence, Honolulu, Hawaii, USA.
[11] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. arXiv:1712.05526 [cs.CR] arXiv.
[12] Yuanyuan Chen, Boyang Li, Han Yu, Pengcheng Wu, and Chunyan Miao. 2021. HyPaRelA: Hypergradient Data Relevance Analysis for Interpreting Deep Neural Networks. In Proceedings of the 59th AAAI Conference on Artificial Intelligence (AAAI’21). Association for the Advancement of Artificial Intelligence, Virtual Only.
[13] Cody A. Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. 2017. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. In Proceedings of the 2017 NeurIPS Workshop on Machine Learning Systems. Curran Associates, Inc., Long Beach, California, USA.
[14] R. Dennis Cook and Sanford Weisberg. 1982. Residuals and Influence in Regression. Chapman and Hall, New York.
[15] Christophe Croux and Peter J. Rousseeuw. 1992. A Class of High-Breakdown Scale Estimators Based on Subranges. Communications in Statistics - Theory and Methods 21, 7 (1992), 1935–1951.
[16] Jesse Davis and Mark Goadrich. 2006. The Relationship Between Precision-Recall and ROC Curves. In Proceedings of the 23rd International Conference on Machine Learning (ICML’06). PMLR, Pittsburgh, Pennsylvania.
[17] Bao Gia Doan, Ehsan Abbasnejad, and Damith C. Ranasinghe. 2020. Februx: Input Purification Defense Against Trojan Attacks on Deep Neural Network Systems. In Proceedings of the 36th Annual Computer Security Applications Conference (ACSAC’20). Association for Computing Machinery, Virtual Only.
[18] Vitaly Feldman and Chiyuan Zhang. 2020. What Neural Networks Memorize and Why: Discovering the Long Tail via Influence Estimation. In Proceedings of the 36th Annual Conference on Neural Information Processing Systems (NeurIPS’20). Curran Associates, Inc., Virtual Only.
[19] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In Proceedings of the 34th International Conference on Machine Learning (ICML’17). PMLR, Sydney, Australia.

Hammmoudeh & Lowd
Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In Proceedings of the 8th Conference on Empirical Methods in Natural Language Processing (EMNLP’13).

Ezekiel O. Soremekun, Sakshi Udeshi, Sudipta Chattopadhyay, and Andreas Zeller. 2020. Exposing Backdoors in Robust Machine Learning Models. arXiv:2003.08665 [cs.LG]

Jacob Steinhardt, Pang Wei Koh, and Percy Liang. 2017. Certified Defenses for Data Poisoning Attacks. In Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS’17). Curran Associates, Inc., Long Beach, California, USA.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing Properties of Neural Networks. arXiv:1312.6199 [cs.CV]

Brandon Tran, Jerry Li, and Aleksander Madry. 2018. Spectral Signatures in Backdoor Attacks. In Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS’18). Curran Associates, Inc., Montreal, Canada.

Sakshi Udeshi, Shanshan Peng, Gerald Woo, Lionel Loh, Louth Rawshan, and Sudipta Chattopadhyay. 2019. Model Agnostic Defence against Backdoor Attacks in Machine Learning. arXiv:1908.02203 [cs.LG]

Miguel Villarreal-Vasquez and Bharat K. Bhargava. 2020. ConFoc: Content-Focus Protection Against Trojan Attacks on Neural Networks. arXiv:2007.00711 [cs.CV]

Eric Wallace, Tony Z. Zhao, Shi Peng, and Sameer Singh. 2021. Concealed Data Poisoning Attacks on NLP Models. In Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL’21).

Binghui Wang, Xiaoyu Cao, Jinyuan Jai, and Neil Zheng. 2020. On Certifying Robustness against Backdoor Attacks via Randomized Smoothing. In Proceedings of the CVPR Workshop on Adversarial Machine Learning in Computer Vision.

Bolin Wang, Yuanzhun Yao, Shawn Shang, Huizing Li, Bimal Virwani, Haitao Zheng, and Ben Y. Zhao. 2019. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In Proceedings of the 40th IEEE Symposium on Security and Privacy (SP’19). San Francisco, CA.

Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A. Efros. 2018. Dataset Distillation. arXiv:1811.10959 [cs]

Wenxiao Wang, Alexander Levine, and Soheil Feizi. 2022. Improved Certified Defenses against Data Poisoning with (Deterministic) Finite Aggregation. In Proceedings of the 39th International Conference on Machine Learning (ICML’22).

Maurice Weber, Xiaojun Xu, Bojan Karlaš, Ce Zhang, and Bo Li. 2021. RAB: Provably Robustness Against Backdoor Attacks. arXiv:2003.08904 [cs.LG]

Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. 2015. Is Feature Selection Secure against Training Data Poisoning?. In Proceedings of the 32nd International Conference on Machine Learning (ICML’15). PMLR, Lille, France.

Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A. Gunter, and Bo Li. 2021. Detecting AI Trojans Using Meta Neural Analysis. In Proceedings of the 42nd IEEE Symposium on Security and Privacy (SP’21). IEEE, Virtual Only.

Chih-Kuan Yeh, Joon Sik Kim, Ian E.H. Yen, and Pradeep Ravikumar. 2018. Representor Point Selection for Explaining Deep Neural Networks. In Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS’18). Curran Associates, Inc., Montreal, Canada.

Da Yu, Huihui Zhang, Wei Chen, Jian Yin, and Tie-Yan Liu. 2021. Indiscriminate Poisoning Attacks are Shortcuts. arXiv:2111.00898 [cs.LG]

Rui Zhang and Shihua Zhang. 2022. Rethinking Influence Functions of Neural Networks in the Over-Parameterized Regime. In Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI’22). Association for the Advancement of Artificial Intelligence, Vancouver, Canada.

Chen Zhu, W Ronny Huang, Ali Shafahi, Hengduo Li, Gavin Taylor, Christoph Studer, and Tom Goldstein. 2019. Transferable Clean-Label Poisoning Attacks on Deep Neural Nets. In Proceedings of the 9th International Conference on Machine Learning (ICML’19). PMLR, Los Angeles, CA.

Liuwan Zhu, Rui Ning, Cong Wang, Chunsheng Xin, and Hongyi Wu. 2020. GangSweep: Sweep out Neural Backdoors by GAN. In Proceedings of the 28th ACM International Conference on Multimedia (MM’20).

Liuwan Zhu, Rui Ning, Chunsheng Xin, Chonggang Wang, and Hongyi Wu. 2021. CLEAR: Clean-up Sample-Targeted Backdoor in Neural Networks. In Proceedings of the 18th International Conference on Computer Vision (ICCV’21).
Identifying a Training-Set Attack’s Target
Using Renormalized Influence Estimation

Supplemental Materials

Table of Contents

A Additional Algorithms .......................................................... A3
  A.1 Runtime Complexity of FIT .............................................. A3
B Proof ............................................................................... A4
C Detailed Experimental Setup .................................................. A4
  C.1 Dataset Configurations .................................................. A4
    C.1.1 Training Set Sizes ...................................................... A5
    C.1.2 Target Set Sizes ......................................................... A5
  C.2 Hyperparameters .......................................................... A6
    C.2.1 Model Training ........................................................ A6
    C.2.2 Upper-Tail Heaviness Hyperparameters ...................... A6
    C.2.3 Target-Driven Mitigation Hyperparameters ................. A7
    C.2.4 Adversarial Set $D_{\text{adv}}$ Crafting ......................... A7
    C.2.5 Baselines ............................................................... A8
  C.3 Network Architectures .................................................. A9
D Convex Polytope Poisoning & GAS Joint Optimization ............... A12
E Representative Perturbed Examples ....................................... A14
  E.1 Speech Recognition Backdoor ........................................ A14
  E.2 Vision Backdoor .......................................................... A14
  E.3 Natural Language Poison ............................................... A15
  E.4 Vision Poison ............................................................ A15
F Additional Experimental Results .......................................... A16
  F.1 Full Experimental Results .............................................. A16
    F.1.1 Speech Recognition Backdoor Full Results ................... A16
    F.1.2 Vision Backdoor Full Results ..................................... A18
    F.1.3 Natural Language Poisoning Full Results ................... A20
    F.1.4 Vision Poisoning Full Results .................................... A22
  F.2 Jointly-Optimized Adaptive Attacker – Full Experimental Results ..... A24
    F.2.1 Adversarial-Set Identification of the Jointly Optimized Poisoning Attack ... A24
    F.2.2 Target Identification of the Jointly Optimized Poisoning Attack ........... A25
    F.2.3 Target-Driven Attack Mitigation of the Jointly Optimized Poisoning Attack ... A26
  F.3 Adaptive Adversarial-Instance Selection .......................... A27
  F.4 An Adversarial Attack on Target-Driven Mitigation ............. A28
  F.5 Poisoning-Rate Ablation Study ...................................... A30
    F.5.1 Adversarial-Set Identification Ablation Study ................ A31
F.5.2 Target-Identification Ablation Study ................................................................. A32
F.5.3 Target-Driven Attack Mitigation Ablation Study ........................................... A33
F.5.4 Ablation Study Reference Tables ................................................................. A35
F.6 Upper-Tail Heaviness Hyperparameter $\kappa$ Sensitivity Analysis ....................... A38
F.7 Alternatives to Renormalization ...................................................................... A39
F.8 Why Aggregate Training Gradients? ............................................................... A40
F.9 Evaluating on the Tail of the Attack’s Influence Distribution ............................ A42
F.10 Adversarial-Set Identification Execution Time ............................................... A43
A ADDITIONAL ALGORITHMS

Algorithm 5 outlines TracIn’s influence estimation procedure for a priori unknown test instance \( \tilde{z}_{te} \). Algorithm 2, which combines the procedures of GAS and TracInCP, is re-included below to facilitate easier side-by-side comparison.

**Algorithm 5: TracIn influence estimation**

**Input:** Training parameters \( \mathcal{P} \), iteration subset \( T \), iteration count \( n_{T} \), batches \( B_{1}, \ldots, B_{T} \), batch size \( b \), and test example \( \tilde{z}_{te} \)

**Output:** Influence vector \( v \)

1: \( v \leftarrow 0 \)  \comment{Initialize}
2: for \( t \leftarrow 1 \) to \( T \) do
3:  if \( t \in T \) then
4:    \( (\eta, \theta) \leftarrow \mathcal{P}[t] \)  \comment{Equiv. to \( (\eta_{t}, \theta_{t-1}) \)}
5:    \( \tilde{g}_{te} \leftarrow \nabla_{\theta} \mathcal{L}(\tilde{z}_{te}; \theta) \)
6:  for each \( z_{i} \in B_{t} \) do
7:    \( g_{i} \leftarrow \nabla_{\theta} \mathcal{L}(z_{i}; \theta) \)  \comment{Batch examples}
8:    \( v_{i} \leftarrow v_{i} + \frac{\eta}{b} (g_{i}, \tilde{g}_{te}) \)  \comment{Unnormalized}
9: return \( v \)

Algorithm 6 overviews the implementation strategy of FIT that was used in our evaluation.

**Algorithm 6: FIT target identification implementation**

**Input:** Training set \( \mathcal{D}_{tr} \), training set size \( n \), test example set \( \tilde{Z}_{te} \), and upper-tail count \( k \)

**Output:** Sanitized model parameters \( \tilde{\theta}_{T} \) & training set \( \tilde{\mathcal{D}}_{tr} \)

1: \( \mathcal{V} \leftarrow \{ \text{GAS}(\tilde{z}_{j}; \mathcal{D}_{tr}) : \tilde{z}_{j} \in \tilde{Z}_{te} \} \)  \comment{Renorm. Inf. (Alg. 2)}
2: \( \Sigma \leftarrow \{ \sum(j) \text{vol}(j) : j \in \mathcal{V} \} \)  \comment{Anomaly score (Sec. 5.2)}
3: \( \mathcal{H} \leftarrow \{ \sigma(n-k) : \sigma \in \Sigma \} \)  \comment{Upper-tail heaviness (Sec. 5.2)}
4: Rank \( \tilde{Z}_{te} \) by heaviness \( \mathcal{H} \)
5: \( \tilde{\theta}_{T}, \tilde{D}_{tr} \leftarrow \theta_{T}, D_{tr} \)
6: for each target \( \tilde{z}_{cag} \) identified using \( \mathcal{H} \) do
7:    \( \tilde{\theta}_{T}, \tilde{D}_{tr} \leftarrow \text{MITIGATE}(\tilde{z}_{cag}, \tilde{\theta}_{T}, \tilde{D}_{tr}) \)  \comment{Sec. 5.3}
8: return \( \tilde{\theta}_{T}, \tilde{D}_{tr} \)

A.1 Runtime Complexity of FIT

We assume that the model being attacked (and defended) is a neural network with parameter vector \( \theta \), trained from dataset \( \mathcal{D}_{tr} \) using some form of stochastic descent for \( T \) epochs, and evaluated on test set \( \tilde{Z}_{te} \). For convenience, let \( n := |\mathcal{D}_{tr}| \) and \( m := |\tilde{Z}_{te}| \). Computing the gradient for a single example can be done in time \( O(|\theta|n) \), so training the network by computing gradient updates for all instances in all epochs is \( O(|\theta|nT) \).

For FIT (Algorithm 6), we need to estimate the influence of each training instance on each test instance. If we save the parameters from \( |T| \leq T \) checkpoints, we can run TracIn, TracInCP, or GAS-L once to compute the influence of each training instance on a single target, requiring time \( O(|\theta|n|T|) \). To compute the influence on all test examples requires computing their gradients at each checkpoint and computing dot products with each training example, for a total runtime of: \( O(|\theta|n|T|m) \). This grows linearly in the number of training examples times the number of test examples and is the slowest part of FIT in practice. An important question for future work is how to accelerate this procedure, such as heuristically pruning the number of training and test examples under consideration, similar to previous work [26].

For mitigation (Algorithm 4), model training and influence estimation are repeated as each test example or group of test examples is removed, until the predicted label changes. Let \( l \) denote the number of iterations before the label flips. However, the influence estimation only needs to be done for the selected target example, not for all test examples. Thus, the model retraining requires time \( O(|\theta|nTl) \), and the influence re-estimation requires time \( O(|\theta||T|l) \), which is less than the initial influence estimation time because \( l < m \).
B PROOF

Proof of Theorem 4.1

Theorem. Let loss function \( \bar{\ell} : \mathbb{R} \to \mathbb{R}_{\geq 0} \) be twice-differentiable and strictly convex as well as either even\(^{17}\) or monotonically decreasing. Then, it holds that

\[
\bar{\ell}(a) < \bar{\ell}(a') \implies \| \nabla_a \bar{\ell}(a) \|_2^2 < \| \nabla_a \bar{\ell}(a') \|_2^2,
\]

(14)

Proof.

Theorem 4.1 specifies that property,

\[
\bar{\ell}(a) < \bar{\ell}(a') \implies \| \nabla_a \bar{\ell}(a) \|_2^2 < \| \nabla_a \bar{\ell}(a') \|_2^2,
\]

holds when loss function \( \bar{\ell} \) is strictly convex (i.e., \( \forall a \in \mathbb{R} \nabla^2_a \bar{\ell}(a) > 0 \)) and either monotonically decreasing or even. We prove the claim separately for these two disjoint cases.

For any monotonically decreasing \( \bar{\ell} \), by definition

\[
\bar{\ell}(a) < \bar{\ell}(a') \implies a > a'.
\]

Then, given \( \forall a \in \mathbb{R} \nabla^2_a \bar{\ell}(a) > 0 \), it holds that

\[
\nabla_a \bar{\ell}(a) > \nabla_a \bar{\ell}(a').
\]

(15)

For any scalar, monotonically decreasing function \( \bar{\ell} \), it holds that \( \nabla_a \bar{\ell}(a') \leq 0 \) meaning Eq. (15)’s inequality flips w.r.t. L2 norms, i.e.,

\[
\| \nabla_a \bar{\ell}(a) \|_2^2 < \| \nabla_a \bar{\ell}(a') \|_2^2,
\]

(16)

as for any \( x, x' \in \mathbb{R}_{\leq 0} \) it holds that \( x > x' \implies \| x \|_2^2 < \| x' \|_2^2 \).

Formally, a function \( \bar{\ell} \) is even if

\[
\forall a \ nabla_a \bar{\ell}(a) = \bar{\ell}(-a).
\]

(17)

For even \( \bar{\ell} \), it holds that \( \nabla_a \bar{\ell}(0) = 0 \) provided twice differentiability. Given \( \forall a \in \mathbb{R} \nabla^2_a \bar{\ell}(a) > 0 \), then \( \forall a \in \mathbb{R} \nabla_a \bar{\ell}(a) < 0 \). Hence over restricted domain \( \mathbb{R}_{\leq 0} \), \( \bar{\ell} \) is monotonically decreasing. Above it was shown that Eq. (8) holds for monotonically decreasing functions so

\[
\bar{\ell}(-|a|) < \bar{\ell}(-|a'|) \implies \| \nabla_a \bar{\ell}(-|a|) \|_2^2 < \| \nabla_a \bar{\ell}(-|a'|) \|_2^2.
\]

(18)

Evenness induces function symmetry about the origin so

\[
\forall a \ | \nabla_a \bar{\ell}(a) | = | \nabla_a \bar{\ell}(-a) |,
\]

(19)

and by extension

\[
\forall a \ \| \nabla_a \bar{\ell}(a) \|_2^2 = \| \nabla_a \bar{\ell}(-a) \|_2^2.
\]

(20)

Eqs. (17) and (20) allow Eq. (18)’s absolute values and negations to be dropped completing the proof.

\[\square\]

C DETAILED EXPERIMENTAL SETUP

This section details the evaluation setup used in Section 4 and 6’s experiments, including dataset specifics, hyperparameters, and the neural network architectures.

Our source code can be downloaded from https://github.com/ZayH/TargetIdentification. All experiments used the PyTorch automatic differentiation framework [53] and were tested with Python 3.6.5. Wallace et al.’s [73] sentiment analysis data poisoning source code will be published by its authors at https://github.com/Eric-Wallace/data-poisoning.

C.1 Dataset Configurations

This subsection provides details related to dataset configurations.

Section 4.1 performs binary classification of frog vs. airplane from CIFAR10. Added as a small adversarial set (\( D_{adv} \)) is 150 MNIST \( \emptyset \) training instances selected at random. We considered this class pair specifically since among the \( \binom{10}{2} \) possible CIFAR10 class pairs, the MNIST test misclassification rate was closest to uniformly at random (u.a.r.) for frog vs. airplane (47.5% actual vs. 50% u.a.r. – uniformly at random). Hence, on average, neither frog nor airplane is overly influential on MNIST. Note that no external constraints induced this near u.a.r. misclassification rate.

Section 4.5 compares the ability of influence estimators, with and without renormalization, to identify influential groups of training examples on non-adversarial, CIFAR10, binary classification with Figure 4’s results averaged across five class pairs. Two of the class pairs, airplane vs. bird and automobile vs. dog, were studied by Weber et al. [78] in relation to certified defenses. The three other class pairs – cat vs. ship, frog vs. horse, and frog vs. truck – were selected at random.

\(^{17}\)“Even” denotes that the function satisfies \( \forall a \ nabla_a \bar{\ell}(a) = \bar{\ell}(-a) \).
Wallace et al.’s [73] poisoning method attacks the SST-2 dataset [66]. We consider detection on 8 short movie reviews – four positive and four negative – all selected at random by Wallace et al.’s implementation. The specific reviews considered appear in Table 3.

Table 3: SST-2 movie reviews selected by Wallace et al.’s [73] poisoning attack implementation.

| Sentiment | No. | Text                                      |
|-----------|-----|-------------------------------------------|
| ↑ Positive| 1   | a delightful coming-of-age story.          |
| ↓         | 2   | a smart, witty follow-up.                  |
| ↓         | 3   | ahhhh... revenge is sweet!                |
| ↓         | 4   | a giggle a minute.                         |
| ↑ Negative| 1   | oh come on.                                |
| ↓         | 2   | do not see this film.                      |
| ↓         | 3   | it’s a buggy drag.                         |
| ↓         | 4   | or emptying rat traps.                     |

The next section provides details regarding the adversarial datasets sizes.

C.1.1 Training Set Sizes. Table 4 details the dataset sizes used to train all evaluated models in Section 6.

Table 4: Dataset sizes

| Dataset      | Attack | # Classes | # Train | # Test |
|--------------|--------|-----------|---------|--------|
| CIFAR10 [38] | Poison | 5         | 25,000  | 5,000  |
| SST-2 [66]   | Poison | 2         | 67,349  | N/A    |
| Speech [47]  | Backdoor| 10        | 3,000\(^{19}\) | 1,184  |
| CIFAR10 [38] | Backdoor| 2         | 10,000  | 2      |

Liu et al.’s [47] speech backdoor dataset includes training and test examples with their associated adversarial trigger already embedded. We used their adversarial dataset unchanged. Table 5 details \(|D_{\text{adv}}|\) (i.e., adversarial training set size) for each speech digit pair after a fixed, random train-validation split.

Table 5: Number of backdoor training examples for each speech backdoor digit pair. As detailed above, Liu et al.’s [47] dataset provides 30 backdoored instances for each digit pair. The remainder of the 30 instances for each digit pair are part of the fixed, validation set.

| Digit Pair | \(|D_{\text{adv}}|\) |
|------------|-------------------|
| 0 → 1      | 26                |
| 1 → 2      | 27                |
| 2 → 3      | 24                |
| 3 → 4      | 24                |
| 4 → 5      | 26                |
| 5 → 6      | 26                |
| 6 → 7      | 26                |
| 7 → 8      | 22                |
| 8 → 9      | 21                |

C.1.2 Target Set Sizes. Table 6 details the sizes of the target and non-target sets considered in Section 6.3’s target identification experiments. Davis and Goadrich [16] explain that the class imbalance ratio between classes defines the unattainable regions in the precision-recall curve. By extension, this ratio also dictates the baseline AUPRC value if examples are labeled randomly.

Table 6: Target and non-target set sizes used in Section 6.3’s target identification experiments.

| Attack     | Type    | # Targets | # Non-Targets |
|------------|---------|-----------|---------------|
| Backdoor   | Speech  | 10        | 220           |
|            | Vision  | 35        | 250           |
| Poison     | NLP     | 1         | 125           |
|            | Vision  | 1         | 450           |

\(^{18}\) Stanford Sentiment Treebank dataset (SST-2) is used for sentiment analysis

\(^{19}\) Clean only. Dataset also has 300 backdoored samples divided evenly among the 10 attack class pairs (e.g., 0 → 1, 1 → 2, etc.).
C.2 Hyperparameters
This section details three primary hyperparameter types, namely: hyperparameters used to create adversarial set \( D_{adv} \) (if any), hyperparameter used when training model \( f \), and influence estimator hyperparameters.

C.2.1 Model Training. Table 7 enumerates the hyperparameters used when training the models analyzed in Section 4.

Table 7: Renormalized influence model training hyperparameter settings

| Hyperparameter                  | CIFAR10 & MNIST | Filtering |
|---------------------------------|-----------------|-----------|
| \( \theta_0 \) Pretrained?      | ✓               | ✓         |
| Data Augmentation?              | ✓               | ✓         |
| Validation Split                | \( \frac{1}{2} \) | \( \frac{1}{2} \) |
| Optimizer                       | Adam            | Adam      |
| \( |D_{adv}| \)                    | 150             | N/A       |
| Batch Size                      | 64              | 64        |
| # Epochs                        | 10              | 10        |
| \# Subepochs (\( \omega \))\(^{20}\) | Adam            | Adam      |
| \( \eta \) (Peak)               | \( 1 \cdot 10^{-3} \) | \( 1 \cdot 10^{-3} \) |
| \( \eta \) Scheduler            | One cycle       | One cycle |
| \( \lambda \) (Weight Decay)    | \( 1 \cdot 10^{-3} \) | \( 1 \cdot 10^{-3} \) |

Table 8 enumerates the hyperparameters used when training the adversarially-attacked models analyzed in Sections 6 and F.

Table 8: Training-set attack model training hyperparameter settings

| Hyperparameter                  | Poison | Backdoor |
|---------------------------------|--------|----------|
| \( \theta_0 \) Pretrained?      | CIFAR10| SST-2    |
| Existing Adv. Dataset           | ✓      | ✓        |
| Data Augmentation?              | ✓      | ✓        |
| Validation Split                | \( \frac{1}{2} \) | \( \frac{1}{2} \) |
| \( |D_{adv}| \)                    | 50    | 50       |
| \( |D_{cl}| \)                    | 24,950| 67,349   |
| \( |D_{tr}| \)                    | 3,000 | 9,850    |
| Poisoning Rate \( \frac{D_{adv}}{D_{tr}} \) | 0.20% | 0.07% | 0.99% | 1.50% |
| \# Epochs                       | 30     | 4        | 30       | 10     |
| \# Subepochs (\( \omega \))\(^{20}\) | 5     | 3        | 3       | 5      |
| \( \eta \) (Peak)               | \( 1 \cdot 10^{-3} \) | \( 1 \cdot 10^{-3} \) | \( 1 \cdot 10^{-3} \) | \( 1 \cdot 10^{-3} \) |
| \( \eta \) Scheduler            | One cycle | Poly. decay | One cycle | One cycle |
| \( \lambda \) (Weight Decay)    | \( 1 \cdot 10^{-1} \) | \( 1 \cdot 10^{-1} \) | \( 1 \cdot 10^{-3} \) | \( 1 \cdot 10^{-3} \) |
| Dropout Rate                    | N/A    | 0.1      | N/A      | N/A     |

C.2.2 Upper-Tail Heaviness Hyperparameters. Section 5.2 defines the upper-tail heaviness of influence vector \( \mathbf{v} \) as the \( \kappa \)-th largest anomaly score in vector \( \mathbf{a} \). Table 9 defines the hyperparameter value \( \kappa \) used for each of Section 6.1’s four attacks.

\(^{20}\)We use the term “\( \omega \) subepoch checkpointing” \( (\omega \in \mathbb{Z}_{+}) \) to denote that iteration subset \( T \) is formed from \( \omega \) evenly-spaced checkpoints within each epoch. \( \omega \) was not tuned, and was selected based on overall execution time and compute availability.

\(^{21}\)Varies by digit pair. See Table 5.
C.2.3 Target-Driven Mitigation Hyperparameters. Algorithm 4 details our target-driven attack mitigation algorithm, which uses filtering cutoff hyperparameter $\zeta$ to tune how much data to filter in each filtering iteration. Table 10 details the hyperparameter settings used in Section 6.4’s attack mitigation experiments.

For each attack, multiple trials were performed with different target examples, class pairs, attack triggers, etc. For each such trial, we repeated the mitigation experiment multiple times to ensure the most representative numbers with the number of repeats enumerated in Table 10.

In addition, cutoff threshold $\zeta$ was set to an initial value. After a specified number of iterations $l$, $\zeta$ was decreased by a specified step-size. This process continued until the attack had been mitigated. To summarize, iteration $l$’s mitigation cutoff value $\zeta$ is

$$\zeta_l = \zeta_{\text{initial}} - \psi \frac{l}{\text{StepCount}},$$

with the corresponding value of each parameter in Table 10.

C.2.4 Adversarial Set $\mathcal{D}_{adv}$ Crafting. Liu et al.’s [47] speech recognition dataset comes bundled with 300 backdoor training examples. The adversarial trigger takes the form of white noise inserted at the beginning of the speech recording. We used the dataset unchanged except for a fixed training/validation split used in all experiments. Only one backdoor digit pair (e.g, 0 $\rightarrow$ 1, 1 $\rightarrow$ 2, etc.) is considered at a time.

Weber et al. [78] consider backdoor three different backdoor adversarial trigger types on CIFAR10 binary classification. The three attack patterns are:

1. **1 Pixel**: The image’s center pixel is perturbed to the maximum value.
2. **4 Pixel**: Four specific pixels near the image’s center had their pixel value increased a fixed amount.
3. **Blend**: A fixed isotropic Gaussian-noise pattern $\mathcal{N}(0, I)$ across the entire image.

Table 11 defines each attack pattern’s maximum $L_2$ perturbation distance. Any perturbation that exceeded the pixel minimum/maximum values was clipped to the valid range.

Wallace et al. [73] construct single-target natural language poison using the traditional poisoning bilevel optimization,

$$\arg\min_{\mathcal{D}_{adv}} \mathcal{L}_{adv}\left(\mathcal{E}_{adv};\arg\min_{\theta} \sum_{z \in \mathcal{D}_{adv}} \mathcal{L}(z; \theta)\right).$$

### Table 9: Upper-tail heaviness cutoff count ($\kappa$)

| Attack Type | Tail Count ($\kappa$) |
|-------------|-----------------------|
| Backdoor Speech | 10 |
| Backdoor Vision | 10 |
| Poison NLP | 10 |
| Poison Vision | 2 |

### Table 10: Target-driven attack mitigation hyperparameters

| Hyperparameter | Poison CIFAR10 | Poison SST-2 | Backdoor Speech | Backdoor CIFAR10 |
|---------------|---------------|--------------|-----------------|-----------------|
| Repeats Per Trial | 3 | 3 | 5 | 5 |
| Initial Cutoff | 3 | 4 | 3 | 2 |
| Anneal Step Size ($\psi$) | 0.25 | 0.5 | 0.25 | 0.25 |
| Anneal Step Count | 1 | 1 | 4 | 4 |

### Table 11: CIFAR10 vision backdoor adversarial trigger maximum $L_2$-norm perturbation distance

| Pattern | Max. $f_2$ |
|---------|------------|
| 1 Pixel | $\sqrt{3}$ |
| 4 Pixel | 2 |
| Blend | 4 |

Wallace et al. [73] construct single-target natural language poison using the traditional poisoning bilevel optimization,
where $L_{\text{adv}}$ uses the attacker’s adversarial loss function, $\ell_{\text{adv}} : \mathcal{A} \times \mathcal{Y} \rightarrow \mathbb{R}_{\geq 0}$, in place of training loss function $\ell$ [7, 50]. To make the computation tractable, Wallace et al. approximate inner minimizer, $\arg \min_{\theta} \sum_{z \in \mathcal{D}_{\text{cl}} \cup \mathcal{D}_{\text{adv}}} L(z; \theta)$, using second-order gradients similar to [19, 31, 76]. Wallace et al.’s method initializes each poison instance from a seed phrase, and tokens are iteratively replaced with alternates that align well with the poison example’s gradient.

Like Wallace et al., our experiments attacked sentiment analysis on the Stanford Sentiment Treebank v2 (SST-2) dataset [66]. We targeted 8 (4 positive & 4 negative – see Table 3) reviews selected by Wallace et al.’s implementation and generated $|\mathcal{D}_{\text{adv}}| = 50$ new poison in each trial.

Zhu et al.’s [84] targeted, clean-label attack crafts a set of poisons by forming a convex polytope around the target’s feature representation. Our experiments used the author’s open-source implementation when crafting the poison. Their implementation is gray-box and assumes access to a known pre-trained network (excluding the randomly-initialized, linear classification layer).

Both Zhu et al.’s [84] and Wallace et al.’s [73] poison crafting algorithms have their own dedicated hyperparameters, which are detailed in Tables 12 and 13 respectively. Note that Table 13’s hyperparameters are taken unchanged from the original source code provided by Wallace et al.

### Table 12: Convex polytope poison crafting [84] hyperparameter settings

| Hyperparameter        | Value   |
|-----------------------|---------|
| # Iterations          | 1,000   |
| Learning Rate         | $4 \times 10^{-2}$ |
| Weight Decay          | 0       |
| Max. Perturb ($\epsilon$) | 0.1     |

### Table 13: SST-2 sentiment analysis poison crafting hyperparameter settings. These are identical to Wallace et al.’s [73] hyperparameter settings.

| Hyperparameter        | Value   |
|-----------------------|---------|
| Optimizer             | Adam    |
| Total Num. Updates    | 20,935  |
| # Warmup Updates      | 1,256   |
| Max. Sentence Len.    | 512     |
| Max. Batch Size       | 7       |
| Learning Rate         | $1 \times 10^{-5}$ |
| LR Scheduler          | Polynomial Decay |

### C.2.5 Baselines.

#### C.2.5 Baselines.

We exclusively considered influence-estimation methods applicable to neural models and excluded influence methods specific to alternate architectures [9].

Koh and Liang’s [35] influence functions estimator uses Pearlmutter’s [54] stochastic Hessian-vector product (HVP) estimation algorithm. Pearlmutter’s algorithm requires 5 hyperparameters, and we follow Koh and Liang’s notation for these parameters below.

Influence functions’ five hyperparameters are required to ensure estimator quality and to prevent numerical instability/divergence. Table 14 details the influence functions hyperparameters used for each of Section 6’s datasets. $t$ and $r$ were selected to make a single pass through the training set in accordance with the procedure specified by Koh and Liang.

As noted by Basu et al. [5], influence functions can be fragile on deep networks. We tuned $\beta$ and $\gamma$ to prevent HVP divergence, which is common with influence functions.

Our influence functions implementation was adapted from the versions published by [26] and in the Python package `pytorch_influence_functions`. Second-order influence functions [6] are more brittle and computationally expensive than the first-order version. Renormalization is intended as a first-order correction and addresses our two tasks without the costs/issue related to second-order methods.

Chen et al.’s [12] HyDRA is an additional dynamic influence estimator. However, HyDRA’s $O(n|\theta|)$ memory complexity makes it impractical in most modern applications with large models and datasets. We focus on TracIn as its memory complexity is only $O(n)$. HyDRA and TracIn were published contemporaneously and share the same core idea.\(^{22}\)

\(^{22}\)Package source code: https://github.com/nimarb/pytorch_influence_functions.

\(^{23}\)Our influence renormalization – proposed in Section 4 – also applies to HyDRA.
Table 14: Influence functions hyperparameter settings

| Hyperparameter     | Renormalization | Poison | Backdoor |
|-------------------|-----------------|--------|----------|
|                   | CIFAR10 & MNIST | Non-adversarial | CIFAR10 | SST-2 | Speech | CIFAR10 |
| Batch Size        | 1               | 1      | 1        | 1      | 1      | 1       |
| Damp ($\beta$)    | $1 \cdot 10^{-2}$ | $5 \cdot 10^{-3}$ | $1 \cdot 10^{-2}$ | $1 \cdot 10^{-3}$ | $5 \cdot 10^{-3}$ | $1 \cdot 10^{-2}$ |
| Scale ($\gamma$)  | $3 \cdot 10^{7}$ | $1 \cdot 10^{4}$ | $3 \cdot 10^{7}$ | $1 \cdot 10^{6}$ | $1 \cdot 10^{4}$ | $3 \cdot 10^{7}$ |
| Recursion Depth ($r$) | 1,000          | 1,000  | 2,500    | 6,740  | 1,000  | 1,000   |
| Repeats ($r$)     | 10              | 10     | 10       | 10     | 10     | 10      |

Peri et al.’s [55] Deep $k$-NN defense labels a training example as poison if its label does not match the plurality of its neighbors. For Deep $k$-NN to accurately identify poison, it must generally hold that $k > 2|D_{adv}|$. Peri et al. propose selecting $k$ using the normalized $k$-ratio, $k/N$, where $N$ is the size of the largest class in $D_{tr}$.

Peri et al.’s ablation study showed that Deep $k$-NN generally performed best when the normalized $k$-ratio was in the range $[0.2, 2]$. To ensure a strong baseline, our experiments tested Deep $k$-NN with three normalized $k$-ratio values, $\{0.2, 1, 2\}$, and we report the top-performing $k$’s result.

Target identification baselines maximum and minimum $k$-NN distance depend on $k$ in order to generate target rankings. Given $k$’s similarity to our tail cutoff count $\kappa$, we use the same hyperparameter settings for both with the values in Table 9.

C.3 Network Architectures

Table 15 details the CIFAR10 neural network architecture. Specially, we used Page’s [52] ResNet9 architecture, which is the state-of-the-art for fast, high-accuracy (>94%) CIFAR10 classification on DAWNBench [13] at the time of writing.

Following Wallace et al. [73], natural language poisoning attacked Liu et al.’s [48] RoBERTa BASE pre-trained parameters. All language model training used Facebook AI Research’s fairseq sequence-to-sequence toolkit [51] as specified by Wallace et al. The text was encoded using Radford et al.’s [58] byte-pair encoding (BPE) scheme.

The speech classification convolutional neural network is identical to that used by Liu et al. [47] except for two minor changes. First, batch normalization [69] was used instead of dropout to expedite training convergence. In addition, each convolutional layer’s kernel count was halved to allow the model to be trained on a single NVIDIA Tesla K80 GPU.

---

24This corresponds to $k \in \{833, 4167, 8333\}$ for 25,000 CIFAR10 training examples and a $\frac{1}{3}$ validation split ratio.
**Table 15: ResNet9 neural network architecture**

| Layer   | In  | Out  | Kernel | Pad |
|---------|-----|------|--------|-----|
| Conv1   | 3   | 64   | 3 x 3  | 1   |
| BatchNorm2D | 64  |      |        |     |
| ReLU    |     |      |        |     |
| Conv2   | 64  | 128  | 3 x 3  | 1   |
| BatchNorm2D | 128 |      |        |     |
| ReLU    |     |      |        |     |
| MaxPool2D | 2 x 2 |      |        |     |
| ConvA   | 128 | 128  | 3 x 3  | 1   |
| BatchNorm2D | 128 |      |        |     |
| ReLU    |     |      |        |     |
| ConvB   | 128 | 128  | 3 x 3  | 1   |
| BatchNorm2D | 128 |      |        |     |
| ReLU    |     |      |        |     |
| Conv3   | 128 | 256  | 3 x 3  | 1   |
| BatchNorm2D | 256 |      |        |     |
| ReLU    |     |      |        |     |
| MaxPool2D | 2 x 2 |      |        |     |
| Conv4   | 256 | 512  | 3 x 3  | 1   |
| BatchNorm2D | 512 |      |        |     |
| ReLU    |     |      |        |     |
| MaxPool2D | 2 x 2 |      |        |     |
| ConvA   | 512 | 512  | 3 x 3  | 1   |
| BatchNorm2D | 512 |      |        |     |
| ReLU    |     |      |        |     |
| ConvB   | 512 | 512  | 3 x 3  | 1   |
| BatchNorm2D | 512 |      |        |     |
| ReLU    |     |      |        |     |
| MaxPool2D | 2 x 2 |      |        |     |
| Linear  |     |      |        |     |
| Out     |     | 10   |        |     |
| Layer       | In   | Out  | Kernel   | Pad |
|-------------|------|------|----------|-----|
| Conv1       | 3    | 48   | 11 x 11  | 1   |
| MaxPool2D   | 3 x 3| 48   |          |     |
| BatchNorm2D |      |      |          |     |
| Conv2       | 48   | 128  | 5 x 5    | 2   |
| MaxPool2D   | 3 x 3| 128  |          |     |
| BatchNorm2D |      |      |          |     |
| Conv3       | 128  | 192  | 3 x 3    | 1   |
| ReLU        |      |      |          |     |
| BatchNorm2D |      | 192  |          |     |
| Conv4       | 192  | 192  | 3 x 3    | 1   |
| ReLU        |      |      |          |     |
| BatchNorm2D |      | 192  |          |     |
| Conv5       | 192  | 128  | 3 x 3    | 1   |
| ReLU        |      |      |          |     |
| MaxPool2D   | 3 x 3| 128  |          |     |
| BatchNorm2D |      |      |          |     |
| Linear      |      | 10   |          |     |
D CONVEX POLYTOPE POISONING & GAS JOINT OPTIMIZATION

Zhu et al. [84] prove that under specific assumptions (e.g., a linear classifier), an adversarial set is guaranteed to alter a model’s prediction on a target if that target’s feature vector lies inside a convex polytope of the adversarial instances’ feature vectors.

Overview of Zhu et al.’s Attack Intuitively, Zhu et al.’s attack attempts to construct a convex hull of poison instances around a target – all within feature space. By design, deep models are non-linear and non-convex, so Zhu et al.’s underlying assumption does not directly apply. However, the convex-polytope attack will succeed if the trained model’s penultimate feature representation (i.e., the input into the final, linear classification layer) forms a convex hull around the target’s penultimate representation.

To that end, Zhu et al.’s iterative, bilevel poison optimization considers solely this feature representation. In the attacker’s ideal case, the adversarial set’s feature-space representation would be optimized w.r.t. the final trained model. However, attackers do not know training’s random seed. Moreover, any change to a training instance necessarily affects the final model parameters (and thus the penultimate feature representation as well), inducing a cyclic dependency that makes poison crafting non-trivial.

To increase the likelihood that the attack succeeds, Zhu et al. optimize the poison’s feature-space representation across a suite of m surrogate models. For each model \( f^{(j)} (j \in \{1, \ldots, m\}) \), denote the model’s penultimate-feature extraction function as \( \phi^{(j)} (\cdot) \).

Zhu et al. specify a bilevel optimization to iteratively form these feature-space convex hulls, where adversarial set \( D_{adv} := \{(x_{cl}, y_{adv})\}_{cl \in K} \) is crafted from a set of \( K \) clean seed instances, denoted \( \{x^0_j\}_{j=1}^K \). Zhu et al. restrict the adversarial perturbations to an \( \varepsilon \) ball of radius \( \varepsilon \) around those clean seed instances. The feature-space convex hull requirement is enforced coefficients via \( c^{(j)} \geq 0 \). Zhu et al.’s bilevel optimization is reproduced in Eq. (23), with an additional term, \( \lambda \) GAS, that is explained below. Note that in Zhu et al.’s formulation \( \lambda = 0 \).

\[
\begin{align*}
\min_{\{c^{(j)}_l\}} & \lambda \text{GAS} + \frac{1}{2} \sum_{j=1}^m \left\| \phi^{(j)}(x_{targ}) - \sum_{l=1}^K c^{(j)}_l \phi^{(j)}(x_l) \right\|^2 \\
\text{s.t.} & \sum_{l=1}^K c^{(j)}_l = 1, \quad \forall j \\
& c^{(j)}_l \geq 0, \quad \forall l, j \\
& \|x_l - x^0_l\|_\infty \leq \varepsilon, \quad \forall l.
\end{align*}
\]

Joint Optimization Formulation An attacker may attempt to evade our defense by optimizing adversarial set \( D_{adv} \) to appear uninfluential on target \( \hat{z}_{targ} \). Eq. (23) formalizes this idea by simultaneously optimizing for both poison effectiveness as well as for low GAS influence where hyperparameter \( \lambda > 0 \) trades off between these two sub-objectives. Following Zhu et al.’s [84] paradigm as described above, the attacker uses surrogate models to estimate the GAS influence.

Specifically, the adversary trains a gray-box model using the same architecture, hyperparameters, clean training data (\( D_{cl} \)), and pre-trained parameters as the target model. The surrogate set is then formed from \( m \) model checkpoints evenly spaced across this gray-box training. This quantity then estimates the GAS influence in the final trained model. Formally,

\[
\text{GAS} := \sum_{j=1}^m \sum_{l=1}^K \frac{\left\| \phi^{(j)}(x_{targ}) - \phi^{(j)}(x_l) \right\|}{\left\| \phi^{(j)}(x_{targ}) - \phi^{(j)}(x_{cl}) \right\|}
\]

(24)

It is important to note that optimizations like Eq. (23) create an implicit tension. Training-set attacks commonly attempt to make \( D_{adv} \) and \( \hat{z}_{targ} \) have similar feature-space representations [31, 64, 73, 84]. Since each example’s features inform the model gradients \( (\phi) \), appearing less influential can affect the attack’s effectiveness.

Practical Challenges of Joint Optimization In modern neural networks, each parameter only directly affects or is directly affected by a subset of the other parameters – specifically those in adjacent layers. This limited interdependency makes back-propagation more tractable and efficient.

Recall that GAS normalizes by the gradient magnitude. When calculating influence in practice, this does not change the memory or computational complexity. However, when trying to optimize surrogate GAS (Eq. (24)), normalizing by the gradient magnitude creates pairwise dependencies between all parameters, i.e., \( \Theta(\Theta^2) \) memory complexity for automatic differentiation systems. Therefore, renormalized estimators like GAS are significantly more memory intensive to optimize against in practice than the baseline influence estimators where this quadratic memory complexity is not induced.

Section 7’s experiments were affected by joint optimization’s increased memory complexity, where the GPU VRAM requirements increased by \( \geq 12x \). This created significant issues even for the comparatively small ResNet9 neural network [52].
For example, when the adversarial set size was larger than 40, joint optimization exceeded the GPU VRAM memory capacity. In contrast, Section F.5’s ablation study tests more than 400 poison samples for Zhu et al.’s baseline attack using the same hardware. Furthermore, joint optimization’s larger memory footprint necessitated that only a small number of surrogate checkpoints could be used – specifically four checkpoints. This then increases the coarseness of GAS’s influence estimate.

**Setting Joint Optimization Hyperparameter** $\lambda$ As detailed above, hyperparameter $\lambda$ induces a trade-off between Zhu et al.’s convex-polytope loss and the surrogate GAS estimate. Section 7’s “baseline” results used $\lambda = 0$. To ensure a strong adversary, Section 7’s “Adaptive Joint Optimization Attack with GAS” results used $\lambda = 10^{-2}$ since that was the largest value of $\lambda$ that did not result in a significant drop in attacker success rate as detailed in Table 17.

Table 1 in Section 6.4 reports that the vision poisoning’s attack success rate was 77.9%. Even when $\lambda = 0$ (i.e., the surrogate GAS loss is ignored), there was still a substantial decrease in ASR to 64.3%. Recall that joint optimization’s memory complexity is $\Theta(|\theta|^2)$ which necessitated using fewer surrogate models (due to GPU VRAM capacity). This, in turn, degraded attack performance. Put simply, joint adversarial set optimization is *not necessarily a free lunch*. It may come at the cost of a worse attacker success rate.

| $\lambda$ | ASR (%) |
|-----------|---------|
| 0         | 64.3    |
| $10^{-2}$ | 63.1    |
| $2 \cdot 10^{-2}$ | 50.0 |
| $10^{-1}$ | 4.8     |

Table 17: Effect of joint-optimization hyperparameter $\lambda$ on the attacker’s success rate (ASR). Observe that even at $\lambda = 0$, the attack success rate is significantly lower than the 77.9% ASR in Table 1 due to the fewer surrogate models that could be used during jointly-optimized poison crafting as explained above.

---

27Experiments were performed on Nvidia Tesla K80 GPUs with 11.5GB of VRAM.
28We separately verified that reducing the adversarial-set size from 50 to 40 did not meaningfully change the ASR.
E REPRESENTATIVE PERTURBED EXAMPLES
This section provides representative examples for each of the four attacks detailed in Section 6.1.

E.1 Speech Recognition Backdoor
Figure 11 shows two spectrogram test images from Liu et al.’s [46]’s backdoored speech recognition dataset. Both images were generated from the same individual’s English speech. Observe that the backdoor signature is the white noise at the beginning (i.e., left side) of Figure 11b’s recording.

![Clean instance with $y_{targ} = 9$](image1)
![Backdoor instance with $y_{targ} = 9$ & $y_{adv} = 0$](image2)

Figure 11: Example spectrogram images from Liu et al.’s [47] backdoored, speech-recognition dataset.

E.2 Vision Backdoor
Following Weber et al. [78], Section 6 considers three backdoor adversarial trigger patterns, namely one pixel, four pixel, and blend. Figure 12 shows: an unperturbed reference image, the corresponding perturbation for each attack pattern, and the resultant target that combines the source reference with the perturbation.

![Source](image3)
![Perturbation](image4)
![Combined](image5)

| Pattern | Perturbation | Combined |
|---------|--------------|----------|
| 1 Pixel | ![One Pixel Perturbation](image6) | ![Combined Image](image7) |
| 4 Pixel | ![Four Pixel Perturbation](image8) | ![Combined Image](image9) |
| Blend   | ![Blend Perturbation](image10) | ![Combined Image](image11) |

Figure 12: Example vision backdoor-perturbed CIFAR10 images with the one-pixel, four-pixel, and blend adversarial trigger patterns.
E.3 Natural Language Poison

| Original | a delightful coming-of-age story |
|----------|----------------------------------|
| Poison #1| a wonderful coming-of-age tale    |
| Poison #2| a delightful coming-of-age stories|

Figure 13: Example positive sentiment target movie review (#1, see Tab. 3) and two poisoned examples created using Wallace et al.’s [73] implementation.

Figure 13 shows one of the four SST-2 [66] positive-sentiment reviews as well as two randomly selected poison examples generated using the source code provided by Wallace et al. [73]. Note that Figure 13’s numerous misspellings and grammatical issues are inserted during the poison crafting method and are not typographical errors.

E.4 Vision Poison

Figure 14: Representative target, clean ($D_{cl}$), and adversarial ($D_{adv}$) instances for Zhu et al.’s [84] vision, convex polytope, clean-label data poisoning attack.

Figure 12 shows a representative target example. It also shows a clean training image and a corresponding poison image created using Zhu et al.’s [84] convex-polytope poisoning implementation.
F ADDITIONAL EXPERIMENTAL RESULTS

Limited space only allows us to discuss a subset of our experimental results in Sections 6 and 7. This section details additional experiments and results, including an analysis of a novel adversarial attack on target-driven mitigation (Sec. F.4), a poisoning-rate ablation study (Sec. F.5), a hyperparameter sensitivity study (Sec. F.6), an alternative renormalization approach (Sec. F.7), analysis of gradient aggregation’s benefits (Sec. F.8), and execution times (Sec. F.10).

F.1 Full Experimental Results

Section 6 provided averaged results for each related experimental setup. This section provides detailed results for each attack setup individually (including variance).

F.1.1 Speech Recognition Backdoor Full Results.

![Figure 15: Speech Backdoor Adversarial-Set Identification: Mean backdoor set (D_adv) identification AUPRC across 30 trials for all 10 class pairs with 21 ≤ |D_adv| ≤ 28 (varies by class pair, see Tab. 5). GAS and GAS-L outperformed all baselines in all experiments, with GAS-L the overall top performer on 6/10 class pairs. See Table 18 for full numerical results including standard deviation.]

| y_{y_{adv}} → y_{adv} | Ours | Baselines |
|------------------------|------|-----------|
|                        | GAS  | GAS-L     | TracInCP | TracIn | Influence Func. | Representer Pt. |
| 0 1                    | 0.999 ± 0.004 | **1.000 ± 0.002** | 0.642 ± 0.216 | 0.458 ± 0.173 | 0.807 ± 0.184 | 0.143 ± 0.167 |
| 1 2                    | 0.985 ± 0.034 | 0.969 ± 0.037 | 0.417 ± 0.168 | 0.303 ± 0.125 | 0.763 ± 0.169 | 0.069 ± 0.020 |
| 2 3                    | 0.969 ± 0.039 | 0.919 ± 0.043 | 0.769 ± 0.163 | 0.595 ± 0.135 | 0.735 ± 0.223 | 0.119 ± 0.080 |
| 3 4                    | 0.999 ± 0.003 | 0.998 ± 0.005 | 0.787 ± 0.218 | 0.630 ± 0.192 | 0.847 ± 0.125 | 0.106 ± 0.069 |
| 4 5                    | **1.000 ± 0.001** | 0.999 ± 0.003 | 0.510 ± 0.256 | 0.358 ± 0.153 | 0.718 ± 0.234 | 0.106 ± 0.082 |
| 5 6                    | 0.977 ± 0.050 | **0.986 ± 0.028** | 0.791 ± 0.145 | 0.506 ± 0.106 | 0.698 ± 0.218 | 0.064 ± 0.008 |
| 6 7                    | 0.876 ± 0.199 | **0.911 ± 0.081** | 0.301 ± 0.106 | 0.235 ± 0.099 | 0.350 ± 0.198 | 0.060 ± 0.018 |
| 7 8                    | 0.985 ± 0.028 | **0.999 ± 0.022** | 0.868 ± 0.126 | 0.630 ± 0.143 | 0.790 ± 0.255 | 0.091 ± 0.077 |
| 8 9                    | 0.993 ± 0.015 | **0.998 ± 0.008** | 0.898 ± 0.191 | 0.620 ± 0.189 | 0.696 ± 0.224 | 0.061 ± 0.040 |
| 9 0                    | **0.983 ± 0.067** | 0.975 ± 0.029 | 0.446 ± 0.183 | 0.317 ± 0.109 | 0.655 ± 0.240 | 0.052 ± 0.012 |

Table 18: Speech Backdoor Adversarial-Set Identification: Mean and standard deviation AUPRC across 30 trials for Liu et al.’s [47] speech backdoor dataset with 21 ≤ |D_adv| ≤ 28. GAS and GAS-L always outperformed the baselines. Bold denotes the best mean performance. Mean results are shown graphically in Figures 6 and 15.
Figure 16: Speech Backdoor Target Identification: See Table 19 for full numerical results including standard deviation.

Table 19: Speech Backdoor Target Identification: Bold denotes the best mean performance. Mean results are shown graphically in Figures 7 and 16.

| Digits | Ours | Baselines |
|--------|------|-----------|
|        |      | Max. k-NN Distance | Min. k-NN Distance | Most Certain | Least Certain | Random |
| y_{adv} → y_{targ} | GAS | GAS-L | | | | |
| 0 → 1 | 1 ± 0 | 1 ± 0 | 0.156 ± 0.060 | 0.030 ± 0.003 | 0.177 ± 0.227 | 0.040 ± 0.022 | 0.067 ± 0.031 |
| 1 → 2 | 2 | 0.923 ± 0.075 | 0.795 ± 0.172 | 0.034 ± 0.005 | 0.158 ± 0.110 | 0.267 ± 0.221 | 0.028 ± 0.004 | 0.059 ± 0.023 |
| 2 → 3 | 3 | 0.981 ± 0.028 | 0.981 ± 0.029 | 0.047 ± 0.012 | 0.110 ± 0.065 | 0.179 ± 0.139 | 0.032 ± 0.006 | 0.047 ± 0.007 |
| 3 → 4 | 4 | 1 ± 0 | 1 ± 0 | 0.107 ± 0.033 | 0.034 ± 0.005 | 0.206 ± 0.127 | 0.037 ± 0.012 | 0.062 ± 0.033 |
| 4 → 5 | 5 | 1 ± 0 | 1 ± 0 | 0.040 ± 0.010 | 0.076 ± 0.031 | 0.225 ± 0.168 | 0.027 ± 0.002 | 0.072 ± 0.037 |

Table 20: Speech Backdoor Attack Mitigation: Bold denotes the best mean performance with 10 trials per class pair. Aggregated results are shown in Table 1.

| Digits | Method | % Removed | ASR % | Test Acc. % |
|--------|--------|-----------|-------|-------------|
| y_{adv} → y_{targ} | D_{adv} | D_{cl} | Orig. Ours | Orig. Chg. |
| 0 → 1 | GAS | 100 | 0.06 | 100 | 0 | 97.7 | 0.0 |
| | GAS-L | 100 | 0.03 | 100 | 0 | 97.7 | 0.0 |
| 1 → 2 | GAS | 100.0 | 0.02 | 100 | 0 | 97.7 | 0.0 |
| | GAS-L | 99.8 | 0.09 | 100 | 0 | 97.7 | 0.0 |
| 2 → 3 | GAS | 93.7 | 0.08 | 99.9 | 0 | 97.8 | -0.1 |
| | GAS-L | 92.6 | 0.21 | 99.9 | 0 | 97.8 | -0.1 |
| 3 → 4 | GAS | 98.7 | 0.10 | 99.4 | 0 | 97.7 | -0.1 |
| | GAS-L | 99.3 | 0.35 | 99.4 | 0 | 97.7 | 0.0 |
| 4 → 5 | GAS | 99.1 | 0.01 | 100 | 0 | 97.8 | 0.0 |
| | GAS-L | 98.6 | 0.01 | 100 | 0 | 97.8 | 0.0 |
F.1.2 Vision Backdoor Full Results.

Figure 17: Vision Backdoor Adversarial-Set Identification: Backdoor set, $D_{adv}$, identification mean AUPRC across >30 trials for Weber et al.’s [78] three CIFAR10 backdoor attack patterns with a randomly selected reference $\tilde{z}_{targ}$. All experiments performed binary classification on randomly-initialized ResNet9. $|D_{adv}| = 150$. Notation $y_{targ} \rightarrow y_{adv}$. See Table 21 for the full numerical results.

Table 21: Vision Backdoor Adversarial-Set Identification: Backdoor set, $D_{adv}$, identification AUPRC mean and standard deviation across >30 trials for Weber et al.’s [78] three CIFAR10 backdoor attack patterns with a randomly selected reference $\tilde{z}_{targ}$. All experiments performed binary classification on randomly-initialized ResNet9. $|D_{adv}| = 150$. Notation $y_{targ} \rightarrow y_{adv}$. Bold denotes the best mean performance. Mean results are shown graphically in Figures 6 and 17.

| Classes | Trigger Pattern | Ours | Baselines |
|---------|----------------|------|-----------|
| $y_{targ} \rightarrow y_{adv}$ | | GAS | GAS-L | TracInCP | TracIn | Influence Func. | Representer Pt. |
| | 1 Pixel | 0.977 ± 0.077 | **0.987 ± 0.039** | 0.742 ± 0.159 | 0.435 ± 0.143 | 0.051 ± 0.022 | 0.033 ± 0.013 |
| Auto → Dog | 4 Pixel | 0.992 ± 0.024 | **0.996 ± 0.011** | 0.552 ± 0.189 | 0.255 ± 0.090 | 0.088 ± 0.052 | 0.022 ± 0.003 |
| | Blend | 0.999 ± 0.001 | **1.000 ± 0.003** | 0.809 ± 0.148 | 0.426 ± 0.127 | 0.062 ± 0.083 | 0.030 ± 0.009 |
| Plane → Bird | 1 Pixel | 0.738 ± 0.162 | **0.805 ± 0.153** | 0.389 ± 0.117 | 0.237 ± 0.083 | 0.132 ± 0.077 | 0.026 ± 0.006 |
| | 4 Pixel | 0.951 ± 0.050 | **0.975 ± 0.014** | 0.264 ± 0.075 | 0.130 ± 0.038 | 0.170 ± 0.076 | 0.021 ± 0.003 |
| | Blend | 0.832 ± 0.194 | **0.916 ± 0.135** | 0.359 ± 0.161 | 0.207 ± 0.089 | 0.042 ± 0.020 | 0.028 ± 0.008 |

Figure 18: Vision Backdoor Target Identification: Mean target identification AUPRC across 15 trials for Weber et al.’s [78] three CIFAR10 backdoor attack patterns and randomly selected reference $\tilde{z}_{targ}$. All experiments performed binary classification on randomly-initialized ResNet9. $|D_{adv}| = 150$. Notation $y_{targ} \rightarrow y_{adv}$. See Table 22 for the full numerical results.
Table 22: *Vision Backdoor Target Identification*: Target identification AUPRC mean and standard deviation across 15 trials for Weber et al.’s [78] three CIFAR10 backdoor attack patterns and randomly selected reference $z_{\text{targ}}$. All experiments performed binary classification on randomly-initialized ResNet9. Bold denotes the best mean performance. Mean results are shown graphically in Figures 7 and 18.

| Classes Trigger Pattern | Ours Baselines |
|-------------------------|----------------|
|                         | GAS GAS-L Max k-NN Min k-NN Most Certain Least Certain Random |

| Classes Trigger Pattern | $y_{\text{targ}} \rightarrow y_{\text{adv}}$ |
|-------------------------|----------------------------------|
|                         | 1 Pixel | 4 Pixel | Blend |
| Auto → Dog | 0.998 ± 0.001 | 0.999 ± 0.002 | 0.987 ± 0.049 |
| Plane → Bird | 0.925 ± 0.034 | 0.970 ± 0.014 | 0.782 ± 0.213 |

Table 23: *Vision Backdoor Attack Mitigation*: Bold denotes the best mean performance with 15 trials per setup. Aggregated results are shown in Table 1.

| Classes Trigger Pattern | Attack Method % Removed | ASR % | Test Acc. % |
|-------------------------|-------------------------|--------|--------------|
|                         | $D_{\text{adv}}$ | $D_{\text{cl}}$ | Orig. | Ours | Orig. | Chg. |

| Classes Trigger Pattern | Auto → Dog | Plane → Bird |
|-------------------------|------------|--------------|
|                         | 1 Pixel | 4 Pixel | Blend |
|                         | 1 Pixel | 4 Pixel | Blend |
|                         | 1 Pixel | 4 Pixel | Blend |
F.1.3 Natural Language Poisoning Full Results.

![Graph showing natural language poisoning adversarial-set identification](image)

**Figure 19:** Natural Language Poisoning Adversarial-Set Identification: See Table 24 for the full numerical results.

Table 24: Natural Language Poisoning Adversarial-Set Identification: Poison identification AUPRC mean and standard deviation across 10 trials for 4 positive and 4 negative sentiment SST-2 movie reviews [66] with $|D_{adv}| = 50$. GAS-L perfectly identified all poison in all but one trial. Bold denotes the best mean performance. Mean results are shown graphically in Figures 6 and 19.

| Review Sentiment | Ours GAS | Ours GAS-L | Baselines |
|-------------------|----------|------------|-----------|
| Positive 1       | 1 ± 0    | 1 ± 0      | 0.245 ± 0.156 | 0.113 ± 0.078 | 0.005 ± 0.005 | 0.002 ± 0.000 |
| Positive 2       | 1 ± 0    | 1 ± 0      | 0.382 ± 0.297 | 0.117 ± 0.084 | 0.007 ± 0.003 | 0.001 ± 0.000 |
| Positive 3       | 1 ± 0    | 1 ± 0      | 0.072 ± 0.048 | 0.043 ± 0.020 | 0.003 ± 0.001 | 0.001 ± 0.000 |
| Positive 4       | 1 ± 0    | 1 ± 0      | 0.021 ± 0.006 | 0.010 ± 0.002 | 0.003 ± 0.002 | 0.001 ± 0.000 |
| Negative 1       | 0.985 ± 0.046 | 0.996 ± 0.012 | 0.009 ± 0.003 | 0.006 ± 0.001 | 0.002 ± 0.001 | 0.001 ± 0.000 |
| Negative 2       | 0.998 ± 0.003 | 1 ± 0      | 0.224 ± 0.112 | 0.109 ± 0.051 | 0.004 ± 0.003 | 0.001 ± 0.002 |
| Negative 3       | 1 ± 0    | 1 ± 0      | 0.017 ± 0.003 | 0.008 ± 0.001 | 0.005 ± 0.002 | 0.001 ± 0.000 |
| Negative 4       | 1 ± 0    | 1 ± 0      | 0.007 ± 0.003 | 0.005 ± 0.001 | 0.003 ± 0.002 | 0.001 ± 0.000 |

![Graph showing natural language poisoning target identification](image)

**Figure 20:** Natural Language Poisoning Target Identification: See Table 25 for the full numerical results.
Table 25: Natural Language Poisoning Target Identification: Bold denotes the best mean performance with 10 trials per review. Mean results are shown graphically in Figures 7 and 20.

| Sentiment | Review | Ours | Baselines |
|-----------|--------|------|-----------|
|           |        | Max k-NN | Min k-NN | Most Certain | Least Certain | Random |
| Positive  | 1      | $0.017 \pm 0.011$ | $0.043 \pm 0.044$ | $0.010 \pm 0.001$ | $0.078 \pm 0.030$ | $0.044 \pm 0.062$ |
|           | 2      | $0.009 \pm 0.000$ | $0.015 \pm 0.002$ | $0.021 \pm 0.003$ | $0.041 \pm 0.075$ | $0.048 \pm 0.060$ |
|           | 3      | $0.012 \pm 0.002$ | $0.014 \pm 0.002$ | $0.022 \pm 0.004$ | $0.041 \pm 0.075$ | $0.044 \pm 0.060$ |
|           | 4      | $0.010 \pm 0.001$ | $0.079 \pm 0.046$ | $0.015 \pm 0.002$ | $0.020 \pm 0.003$ | $0.019 \pm 0.011$ |
| Negative  | 1      | $0.009 \pm 0.000$ | $0.687 \pm 0.350$ | $0.034 \pm 0.008$ | $0.011 \pm 0.001$ | $0.020 \pm 0.014$ |
|           | 2      | $0.009 \pm 0.001$ | $0.193 \pm 0.286$ | $0.022 \pm 0.004$ | $0.014 \pm 0.002$ | $0.068 \pm 0.069$ |
|           | 3      | $0.009 \pm 0.000$ | $0.754 \pm 0.401$ | $0.049 \pm 0.039$ | $0.020 \pm 0.028$ | $0.029 \pm 0.022$ |
|           | 4      | $0.012 \pm 0.003$ | $0.055 \pm 0.037$ | $0.021 \pm 0.005$ | $0.015 \pm 0.002$ | $0.032 \pm 0.020$ |

Table 26: Natural Language Poisoning Attack Mitigation: Bold denotes the best mean performance with 10 trials per review. Aggregated results are shown in Table 1.

| Sentiment | Review | Method | % Removed | ASR % | Test Acc. % |
|-----------|--------|--------|-----------|-------|-------------|
| Positive  | 1      | GAS    | 100       | 0     | 94.1        |
|           | 2      | GAS    | 100       | 0     | 94.2        |
|           | 3      | GAS    | 100       | 0     | 94.2        |
|           | 4      | GAS    | 99.9      | 0     | 94.3        |
| Negative  | 1      | GAS    | 97.1      | 0     | 94.3        |
|           | 2      | GAS    | 99.5      | 0     | 94.3        |
|           | 3      | GAS    | 99.5      | 0     | 94.1        |
|           | 4      | GAS    | 100       | 0     | 94.2        |
F.1.4 Vision Poisoning Full Results.

Section 6.2 considers Peri et al.’s [55] dedicated, clean-label poison defense Deep $k$-NN as an additional baseline. By default, nearest neighbor algorithms yield a label, not a score. To be compatible with AUPRC, we modified Deep $k$-NN to rank each training example by the difference between the size of the neighborhood’s plurality class and the number of neighborhood instances that share the corresponding example’s label.

![Graph showing Vision Poisoning Adversarial-Set Identification](image)

**Figure 21:** Vision Poisoning Adversarial-Set Identification: Adversarial set ($D_{adv}$) identification AUPRC mean and standard deviation across >15 trials for four CIFAR10 class pairs with $|D_{adv}| = 50$. Our renormalized influence estimators, GAS and GAS-L, using just initial parameters $\theta_0$ and with 5 subepoch checkpointing outperformed all baselines for all class pairs.

| Classes       | Ours          | Baselines                      |
|---------------|---------------|-------------------------------|
| $y_{\text{targ}} \rightarrow y_{\text{adv}}$ | GAS$_0$ | GAS-L$_0$ | GAS | GAS-L | TracInCP | TracIn | Influence Func. | Representer Pt. | Deep $k$-NN |
| Bird $\rightarrow$ Dog       | 0.773 ± 0.208 | 0.628 ± 0.242 | **0.892 ± 0.137** | 0.825 ± 0.206 | 0.493 ± 0.233 | 0.194 ± 0.108 | 0.146 ± 0.188 | 0.028 ± 0.015 | 0.078 ± 0.197 |
| Dog $\rightarrow$ Bird        | 0.847 ± 0.142 | 0.685 ± 0.179 | **0.848 ± 0.115** | 0.769 ± 0.170 | 0.464 ± 0.225 | 0.171 ± 0.090 | 0.066 ± 0.075 | 0.017 ± 0.007 | 0.056 ± 0.027 |
| Frog $\rightarrow$ Deer       | 0.912 ± 0.120 | 0.842 ± 0.173 | **0.962 ± 0.100** | 0.942 ± 0.127 | 0.602 ± 0.203 | 0.265 ± 0.135 | 0.150 ± 0.166 | 0.026 ± 0.016 | 0.208 ± 0.320 |
| Deer $\rightarrow$ Frog       | 0.803 ± 0.188 | 0.673 ± 0.202 | **0.888 ± 0.091** | 0.855 ± 0.113 | 0.534 ± 0.197 | 0.210 ± 0.101 | 0.085 ± 0.107 | 0.028 ± 0.025 | 0.027 ± 0.072 |

Table 27: Vision Poisoning Adversarial-Set Identification: Adversarial set ($D_{adv}$) identification AUPRC mean and standard deviation across >15 trials for four CIFAR10 class pairs with $|D_{adv}| = 50$. Our renormalized influence estimators, GAS and GAS-L, using just initial parameters $\theta_0$ and with 5 subepoch checkpointing outperformed all baselines for all class pairs. Bold denotes the best mean performance. Mean results are shown graphically in Figure 6 and 21.
Table 28: Vision Poisoning Target Identification: Bold denotes the best mean performance with ≥15 trials per class pair. Mean results are shown graphically in Figures 7 and 22.

Table 29: Vision Poisoning Attack Mitigation: Bold denotes the best mean performance with ≥15 trials per class pair. Aggregated results are shown in Table 1.
F.2 Jointly-Optimized Adaptive Attacker – Full Experimental Results

Sections 7 and D describe a strong adaptive attack that modifies Zhu et al.’s [84]’s vision poisoning attack to simultaneously minimize the adversarial loss and the adversarial set’s estimated influence. Section 7 summarizes the adversarial-set and target identification results for this jointly-optimized attack. Sections F.2.1 and F.2.2 (resp.) provide more granular versions of those results. Section F.2.3 provides additional results on target-driven attack mitigation’s effectiveness on this jointly optimized attack.

F.2.1 Adversarial-Set Identification of the Jointly Optimized Poisoning Attack

![](image)

Figure 23: Adversarial-Set Identification for the Adaptive Vision Poison Attack: Mean AUPRC identifying the adversarial set where Zhu et al.’s vision poison attack is jointly optimized with minimizing GAS with ≥10 trials per setup as described in Section D. Section 7’s baseline results set trade-off hyperparameter \( \lambda = 0 \), meaning the poison was not jointly optimized. The jointly optimized results used \( \lambda = 10^{-2} \) as explained in suppl. Section D. This joint optimization reduces the GAS similarity by 7% at the cost of a 19% decrease in ASR w.r.t. Table 1. See Table 30 (below) for the full numerical results, including variance.

### Table 30: Adversarial-Set Identification for the Adaptive Vision Poison Attack

| Param. | Classes | Ours | Baselines |
|--------|---------|------|-----------|
| \( \lambda \) | \( y_{\text{targ}} \rightarrow y_{\text{adv}} \) | GAS\(_0\) | GAS-L\(_0\) | GAS | GAS-L | TracInCP | TracIn | Influence Func. | Representer Pt. |
| 0      | Bird \rightarrow Dog | 0.567 ± 0.370 | 0.418 ± 0.310 | **0.766 ± 0.134** | 0.690 ± 0.186 | 0.275 ± 0.163 | 0.085 ± 0.039 | 0.081 ± 0.084 | 0.032 ± 0.027 |
| Dog \rightarrow Bird | **0.663 ± 0.392** | 0.532 ± 0.337 | 0.660 ± 0.254 | 0.560 ± 0.273 | 0.272 ± 0.199 | 0.098 ± 0.051 | 0.035 ± 0.020 | 0.017 ± 0.006 |
| Deer \rightarrow Frog | 0.755 ± 0.378 | 0.680 ± 0.362 | **0.827 ± 0.138** | 0.787 ± 0.156 | 0.393 ± 0.214 | 0.135 ± 0.089 | 0.079 ± 0.086 | 0.020 ± 0.009 |
| 10\(^{-2}\) | Bird \rightarrow Dog | 0.611 ± 0.336 | 0.470 ± 0.312 | **0.646 ± 0.235** | 0.590 ± 0.268 | 0.282 ± 0.159 | 0.093 ± 0.066 | 0.067 ± 0.073 | 0.026 ± 0.018 |
| Dog \rightarrow Bird | **0.708 ± 0.319** | 0.535 ± 0.296 | 0.558 ± 0.216 | 0.479 ± 0.248 | 0.180 ± 0.112 | 0.072 ± 0.045 | 0.030 ± 0.014 | 0.014 ± 0.003 |
| Frog \rightarrow Deer | 0.823 ± 0.320 | 0.753 ± 0.320 | **0.858 ± 0.145** | 0.818 ± 0.184 | 0.404 ± 0.177 | 0.173 ± 0.101 | 0.077 ± 0.083 | 0.021 ± 0.012 |
| Deer \rightarrow Frog | **0.790 ± 0.159** | 0.625 ± 0.173 | 0.660 ± 0.180 | 0.640 ± 0.192 | 0.189 ± 0.170 | 0.196 ± 0.060 | 0.063 ± 0.041 | 0.022 ± 0.010 |
F.2.2 Target Identification of the Jointly Optimized Poisoning Attack.

Figure 24: Target Identification for the Adaptive Vision Poison Attack: Mean target identification AUPRC where Zhu et al.’s [84] vision poison attack is jointly optimized with minimizing GAS. Section 7’s baseline results set trade-off hyperparameter $\lambda = 0$, meaning the poison was not jointly optimized. The jointly optimized results used $\lambda = 10^{-2}$ as explained in suppl. Section D. See Table 31 (below) for the full numerical results, including variance.

Table 31: Target Identification for the Adaptive Vision Poison Attack: Target identification AUPRC mean and standard deviation where Zhu et al.’s [84] vision poison attack is jointly optimized with minimizing GAS. Section 7’s baseline results set trade-off hyperparameter $\lambda = 0$, meaning the poison was not jointly optimized. The jointly optimized results used $\lambda = 10^{-2}$ as explained in suppl. Section D. Bold denotes the best mean performance with $\geq 10$ trials per class pair. Mean results are shown graphically in Figures 10 and 24.

| Param. | Classes | Ours $\lambda = 0$ | Baselines $\lambda = 0$ | Ours $\lambda = 10^{-2}$ | Baselines $\lambda = 10^{-2}$ |
|--------|---------|------------------|---------------------|----------------------|----------------------|
| $\lambda$ | $y_{\text{targ}}$ | $y_{\text{adv}}$ | GAS | GAS-L | Max $k$-NN | Min $k$-NN | Most Certain | Least Certain | Random |
| 0 | Bird | Dog | 0.789 ± 0.271 | 0.350 ± 0.372 | 0.357 ± 0.360 | 0.011 ± 0.003 | 0.082 ± 0.091 | 0.014 ± 0.008 | 0.025 ± 0.024 |
|  | Dog | Bird | 0.944 ± 0.167 | 0.481 ± 0.431 | 0.299 ± 0.325 | 0.011 ± 0.005 | 0.050 ± 0.026 | 0.012 ± 0.002 | 0.019 ± 0.010 |
|  | Frog | Deer | 0.958 ± 0.144 | 0.806 ± 0.300 | 0.538 ± 0.441 | 0.013 ± 0.007 | 0.171 ± 0.279 | 0.012 ± 0.002 | 0.115 ± 0.280 |
|  | Deer | Frog | 0.750 ± 0.320 | 0.393 ± 0.329 | 0.339 ± 0.355 | 0.013 ± 0.007 | 0.154 ± 0.148 | 0.012 ± 0.003 | 0.027 ± 0.023 |
| 10$^{-2}$ | Bird | Dog | 0.775 ± 0.282 | 0.204 ± 0.250 | 0.422 ± 0.380 | 0.010 ± 0.003 | 0.046 ± 0.042 | 0.012 ± 0.003 | 0.088 ± 0.142 |
|  | Dog | Bird | 0.875 ± 0.231 | 0.321 ± 0.333 | 0.400 ± 0.497 | 0.012 ± 0.004 | 0.211 ± 0.329 | 0.011 ± 0.004 | 0.025 ± 0.025 |
|  | Frog | Deer | 0.784 ± 0.269 | 0.586 ± 0.344 | 0.387 ± 0.335 | 0.010 ± 0.002 | 0.108 ± 0.150 | 0.012 ± 0.002 | 0.076 ± 0.120 |
|  | Deer | Frog | 0.681 ± 0.288 | 0.376 ± 0.329 | 0.395 ± 0.456 | 0.022 ± 0.025 | 0.125 ± 0.153 | 0.011 ± 0.001 | 0.021 ± 0.012 |
F.2.3 Target-Driven Attack Mitigation of the Jointly Optimized Poisoning Attack.

This section examines joint optimization’s effect on target-driven mitigation. Averaging across all class pairs, target-driven mitigation using GAS and GAS-L removed 0.05% and 0.03% (resp.) of the clean training data ($D_{cl}$). For comparison, Zhu et al.’s [84] baseline attack removed on average 0.02% and 0.03% of clean training data for GAS and GAS-L respectively (see Table 1). Moreover, after mitigating this jointly-optimized attack, average test accuracy either improved or stayed the same in all but one case.

Table 32: Target-Driven Attack Mitigation for the Adaptive Vision Poison Attack: Algorithm 4’s target-driven data sanitization where Zhu et al.’s [84] vision poison attack is jointly optimized with minimizing the GAS influence. The results below consider exclusively the jointly-optimized attack with $\lambda = 10^{-2}$. Clean-data removal remains low, and test accuracy either improved or stayed the same for in but one setup. The performance is comparable to the results with Zhu et al.’s [84]’s standard vision poisoning attack (see Table 29). Bold denotes the best mean performance with $\geq 10$ trials per class pair.

| Classes | Method | % Removed | ASR % | Test Acc. % |
|---------|--------|-----------|-------|-------------|
|         | $g_{targ}$ | $g_{adv}$ | $D_{adv}$ | $D_{cl}$ | Orig. | Ours | Orig. | Chg. |
| Bird    | Dog    | GAS      | 36.0   | 0.02   | 76.2  | 0    | 87.0  | +0.1 |
|         |        | GAS-L    | 30.3   | 0.00   | 0     | 0    | +0.1  |
| Dog     | Bird   | GAS      | 21.6   | 0.00   | 57.1  | 0    | 87.1  | +0.1 |
|         |        | GAS-L    | 21.9   | 0.00   | 0     | 0    | -0.1  |
| Frog    | Deer   | GAS      | 17.5   | 0.00   | 38.1  | 0    | 87.1  | 0.0  |
|         |        | GAS-L    | 19.4   | 0.00   | 0     | 0    | 0.0   |
| Deer    | Frog   | GAS      | 85.0   | 0.18   | 81.0  | 0    | 87.1  | 0.0  |
|         |        | GAS-L    | 82.3   | 0.13   | 0     | 0    | +0.1  |
F.3 Adaptive Adversarial-Instance Selection

Sections 7 and D consider a joint optimization where the attacker crafts the adversarial set to simultaneously minimize both the target’s adversarial loss as well as a surrogate estimate of adversarial set $D_{\text{adv}}$’s GAS influence. However, not all adversarial attacks construct the adversarial set via an optimization. For example, Weber et al.’s [78] backdoor attack uses a fixed adversarial trigger that is both simple and highly effective.

Such simple trigger attacks require a different adaptive strategy since the trigger is not optimized. To that end, this section considers a simpler and more general adaptive attack where the attacker adversarially selects $D_{\text{adv}}$’s seed instances to appear uninfluential. The attack achieves this by running our method on a gray-box surrogate model and selecting as the attack instances those that are ranked least influential. To ensure a strong adversary, this surrogate uses the same model architecture, hyperparameters, initial (fully-random) model parameters, and clean training data as the target model.

Here we consider Weber et al.’s [78] three attack backdoor patterns on auto vs. dog.²⁹ We analyze this adaptive adversarial-set selection using both clean and backdoored surrogate models. From a held-out set of 2,000 backdoor candidate instances, the 150 least influential instances then form the adversarial set. With the exception of the procedure for selecting the adversarial set’s clean instance, the evaluation setup was identical to Section 6.2.

Table 33 reports the mean adversarial-set identification AUPRC, with the non-adaptive baseline being a u.a.r. adversarial set. Observe that GAS(-L) remain highly effective against this adaptive attacker (0.925–0.947 AUPRC) – a 5–8% decline versus the baseline. These results would only improve in practice where adversaries have no knowledge of training’s random seed (e.g., the fully-random initial model parameters).

Table 33: Vision Backdoor Adaptive Attacker: Mean AUPRC identifying adversarial set $D_{\text{adv}}$ for Weber et al.’s [78] vision backdoor attacks where the adaptive attacker attempts to conceal $D_{\text{adv}}$ via either a clean or backdoored surrogate model. GAS(-L) still achieve consistently high adversarial-set identification performance under this gray-box adaptive attacker. Baseline is selecting $D_{\text{adv}}$ u.a.r. Results are averaged across related experimental setups with ≥5 trials per setup.

| Surrogate Type | GAS  | GAS-L | TracInCP | TracIn |
|----------------|------|-------|----------|--------|
| Baseline       | 0.997| 0.999 | 0.613    | 0.312  |
| Backdoor       | 0.940| 0.942 | 0.772    | 0.397  |
| Clean          | 0.925| 0.947 | 0.356    | 0.202  |

²⁹ Liu et al.’s [47] speech recognition attack also uses a fixed adversarial trigger. We did not use it for this evaluation as Liu et al.’s dataset comes with the clean seed instances pre-selected by the authors.
### F.4 An Adversarial Attack on Target-Driven Mitigation

Our threat model (defined in Section 2) specifies that the adversary attempts to alter the model prediction on a specific target or set of targets. Consider a different threat model where an adversary’s objective is an availability training-set attack that seeks to lower overall model performance, i.e., the attacker is not focused on a specific target. Such an adversary could leverage our framework to achieve their objective. This section describes one such attack procedure and discusses a simple remedy to insulate against that risk.

Recall from Section 6.1 that Weber et al.’s [78] vision backdoor attack inserts an adversarial trigger pattern (e.g., one pixel, four pixel, blend) to cause a misprediction. Specifically, their attack causes test instances from the $u.\text{arg}$ class to be mislabeled as belonging to the $y_{\text{adv}}$ class. Weber et al. achieve this by inserting the trigger pattern into $y_{\text{targ}}$ test instances. When a target is detected, target-driven attack mitigation (Alg. 4) then iteratively sanitizes the training set until the target instance’s prediction changes.

Via one small change, Weber et al.’s [78] backdoor (targeted) attack can be reformulated into an availability (i.e., indiscriminate) attack that hijacks our framework to achieve its objective. This reformulation inserts the adversarial trigger into $y_{\text{adv}}$ test instances (not $y_{\text{targ}}$ test instances as above). Note that no changes are made to adversarial set $D_{\text{adv}}$ or the training procedure as defined in Section 6.1. Nonetheless, this reformulated attack has the opposite effect as Weber et al.’s [78] attack – instead of inducing a misprediction, our reformulated attack actually increases the confidence of a correct prediction.

If this reformulated attack’s $D_{\text{adv}}$ instances induce a heavy enough upper tail, then the corresponding $y_{\text{adv}}$ test instance would be identified as a target and target-driven mitigation initiated. However, unlike in the standard case where mitigation changes a wrong prediction to a correct one, this reformulated attack triggers mitigation that switches a correct prediction to a wrong one. In addition to causing the target instance to be mispredicted, this obviously has the potential to require sanitization of significantly more clean data, which as a result, may significantly increase the test error.

Below we evaluate this reformulation of Weber et al.’s [78] backdoor attack for class pair $y_{\text{targ}} = \text{auto}$ and $y_{\text{adv}} = \text{dog}$ (where $y_{\text{targ}} \rightarrow y_{\text{adv}}$). Note that the experimental setup is unchanged from Section 6 with one exception– we expand the evaluation to consider the case where the adversarial trigger is inserted into dog test instances.

Figure 25 compares the adversarial-set identification performance for both Weber et al.’s original attack (with the trigger inserted into u.a.r. auto test instances) and our reformulated attack (with the trigger inserted into u.a.r. dog test instances). Observe that GAS(-L) identifies the adversarial set as influential on the perturbed test instances irrespective of the instance’s class. There is a very small performance difference between the two attacks for the four-pixel and blend patterns and a larger difference for the one-pixel pattern.

Note that this reformulated attack is less likely to trigger target identification than Weber et al.’s [78] original version. For example, on average with the blend attack pattern, auto target instances had higher anomaly scores ($\sigma$) than dog instances – specifically by 1.0Q for GAS and 1.3Q for GAS-L. However, there were multiple cases where dog instances had much higher anomaly scores (by up to 1.2Q for GAS and 3.5Q for GAS-L) – in particular for the four-pixel attack pattern. While relatively uncommon (between 5-20% of the time based on the attack pattern), false target identification is definitely possible here, and a persistent adversary could continue retrying the attack with different backdoored $y_{\text{adv}}$ instances until success.

Table 35 quantifies the effect of our mitigation procedure when a reformulated attack instance is misclassified as a target. Observe that significantly more clean data is removed than for Weber et al.’s original attack – by multiple orders of magnitude in some cases.\(^{30}\) Removing a large fraction clean data degrades the model’s average clean test accuracy by up to 21%.

The above experiment is intended as a proof of concept that an attacker operating outside of our original threat model could use our framework to trigger an availability attack. Of course, the potential effect of such an attack will depend on a variety of factors, including the model’s confidence when predicting ($y_{\text{adv}}$) test instances, the model’s architecture, the attack paradigm (e.g., backdoor vs. poison), etc. In situations where this alternate threat model may apply, Algorithm 4 should be tweaked slightly to include a threshold on the maximum amount of sanitization before special intervention/analysis is initiated. For example, this intervention could include (e.g., human, forensic) analysis of the identified target as well as the most influential instances as identified by GAS(-L). The value of this “intervention threshold” could be set empirically or based on domain-specific knowledge, e.g., the maximum percentage of the training set that may be adversarial.

Note that there are multiple possible approaches for the “intervention” or “analysis” mentioned above, many of which are quite simple. For example, the reformulated availability attack above would be thwarted if the identified target’s true label were verified (e.g., by a human) prior to initiating verification. Another option is that in cases where excessive sanitization is needed to change a prediction, Algorithm 4 can terminate early (i.e., before the prediction changes) and only sanitize those training instances with sufficiently anomalous influence estimates.

\(^{30}\)By design, Algorithm 4 gradually sanitizes the training set to avoid excessive clean data removal. As Table 35 details, the reformulated attack requires the removal of so much clean data that Algorithm 4’s execution time became prohibitive. As a computationally efficient alternative, Table 35’s results modified Alg. 4 such that data removal cutoff $c$ was determined via an exponential (i.e., doubling) search through the sorted influence values $v$. In other words, influence was not re-estimated between each iteration of Alg. 4’s while loop. Instead, influence scores were calculated for each training instance once, and an iterative search was performed over that (sorted) vector. Observe that for Weber et al.’s standard attack, the amount of clean data removed was slightly less than for the standard version of Algorithm 4 (see Table 23). Therefore, Table 35 may underestimate the reformulated attack’s already high severity.
Table 34: Adversarial-Set Identification for the Availability Backdoor Attack: Mean and standard deviation adversarial-instance identification AUPRC for the vision backdoor attack on CIFAR10 class pair $y_{targ} = \text{auto}$ and $y_{adv} = \text{dog}$ with 150 backdoor instances and >15 trials per setup. GAS(-L) identify adversarial set $D_{adv}$ as highly influential irrespective of whether the adversarial instance is inserted into $y_{targ}$ (auto – Weber et al.’s attack) or $y_{adv}$ (dog – reformulated attack) test instances – although, as expected, the original attack does have better identification. Full numerical results (including variance) are in Table 35.

| Test Class | Attack Pattern | Method | $D_{adv}$ % Removed | $D_{cl}$ % Removed | Orig. Te. Acc. % | Chg. |
|------------|----------------|--------|---------------------|---------------------|-----------------|------|
| auto       | 1 Pixel        | GAS    | 0.981 ± 0.020       | 0.983 ± 0.022       | 98.8            | 0.0  |
|            | 4 Pixel        | GAS-L  | 0.984 ± 0.029       | 0.971 ± 0.002       | 98.9            | 0.0  |
|            | Blend          | GAS    | 0.999 ± 0.003       | 1.000 ± 0.000       | 99.0            | −0.1 |
| dog        | 1 Pixel        | GAS    | 0.660 ± 0.294       | 0.549 ± 0.355       | 98.8            | −14.8|
|            | 4 Pixel        | GAS-L  | 0.937 ± 0.215       | 0.935 ± 0.192       | 98.9            | −20.9|
|            | Blend          | GAS    | 0.987 ± 0.033       | 0.995 ± 0.015       | 99.0            | −11.8|

Table 35: Target-Driven Mitigation for the Availability Backdoor Attack: Data sanitization applied to the vision backdoor attack on CIFAR10 class pair $y_{targ} = \text{auto}$ (Weber et al.’s [78] attack) and $y_{adv} = \text{dog}$ (our reformulated attack) with 150 backdoor instances and ≥10 trials per setup. These results demonstrate that Weber et al.’s backdoor attack can be reformulated as an availability attack that hijacks our framework to remove significant clean training data. The risk of such an attack can be mitigated by modifying Algorithm 4 to include a threshold on mitigation’s maximum effect before additional intervention is initiated. Attack success rate (ASR) is w.r.t. the analyzed target.

| Test Class | Attack Pattern | Method | $D_{adv}$ % Removed | $D_{cl}$ % Removed | Orig. Te. Acc. % | Chg. |
|------------|----------------|--------|---------------------|---------------------|-----------------|------|
| auto       | 1 Pixel        | GAS    | 47.2                | 0                   | 98.8            | −14.8|
|            | 4 Pixel        | GAS-L  | 48.2                | 0                   | 98.9            | −15.4|
|            | Blend          | GAS    | 67.6                | 0                   | 99.0            | −20.9|
| dog        | 1 Pixel        | GAS    | 94.0                | 45.77               | 98.8            | −11.8|
|            | 4 Pixel        | GAS-L  | 89.0                | 44.90               | 98.9            | −9.5 |
|            | Blend          | GAS    | 99.3                | 51.25               | 99.0            | −6.2 |
F.5 Poisoning-Rate Ablation Study

This section analyzes the effect poisoning rate (i.e., fraction of the training set that is adversarial) has on our method’s performance. Target identification and target-driven attack mitigation rely on successfully identifying the adversarial set $D_{\text{adv}}$. As such, the ablation study focuses on the effect of poisoning rate on adversarial-instance identification – providing results for all four attacks in Section 6.1.

Recall from Section 6 that our method had the worst performance on Zhu et al.’s [84] vision poisoning attack. Therefore, we focus on that attack and study the effect of poisoning rate on the attack’s target identification and target-driven mitigation performance. For completeness, our ablation study also includes target identification and target-driven mitigation results for one backdoor attack. We selected Liu et al.’s [46] speech recognition backdoor attack since it is a different data modality and dataset than Zhu et al. – unlike Weber et al. [78] which also uses CIFAR10.

Overall, our method was remarkably stable across poisoning rates in all tested cases. Evaluation was limited to those poisoning rates that had $\geq 50\%$ ASR on related experiment setups.

As detailed in Section C.2.4, Liu et al. [47] provide a speech recognition dataset that comes bundled with 300 backdoor training examples, where Liu et al.’s adversarial trigger was white noise inserted at the beginning of the speech recording. Our speech recognition experiments used a fixed validation set selected u.a.r. Table 5 details the number of backdoor training instances for each speech class pair, with the remaining instances (out of 30) being part of said validation set. The ablation study was limited to $\geq 10$ backdoor instances to ensure the attack succeeded for all class pairs.

For Wallace et al.’s [73] natural-language poisoning attack, adversarial sets with fewer than 10 instances did not consistently succeed and are excluded in the analysis below.

Zhu et al.’s [84] vision poisoning attack was limited to $|D_{\text{adv}}| \leq 400$ since larger quantities exceeded the Nvidia K80’s GPU VRAM capacity of 11.5GB.

The following three subsections visualize our method’s performance across adversarial-set identification, target identification, and attack mitigation, respectively. To improve this section’s readability, tables with the full numerical adversarial-set identification and target identification results (including variance) are deferred to Section F.5.4.
F.5.1 Adversarial-Set Identification Ablation Study.

To identify the target (Alg. 3) and mitigate the attack (Alg. 4), we must be able to identify the adversarial set, \( \mathcal{D}_{adv} \). Our approach uses (renormalized) influence estimators, which should rank the adversarial set as more influential on the target than clean instances (\( \mathcal{D}_cl \)). This section evaluates how well different influence estimators perform this ranking for a u.a.r. target across various poisoning rates.

Across all four attacks (Figures 26–29), these experiments highlight GAS(-L)’s stability identifying adversarial set \( \mathcal{D}_{adv} \) across the poisoning rate spectrum – even outperforming Section 6’s results in many cases.

Figure 26: Speech Backdoor Rate Adversarial-Set Identification Ablation Study: Effect of the number of backdoor instances on mean adversarial-set identification AUPRC (\( |\mathcal{D}_{sl}| = 3,000 \)). GAS’s performance improved slightly with fewer adversarial instances, while GAS-L’s performance increased with larger \( \mathcal{D}_{adv} \). Results are averaged across related experimental setups with \( \geq 10 \) trials per setup. Table 38 provides the full numerical results, including variance.

Figure 27: Vision Backdoor Rate Adversarial-Set Identification Ablation Study: Effect of the fraction of the training set that is backdoors on mean adversarial-set identification AUPRC (\( |\mathcal{D}_{tr}| = 10,000 \)). Only attacks with a minimum success rate of 50% were considered. Results are averaged across related experimental setups with \( \geq 10 \) trials per setup. Table 39 provides the full numerical results, including variance.

Figure 28: Natural-Language Poisoning Rate Adversarial-Set Identification Ablation Study: Effect of the number of adversarial instances on mean adversarial-set identification AUPRC for Wallace et al.’s [73] natural-language poisoning attack on SST-2 (\( |\mathcal{D}_{sl}| = 67,349 \)). While TracInCP’s performance changes significantly across the entire poisoning rate range, GAS and GAS-L’s performance is essentially perfect. Results are averaged across related experimental setups with \( \geq 5 \) trials per setup for each of the first four reviews. Table 40 provides the full numerical results, including variance.

Figure 29: Vision Poisoning Rate Adversarial-Set Identification Ablation Study: Effect of the number of adversarial instances on mean adversarial-set identification AUPRC for Zhu et al.’s [84] vision poisoning attack on CIFAR10 (\( |\mathcal{D}_{tr}| = 25,000 \)). Results are averaged across related experimental setups with \( \geq 10 \) trials per setup. Only attacks that successfully changed the target’s label are considered. The adversarial set was limited to \( \leq 400 \) instances by the Nvidia K80’s GPU VRAM capacity of 11.5GB. Table 41 provides the full numerical results, including variance.
F.5.2 Target-Identification Ablation Study.

This section considers poisoning rate’s effects on target identification. First, for Liu et al.’s [47] speech backdoor attack, we consider two class pairs, one of which is $1 \rightarrow 2$ for which we observed the worst performance among those speech class pairs tested (see Table 18). Note that in Section 6, $\kappa = 10$ was used to determine the speech attack’s upper-tail heaviness. However, these experiments consider the case where adversarial set $\mathcal{D}_{\text{adv}}$ has as few as 6 instances. As discussed in Section 5.2, if $|\mathcal{D}_{\text{adv}}| < \kappa$, target identification performance degrades severely. As such, Figure 30 uses an upper-tail count of $\kappa = \min\{10, |\mathcal{D}_{\text{adv}}|\}$.

The other attack considered below is Zhu et al.’s [84] vision poisoning attack on CIFAR10. We selected this attack since it was the one on which we observed the worst target identification performance (see Figure 6.3). Note that as poisoning rate increased, a larger upper-tail count ($\kappa$) improved target identification performance. Below we report target identification performance on vision poisoning with $\kappa = 2$ (like in Section 6.3) as well as with $\kappa = 10$.

Overall, target identification was stable across all tested poisoning rates for both the backdoor and poisoning attacks.

![Graph](image)

**Figure 30:** Speech Backdoor Rate Target Identification Ablation Study: Effect of the number of backdoor instances on mean target identification AUPRC for class pairs $0 \rightarrow 1$ and $1 \rightarrow 2$ ($|\mathcal{D}_{\text{cl}}| = 3,000$). Liu et al.’s [47] speech dataset includes 30 backdoor instances per class, which were downsampled uniformly at random. Results are averaged across related experimental setups with $\geq 5$ trials per setup. Table 42 provides the full numerical results, including variance.

![Graph](image)

(a) $\kappa = 2$

(b) $\kappa = 10$

**Figure 31:** Vision Poisoning Rate Target Identification Ablation Study: Effect of the number of adversarial instances on mean target identification AUPRC for Zhu et al.’s [84] vision poisoning attack on CIFAR10 ($|\mathcal{D}_{\text{tr}}| = 25,000$). Results are averaged across all four class pairs with $\geq 10$ trials per setup. Table 43 provides the full numerical results, including variance.
F.5.3 Target-Driven Attack Mitigation Ablation Study.

This section considers poisoning rate’s effects on target-driven attack mitigation – specifically for the speech backdoor and vision poisoning attacks. In both cases, the fraction of clean data removed was slightly larger when the adversarial set was small.

For Zhu et al.’s [84] vision poisoning attack, the average fraction of clean data removed was ≤0.26% across the poisoning rate range – a very small fraction. Similarly, the test accuracy post-sanitization never lagged the baseline test accuracy by more than 0.2%; see Table 43 for the baseline results.

### Table 36: Speech Backdoor Rate Target-Driven Attack Mitigation Ablation Study: Algorithm 4’s target-driven, iterative data sanitization applied to Liu et al.’s [47] backdoored speech recognition dataset for class pairs $0 \rightarrow 1$ and $1 \rightarrow 2$ across different backdoor quantities ($|D_{cl}| = 3,000$). The attacks were neutralized with few clean instances removed and little change in test accuracy. Attack success rate (ASR) is w.r.t. specifically the analyzed target. Bold denotes the best mean performance with 10 trials per class pair.

| # BD | Digits | Method | % Removed | ASR % | Test Acc. % |
|------|--------|--------|-----------|-------|-------------|
|      | $y_{tgt}$ | $y_{adv}$ | $\mathcal{D}_{adv}$ | $\mathcal{D}_{cl}$ | Orig. | Ours | Orig. | Chg. |
| 10   | 0 1 | GAS | 100 | 0.17 | 97.2 | 0 | 97.7 | 0.0 |
|      |    | GAS-L | 100 | 0.08 |    | 0 | 0.0 |     |
|      | 1 2 | GAS | 99.0 | 0.03 | 100 | 0 | 97.8 | 0.0 |
|      |    | GAS-L | 98.7 | 0.07 |    | 0 | 0.0 |     |
| 15   | 0 1 | GAS | 100 | 0.12 | 99.8 | 0 | 97.8 | -0.1 |
|      |    | GAS-L | 100 | 0.08 |    | 0 | 0.0 |     |
|      | 1 2 | GAS | 99.4 | 0.04 | 100 | 0 | 97.8 | -0.1 |
|      |    | GAS-L | 98.8 | 0.07 |    | 0 | 0.0 |     |
| 20   | 0 1 | GAS | 100 | 0.04 | 100 | 0 | 97.7 | 0.0 |
|      |    | GAS-L | 100 | 0.02 |    | 0 | 0.0 |     |
|      | 1 2 | GAS | 99.3 | 0.00 | 100 | 0 | 97.7 | 0.0 |
|      |    | GAS-L | 100 | 0.04 |    | 0 | -0.1 |     |
| 25   | 0 1 | GAS | 100 | 0.07 | 100 | 0 | 97.8 | 0.0 |
|      |    | GAS-L | 100 | 0.02 |    | 0 | 0.0 |     |
|      | 1 2 | GAS | 99.7 | 0.00 | 100 | 0 | 97.8 | 0.0 |
|      |    | GAS-L | 99.3 | 0.13 |    | 0 | -0.1 |     |
| 30   | 0 1 | GAS | 100 | 0.06 | 100 | 0 | 97.7 | 0.0 |
|      |    | GAS-L | 100 | 0.03 |    | 0 | 0.0 |     |
|      | 1 2 | GAS | 100.0 | 0.02 | 100 | 0 | 97.7 | 0.0 |
|      |    | GAS-L | 99.8 | 0.09 |    | 0 | 0.0 |     |
Table 37: *Vision Poisoning Rate Target-Driven Attack Mitigation Ablation Study.* Algorithm 4’s target-driven, iterative data sanitization applied to Zhu et al.’s [84] vision poisoning attack on CIFAR10 for class pairs bird → dog and dog → bird ($|D_t| = 25,000$). The attacks were neutralized with few clean instances removed and little change in test accuracy. Attack success rate (ASR) is w.r.t. specifically the analyzed target. Bold denotes the best mean performance with ≥5 trials per class pair.

| # Pois. | Classes | Method | % Removed | ASR % | Test Acc. % |
|----------|---------|--------|-----------|-------|-------------|
|         | $y_{avg}$ | $y_{adv}$ | $D_{adv}$ | $D_{cl}$ | Orig. | Ours | Orig. | Chg. |
| 50       | Bird     | Dog    | GAS      | 38.0  | 0.08  | 93.8 | 0 | 87.0 | +0.2 |
|          |          |        | GAS-L    | 26.4  | 0.07  | 0   | 0 | +0.4 |
|          | Dog      | Bird   | GAS      | 43.2  | 0.00  | 68.8 | 0 | 87.1 | 0.0 |
|          |          |        | GAS-L    | 31.2  | 0.00  | 0   | 0 | 0.0 |
| 150      | Bird     | Dog    | GAS      | 51.6  | 0.07  | 75.0 | 0 | 86.9 | +0.1 |
|          |          |        | GAS-L    | 55.3  | 0.13  | 0   | 0 | +0.1 |
|          | Dog      | Bird   | GAS      | 45.8  | 0.03  | 62.5 | 0 | 87.1 | 0.1 |
|          |          |        | GAS-L    | 37.0  | 0.00  | 0   | 0 | +0.1 |
| 200      | Bird     | Dog    | GAS      | 67.7  | 0.07  | 68.8 | 0 | 87.0 | 0.0 |
|          |          |        | GAS-L    | 72.1  | 0.11  | 0   | 0 | 0.0 |
|          | Dog      | Bird   | GAS      | 50.1  | 0.02  | 62.5 | 0 | 87.3 | 0.0 |
|          |          |        | GAS-L    | 37.2  | 0.00  | 0   | 0 | 0.0 |
| 250      | Bird     | Dog    | GAS      | 60.3  | 0.08  | 87.5 | 0 | 87.0 | 0.0 |
|          |          |        | GAS-L    | 46.6  | 0.02  | 0   | 0 | +0.1 |
|          | Dog      | Bird   | GAS      | 28.8  | 0.00  | 81.3 | 0 | 87.2 | 0.0 |
|          |          |        | GAS-L    | 26.4  | 0.00  | 0   | 0 | 0.0 |
| 400      | Bird     | Dog    | GAS      | 68.1  | 0.18  | 87.5 | 0 | 86.9 | +0.1 |
|          |          |        | GAS-L    | 63.1  | 0.26  | 0   | 0 | 0.0 |
|          | Dog      | Bird   | GAS      | 64.3  | 0.06  | 43.8 | 0 | 87.1 | 0.0 |
|          |          |        | GAS-L    | 42.5  | 0.04  | 0   | 0 | +0.3 |
F.5.4 Ablation Study Reference Tables.

As explained above, the ablation study’s result tables were deferred until this section to improve overall readability.

Table 38: Speech Backdoor Rate Adversarial-Set Identification Ablation Study: Mean and standard deviation adversarial-instance identification AUPRC for Liu et al.’s [47] speech dataset, which contains 30 backdoor instances per class pair and 3,000 total clean instances. Bold denotes the best mean performance with ≥10 trials per setup. The backdoor count includes examples held-out as part of the fixed validation set (see Table 5), with results limited to ≥10 backdoor instances to ensure the attack succeeds consistently for all class pairs. These experiments highlight the stability of GAS and GAS-L at identifying adversarial set $D_{adv}$ even at small poisoning rates. Results are shown graphically in Figure 26.

| # Backdoors | Ours      | Baselines       |
|-------------|-----------|-----------------|
|             | GAS       | GAS-L           | TracInCP | TracIn | Influence Func. | Representer Pt. |
| 10          | 0.968 ± 0.039 | 0.922 ± 0.090   | 0.769 ± 0.117 | 0.701 ± 0.136 | 0.569 ± 0.150 | 0.140 ± 0.082 |
| 12          | 0.974 ± 0.047 | 0.939 ± 0.081   | 0.772 ± 0.142 | 0.674 ± 0.154 | 0.507 ± 0.170 | 0.116 ± 0.069 |
| 15          | 0.986 ± 0.026 | 0.954 ± 0.055   | 0.768 ± 0.175 | 0.637 ± 0.199 | 0.547 ± 0.149 | 0.097 ± 0.038 |
| 17          | 0.985 ± 0.021 | 0.960 ± 0.039   | 0.694 ± 0.224 | 0.568 ± 0.208 | 0.522 ± 0.167 | 0.100 ± 0.074 |
| 20          | 0.980 ± 0.038 | 0.957 ± 0.059   | 0.685 ± 0.186 | 0.545 ± 0.162 | 0.532 ± 0.168 | 0.099 ± 0.087 |
| 22          | 0.986 ± 0.022 | 0.956 ± 0.044   | 0.701 ± 0.212 | 0.532 ± 0.191 | 0.623 ± 0.129 | 0.078 ± 0.086 |
| 25          | 0.990 ± 0.019 | 0.964 ± 0.044   | 0.682 ± 0.224 | 0.506 ± 0.178 | 0.642 ± 0.135 | 0.090 ± 0.063 |
| 27          | 0.988 ± 0.025 | 0.972 ± 0.040   | 0.677 ± 0.185 | 0.487 ± 0.135 | 0.693 ± 0.125 | 0.082 ± 0.072 |
| 30          | 0.977 ± 0.037 | 0.974 ± 0.033   | 0.643 ± 0.211 | 0.467 ± 0.149 | 0.700 ± 0.135 | 0.087 ± 0.030 |

Table 39: Vision Backdoor Rate Adversarial-Set Identification Ablation Study: Mean and standard deviation adversarial-instance identification AUPRC for Weber et al.’s [78] vision backdoor attack with ≥7 trials per experimental setup ($|D_{tr}| = 10,000$). Bold denotes the best mean performance. These experiments highlight the stability of GAS and GAS-L at identifying adversarial set $D_{adv}$. Attacks were limited to ≥100 backdoor instances to ensure ≥50% attack success rate. Results are shown graphically in Figure 27.

| % Backdoor | Ours      | Baselines       |
|------------|-----------|-----------------|
|            | GAS       | GAS-L           | TracInCP | TracIn | Influence Func. | Representer Pt. |
| 1          | 0.848 ± 0.208 | 0.866 ± 0.189   | 0.517 ± 0.229 | 0.288 ± 0.137 | 0.067 ± 0.041 | 0.027 ± 0.011 |
| 1.5        | 0.915 ± 0.106 | 0.946 ± 0.076   | 0.519 ± 0.220 | 0.282 ± 0.123 | 0.091 ± 0.051 | 0.027 ± 0.005 |
| 2.5        | 0.948 ± 0.066 | 0.973 ± 0.034   | 0.479 ± 0.198 | 0.280 ± 0.118 | 0.156 ± 0.115 | 0.040 ± 0.009 |
| 5          | 0.984 ± 0.022 | 0.992 ± 0.012   | 0.540 ± 0.140 | 0.322 ± 0.087 | 0.238 ± 0.104 | 0.071 ± 0.015 |
Table 40: Natural-Language Poisoning Rate Adversarial-Set Identification Ablation Study. Mean and standard deviation adversarial-instance identification AUPRC for Wallace et al.’s [73] natural-language poisoning attack on SST-2 for different poisoning rates ($|\mathcal{D}_a|=67,349$). Results are averaged across related experimental setups with $\geq 5$ trials per setup for each of the first four reviews with the best mean performance in bold. Only attacks that successfully changed the target’s label are considered. For attacks with adversarial sets smaller than 10, the target’s label did not consistently change and are excluded. Results are shown graphically in Figure 28.

| # Poison | Ours | Baselines |
|----------|------|-----------|
|          | GAS  | GAS-L     | TracInCP | TracIn | Influence Func. | Representer Pt. |
| 10       | 0.999 ± 0.004 | 0.999 ± 0.002 | 0.575 ± 0.512 | 0.048 ± 0.041 | 0.002 ± 0.001 | 0.001 ± 0.001 |
| 25       | 1 ± 0 | 1 ± 0 | 0.407 ± 0.290 | 0.127 ± 0.084 | 0.019 ± 0.033 | 0.001 ± 0.001 |
| 40       | 1.000 ± 0.001 | 1.000 ± 0.000 | 0.363 ± 0.325 | 0.118 ± 0.110 | 0.014 ± 0.020 | 0.001 ± 0.001 |
| 45       | 1.000 ± 0.001 | 1.000 ± 0.000 | 0.298 ± 0.281 | 0.114 ± 0.096 | 0.010 ± 0.011 | 0.001 ± 0.001 |
| 50       | 0.996 ± 0.023 | 0.999 ± 0.002 | 0.316 ± 0.259 | 0.120 ± 0.098 | 0.005 ± 0.002 | 0.001 ± 0.001 |

Table 41: Vision Poisoning Rate Adversarial-Set Identification Ablation Study. Mean and standard deviation adversarial-instance identification AUPRC for Zhu et al.’s [84] vision poisoning attack on CIFAR10 across different poisoning rates ($|\mathcal{D}_a|=25,000$). These experiments highlight the stability of GAS and GAS-L at identifying adversarial set $\mathcal{D}_\text{adv}$ even at small poisoning rates. In contrast, TracInCP’s performance changes significantly across the entire poisoning rate range. Results are averaged across related experimental setups with $\geq 10$ trials per setup with the best mean performance in bold. Only attacks that successfully changed the target’s label are considered. Results are shown graphically in Figure 29.

| # Poison | Ours | Baselines |
|----------|------|-----------|
|          | GAS  | GAS-L     | TracInCP | TracIn | Influence Func. | Representer Pt. |
| 40       | 0.726 ± 0.085 | 0.692 ± 0.092 | 0.270 ± 0.030 | 0.107 ± 0.011 | 0.066 ± 0.020 | 0.025 ± 0.007 |
| 100      | 0.835 ± 0.061 | 0.805 ± 0.055 | 0.275 ± 0.068 | 0.122 ± 0.017 | 0.067 ± 0.022 | 0.031 ± 0.004 |
| 150      | 0.790 ± 0.074 | 0.761 ± 0.082 | 0.241 ± 0.013 | 0.107 ± 0.006 | 0.060 ± 0.015 | 0.038 ± 0.003 |
| 200      | 0.857 ± 0.074 | 0.839 ± 0.063 | 0.255 ± 0.042 | 0.123 ± 0.017 | 0.074 ± 0.021 | 0.048 ± 0.005 |
| 250      | 0.837 ± 0.046 | 0.821 ± 0.050 | 0.242 ± 0.043 | 0.129 ± 0.016 | 0.069 ± 0.015 | 0.054 ± 0.004 |
| 400      | 0.869 ± 0.046 | 0.866 ± 0.059 | 0.238 ± 0.017 | 0.139 ± 0.013 | 0.090 ± 0.014 | 0.080 ± 0.005 |
Table 42: **Speech Backdoor Rate Target Identification Ablation Study**: Mean and standard deviation target identification AUPRC for Liu et al.’s [47] speech dataset across different backdoor quantities ($|D_{cl}|=3,000$). We consider class pairs $0 \rightarrow 1$ and $1 \rightarrow 2$ with $\geq 5$ trials per setup. Liu et al.’s [47] speech dataset includes 30 backdoor instances per class which were downsampled uniformly at random. Results are averaged across related experimental setups with $\geq 5$ trials per setup with the best mean performance in bold. Results are shown graphically in Figure 30.

| # Backdoors | Ours | Baselines |
|-------------|------|-----------|
|             | GAS  | GAS-L     | Max k-NN | Min k-NN | Most Certain | Least Certain | Random  |
| 10          | 0.801 ± 0.158 | 0.742 ± 0.155 | 0.023 ± 0.001 | 0.440 ± 0.052 | 0.080 ± 0.072 | 0.121 ± 0.070 | 0.070 ± 0.045 |
| 15          | 0.778 ± 0.103 | 0.719 ± 0.147 | 0.023 ± 0.001 | 0.325 ± 0.039 | 0.047 ± 0.022 | 0.160 ± 0.082 | 0.059 ± 0.016 |
| 20          | 0.885 ± 0.150 | 0.823 ± 0.151 | 0.025 ± 0.002 | 0.195 ± 0.089 | 0.032 ± 0.008 | 0.331 ± 0.207 | 0.064 ± 0.054 |
| 25          | 0.873 ± 0.099 | 0.800 ± 0.067 | 0.033 ± 0.010 | 0.243 ± 0.067 | 0.051 ± 0.010 | 0.265 ± 0.136 | 0.069 ± 0.043 |
| 30          | 0.961 ± 0.038 | 0.897 ± 0.086 | 0.094 ± 0.057 | 0.095 ± 0.032 | 0.034 ± 0.013 | 0.222 ± 0.224 | 0.063 ± 0.027 |

Table 43: **Vision Poisoning Rate Target Identification Ablation Study**: Mean and standard deviation target identification AUPRC for Zhu et al.’s [84] vision poisoning attack on CIFAR10 across different poisoning rates ($|D_{tr}|=25,000$). Results are averaged across all four class pairs with $\geq 10$ trials per setup with the best mean performance in bold. Results are shown graphically in Figure 31.

(a) $\kappa = 2$

| # Poison | Ours | Baselines |
|----------|------|-----------|
|          | GAS  | GAS-L     | Max k-NN | Min k-NN | Most Certain | Least Certain | Random  |
| 50       | 0.758 ± 0.251 | 0.402 ± 0.364 | 0.400 ± 0.411 | 0.016 ± 0.013 | 0.054 ± 0.047 | 0.014 ± 0.004 | 0.039 ± 0.052 |
| 150      | 0.840 ± 0.259 | 0.499 ± 0.347 | 0.467 ± 0.427 | 0.012 ± 0.005 | 0.053 ± 0.047 | 0.013 ± 0.003 | 0.053 ± 0.091 |
| 200      | 0.721 ± 0.309 | 0.463 ± 0.342 | 0.474 ± 0.398 | 0.011 ± 0.003 | 0.053 ± 0.040 | 0.013 ± 0.003 | 0.046 ± 0.071 |
| 400      | 0.717 ± 0.301 | 0.418 ± 0.371 | 0.460 ± 0.430 | 0.016 ± 0.016 | 0.085 ± 0.139 | 0.017 ± 0.010 | 0.069 ± 0.111 |

(b) $\kappa = 10$

| # Poison | Ours | Baselines |
|----------|------|-----------|
|          | GAS  | GAS-L     | Max k-NN | Min k-NN | Most Certain | Least Certain | Random  |
| 50       | 0.750 ± 0.310 | 0.193 ± 0.225 | 0.181 ± 0.319 | 0.016 ± 0.013 | 0.054 ± 0.047 | 0.014 ± 0.004 | 0.039 ± 0.052 |
| 150      | 0.866 ± 0.162 | 0.352 ± 0.298 | 0.364 ± 0.418 | 0.012 ± 0.005 | 0.055 ± 0.047 | 0.013 ± 0.003 | 0.053 ± 0.091 |
| 200      | 0.884 ± 0.190 | 0.427 ± 0.350 | 0.415 ± 0.445 | 0.011 ± 0.003 | 0.053 ± 0.040 | 0.013 ± 0.003 | 0.046 ± 0.071 |
| 250      | 0.853 ± 0.265 | 0.389 ± 0.384 | 0.342 ± 0.399 | 0.018 ± 0.018 | 0.088 ± 0.130 | 0.022 ± 0.017 | 0.087 ± 0.135 |
| 400      | 0.808 ± 0.309 | 0.436 ± 0.384 | 0.401 ± 0.458 | 0.016 ± 0.016 | 0.085 ± 0.139 | 0.017 ± 0.010 | 0.069 ± 0.111 |
Section 5.2 defines the upper-tail heaviness of influence vector $v$ as the $\kappa$-th largest anomaly score in vector $\sigma$. Test examples in set $\tilde{Z}_{te}$ are then ranked by their respective heaviness with those highest ranked more commonly targets.

Figure 32 reports GAS’s and GAS-L’s target identification AUPRC for the vision backdoor [78] and natural language poisoning [73] attacks across a range of $\kappa$ values. In all cases, our performance is remarkably stable. For example, GAS-L’s and GAS’s natural language target identification AUPRC varied only 0.2% and 2.1% respectively for all tested $\kappa \in [1, 25]$. The vision backdoor target identification AUPRC varied only 1.8% and 5.8% respectively for $\kappa \in [2, 50]$.

Recall from Section 6.1 that the vision backdoor adversarial set is three times larger than that of natural language poisoning (150 vs. 50). That is why vision backdoor target identification’s performance is stable over a wider range of $\kappa$ values than natural language poisoning.

Figure 32: Upper-Tail Heaviness Hyperparameter ($\kappa$) Sensitivity Analysis: Mean target identification AUPRC across a range of tail-heaviness values ($\kappa$) for GAS and GAS-L. Target identification performance fluctuates very little even when $\kappa$ changes by more than an order of magnitude. Results are averaged across all experimental setups/class pairs with $\geq 10$ trials per setup.
Alternatives to Renormalization

Renormalization directly addresses the low-loss penalty of Yeh et al.'s [81] representer point estimator by normalizing by \( \frac{\partial L(z_i; \theta^T)}{\partial a y_i} \). For the other influence estimators (TracIn, TracInCP, and influence functions), a slightly different approach is taken.

For influence estimator \( I \), Section 4.3 defines that renormalized influence \( I_{re} \) replaces each gradient vector \( g \) in \( I \) with unit vector \( \frac{g}{\|g\|} \). This modified approach does not correct solely for the low-loss penalty. This choice was made for a few reasons. First, loss and gradient magnitude are generally very tightly coupled as shown in Figure 2, making the two often interchangeable. In addition, most automatic differentiation frameworks (e.g., torch) do not directly return just the loss function’s gradient; rather, they provide the gradient for each parameter. It is therefore an easier implementation to normalize by the full gradient’s magnitude. Lastly, using the full gradient vector’s norm also corrects for variance in the other parts of the gradient’s magnitude – specifically w.r.t. the parameter values.

This section examines the change in GAS’s performance had \( \frac{\partial L(z_i; \theta^T)}{\partial a y_i} \) been used to renormalize TracInCP instead of \( \|g\| \). Figure 33 compares those two approaches against the TracInCP baseline for the CIFAR10 & MNIST joint training experiment in Section 4.1 as well as Section 6.2’s adversarial-set identification experiments.31 As in the original experiments, performance was measured using AUPRC.

For backdoor vision and poison, performance was comparable irrespective of which of the two renormalization schemes was used. Renormalizing TracInCP using \( \frac{\partial L(z_i; \theta^T)}{\partial a y_i} \) actually performed better on the CIFAR10 & MNIST joint training experiment, achieving near-perfect (0.998) mean AUPRC. In contrast, GAS was the top performer for the natural language poisoning experiments. The performance difference on that baseline is primarily due to RoBERTaBASE’s very large model size and the large variance that can induce on the magnitude of vector \( \frac{a}{\Delta x} \).

Figure 33: Alternate GAS Renormalization Using \( \frac{\partial L(z_i; \theta^T)}{\partial a y_i} \): Mean \( D_{adv} \) identification AUPRC for the CIFAR10 & MNIST joint training and adversarial attack experiments. For the two backdoor attacks, both renormalization schemes performed similarly. Renormalization using only the loss function’s norm performed best for the joint training experiments while GAS was the top performer for natural language poisoning. Results are averaged across all experimental setups/class pairs with \( \geq 10 \) trials per setup.

Barshan et al. [4] propose relative influence, which normalizes specifically influence functions by Hessian-vector product (HVP), \( H^{-1}_\theta g_i \). They theoretically motivate their method via a global-vs.-local influence tradeoff. However, as Barshan et al. acknowledge, their method is computationally intractable. Estimating a single HVP can take up to several hours. Since Hessian-vector product \( H^{-1}_\theta g_i \) must be calculated for each training instance \( z_i \), Barshan et al. only evaluate their method on training sets of a few hundred instances. Even if it were computationally feasible, existing HVP-estimation methods are fragile and generally perform poorly on large models [5].

Moreover, relative influence is specific to influence functions while our renormalized influence in Section 4 applies generally to all loss-based influence estimators (e.g., TracIn, HyDRA, representer point). An additional consequence of that is Barshan et al. do not consider normalizing by test gradient magnitude \( \|b g_t\| \) to prevent penalizing low-loss training iterations.

31At the time of submission, data collection for Zhu et al.’s [84] vision poisoning attack had not yet completed.
F.8 Why Aggregate Training Gradients?

Recall from Section 3.2 that static influence estimators (e.g., influence functions [35], representer point [81]) quantify influence using only final model parameters \( \theta_T \). In contrast, dynamic influence estimators (e.g., TracIn & TracInCP [57], HyDRA [12]) quantify influence by aggregating changes across (a subset of) intermediate training parameters, \( \theta_0, \ldots, \theta_T \). These experiments examine the benefit of gradient aggregation for our renormalized influence estimator, GAS, which is based on TracInCP.

Similar to Section 6.2’s experimental setup, Figure 34 compares the mean adversarial set \( (D_{\text{adv}}) \) identification AUPRC for GAS against two custom variants GAS\(_0\) and GAS\(_T\), where GAS\(_\tau\) denotes that the corresponding iteration subset is \( \tau = \{ \tau \} \).\(^{32}\) In all cases, GAS aggregation outperformed the single best checkpoint on average. Therefore, even if the optimal checkpoint could be known via an oracle, it still may not beat aggregation in many cases. In particular for poisoning but also for Weber et al.’s [78] vision backdoor attack where the adversarial trigger may be clipped, each adversarial instance in \( D_{\text{adv}} \) may not have the same attack pattern/data. Therefore, instances in \( D_{\text{adv}} \) may be most influential at different points in training depending on when the corresponding adversarial feature(s) best aligns with the target. Averaging across multiple checkpoints enables dynamic methods to potentially detect all such points in training.

Note also that \( \theta_T \) was better at identifying \( D_{\text{adv}} \) for only two of three attacks, namely natural-language poisoning and vision backdoor. For vision poisoning, \( \theta_0 \) yielded better identification. This performance difference is not due to whether \( \theta_0 \) is pre-trained as both poisoning attacks used pre-trained models.

---

\(^{32}\)For example, GAS\(_0\) denotes \( \tau = \{ 0 \} \), i.e., only initial parameters \( \theta_0 \) are analyzed.
Table 44: **Vision Backdoor Gradient Aggregation**: Adversarial set ($D_{adv}$) identification AUPRC mean and standard deviation averaged over ≥15 trials for CIFAR10 label pair Auto → Dog ($y_{targ} \rightarrow y_{adv}$) using Weber et al.’s [78] three attack patterns. Following Weber et al.’s [78] experimental setup, $\theta_0$ was randomly initialized causing GAS$_0$ as expected to perform poorly. Bold denotes the best mean performance. Mean values are shown graphically in Figure 34a.

| $y_{targ} \rightarrow y_{adv}$ | Attack | GAS$_0$ | GAS | GAS$_T$ |
|-------------------------------|--------|---------|------|---------|
| Auto → Dog                    | 1 Pixel| 0.043 ± 0.015 | 0.982 ± 0.015 | 0.909 ± 0.069 |
|                              | 4 Pixel| 0.048 ± 0.021 | 0.989 ± 0.033 | 0.860 ± 0.157 |
|                              | Blend  | 0.042 ± 0.021 | 1 ± 0 | 0.999 ± 0.002 |

Table 45: **Natural Language Poisoning Gradient Aggregation**: Adversarial set ($D_{adv}$) identification AUPRC mean and standard deviation averaged over 10 trials for two positive and two negative SST-2 movie reviews. Bold denotes the best mean performance. Mean values are shown graphically in Figure 34b.

| Sentiment | No. | GAS$_0$ | GAS | GAS$_T$ |
|-----------|-----|---------|------|---------|
| Pos.      | 3   | 0.789 ± 0.144 | 1 ± 0 | 0.986 ± 0.037 |
|           | 4   | 0.453 ± 0.066 | 1 ± 0 | 0.950 ± 0.083 |
| Neg.      | 3   | 0.283 ± 0.087 | 0.998 ± 0.005 | 0.886 ± 0.213 |
|           | 4   | 0.339 ± 0.108 | 1 ± 0 | 0.992 ± 0.025 |

Table 46: **Vision Poisoning Gradient Aggregation**: Adversarial set ($D_{adv}$) identification AUPRC mean and standard deviation averaged over 30 trials for three CIFAR10 class pairs ($y_{targ} \rightarrow y_{adv}$) with Zhu et al.’s [84] convex polytope poisoning attack. Bold denotes the best mean performance. Mean values are shown graphically in Figure 34c.

| $y_{targ} \rightarrow y_{adv}$ | GAS$_0$ | GAS | GAS$_T$ |
|-------------------------------|--------|------|---------|
| Bird → Dog                    | 0.773 ± 0.208 | 0.892 ± 0.137 | 0.678 ± 0.265 |
| Frog → Deer                   | 0.912 ± 0.120 | 0.962 ± 0.100 | 0.837 ± 0.160 |
| Deer → Frog                   | 0.803 ± 0.188 | 0.888 ± 0.091 | 0.604 ± 0.234 |
F.9 Evaluating on the Tail of the Attack’s Influence Distribution

When the number of attack instances is larger than necessary, it becomes increasingly important to detect all of the adversarial set (with very few false positives) in order to mitigate the attack and minimize damage. As a complement to this paper’s identification and mitigation experiments, here we evaluate one specific aspect of our method in one specific challenging setting: performance on the 10% most difficult instances in the attack.

Figure 35 considers this case on Weber et al.’s [78] three vision backdoor attack patterns. Each attack’s maximum perturbation distance is halved w.r.t. the earlier experiments (to reduce their individual influence and increase the challenge) and 1,000 adversarial instances (10% of $D_{\text{tr}}$) are used. Each method’s gray bar represents the adversarial-instance identification AUPRC when considering the entire adversarial set $D_{\text{adv}}$. The purple bar shows the adversarial-instance identification AUPRC for just the bottom 10% least influential backdoor training instances. This quantity is necessarily less than or equal to the AUPRC for the full $D_{\text{adv}}$. This experiment mimics an attacker secretly concealing from our defense the top-90% of $D_{\text{adv}}$. Figures 36 and 37 provide similar results for the Wallace et al.’s [73] natural-language poisoning and Liu et al.’s [47] speech recognition backdoor attacks respectively.

In summary, GAS(-L) are highly adept at detecting these least influential instances for all three attacks. In contrast, the baseline estimators’ performance drops substantially – generally to nearly zero AUPRC.

Figure 35: Vision Backdoor Adversarial-Set Identification of the Least Influential Instances: Mean AUPRC identifying adversarial set $D_{\text{adv}}$ for Weber et al.’s [78] vision backdoor with 1,000 backdoor instances. Even if the attacker can conceal the top 90% most influential instances from our defense (purple), GAS(-L) still remain highly effective at identifying the adversarial set – unlike the baselines whose performance severely degrades. Results are averaged across related experimental setups with $\geq 10$ trials per setup.

Figure 36: Natural-Language Poison Adversarial-Set Identification of the Least Influential Instances: Mean AUPRC identifying adversarial set $D_{\text{adv}}$ for Wallace et al.’s [73] natural-language poisoning attack. Even if the attacker can conceal the top 90% most influential instances (i.e., top 45/50 instances) from our defense (purple), GAS(-L) remain highly effective at identifying $D_{\text{adv}}$ even for these 10% least-influential instances (bottom 5/50). Results are averaged across related experimental setups with 10 trials per setup.

Figure 37: Speech Backdoor Adversarial-Set Identification of the Least Influential Instances: Mean AUPRC identifying adversarial set $D_{\text{adv}}$ for Liu et al.’s [47] speech backdoor attack. Even if the attacker can conceal the top 90% most influential instances (top 27/30 instances) from our defense (purple), our method still remains highly effective at identifying the adversarial set using just the 10% least influential instances (bottom 3/30 instances). Results are averaged across related experimental setups with $\geq 10$ trials per setup.
F.10 Adversarial-Set Identification Execution Time

Recall from Section 4.3 that GAS(-L) is a renormalized version of TracInCP that removes the low-loss penalty. Therefore, GAS(-L)’s execution time is, in essence, that of TracInCP. Table 47 compares the execution time of TracInCP/GAS(-L) to the other influence estimators. All results were collected on an HPC system with 3 Intel E5-2690v4 cores, 48GB of 2400MHz DDR4 RAM, and one NVIDIA Tesla K80. The reported execution times consider calculating vector \( v \) for training set \( D_{tr} \) w.r.t. a single random test example \( z_{te} \). The only exception is the “Amortized” results where training gradient \( (g_i) \) computation is amortized across multiple test examples allowing for significant speed-up (see Alg. 2). Amortization relies on simultaneously storing each test example’s gradient \( g_{te} \) in GPU memory. Therefore, the GPU model constrains amortization’s possible benefits. Table 48 enumerates the number of concurrent test examples considered when calculating amortized results on a K80. More modern GPUs with larger onboard memory see considerable single and amortized speed-ups compared to Table 47.

Recall from Section 6 that natural-language poisoning attacked RoBERTaBASE (125M parameters) as specified by Wallace et al. [73]. Large models can be slow to analyze – even with amortization. That is why natural language target identification uses the two-phase target identification procedure described in Section 5.2. We observed no meaningful performance drop with this streamlined approach (see Figure 7).

Recall from Section 3.2 that Yeh et al.’s [81] representer point estimator only considers the network’s final linear classification layer, which is why it was the fastest. Notably, it also had the worst performance (see Figure 6).

For completeness, Peri et al.’s [55] Deep \( k \)-NN empirical clean-label data poisoning defense’s mean execution time was 242s with standard deviation 1.1s.

Target identification execution times can be extrapolated from Table 47’s “Amortized” values.

Table 47: Adversarial-Set Identification Execution Time: Mean and standard deviation algorithm execution time (in seconds) to analyze a single test instance across >50 trials for GAS(-L) and the influence estimator baselines on Section 6.1’s four training-set attacks. See Table 48 for the number of parallel instances analyzed by TracInCP and GAS(-L) for each attack.

| Attack Type | Dataset | TracInCP & GAS(-L) | Others |
|-------------|---------|--------------------|--------|
| | | Single | Amortized | TracIn | Inf. Func. | Rep. Pt. |
| Backdoor | Speech [47] | 2,605 ± 710 | 489 ± 155 | 894 ± 279 | 4,595 ± 177 | 49 ± 1 |
| | Vision CIFAR10 | 9,252 ± 2,253 | 1,473 ± 408 | 2,418 ± 738 | 6,316 ± 250 | 128 ± 3 |
| Poison | NLP SST-2 | 27,723 ± 7,933 | 8,971 ± 3,719 | 16,667 ± 187 | 21,409 ± 77 | 1,697 ± 11 |
| | Vision CIFAR10 | 24,267 ± 1,939 | 2,088 ± 177 | 5,910 ± 600 | 15,634 ± 641 | 187 ± 2 |

Table 48: Single GPU Amortization: Number of test examples analyzed by TracInCP/GAS(-L) when amortizing on an Nvidia Tesla K80 computation of training gradients \( (g_i) \) in Table 47’s “Amortized” execution time result.

| Attack Type | |
|-------------|---------|
| Backdoor Speech | 20 |
| Vision | 16 |
| Poison NLP | 5 |
| Vision | 135 |