TIDE: DEFENDING AGAINST BACKDOOR DATA POISONING VIA DENSE INFLUENCE SUBGRAPH DETECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Targeted poisoning attacks perturb the training data of machine learning models to increase the probability that the trained models make certain errors. Stealthy backdoor attacks, which perturb data by injecting inconspicuous triggers into the features, pose a particularly serious threat to the reliability and security of machine learning. To detect and filter backdoor-poisoned data, we shift the focus from identifying individually anomalous data points to identifying groups of training data that have a disproportionately high impact on each other during training. Specifically, we first introduce an efficient approach for estimating how the inclusion of a data point in model training influences the output of the trained model for another data point. Leveraging the observation that this influence is disproportionally high between pairs of poisoned data, we introduce an efficient algorithm for detecting poisoned data as a dense subgraph in the graph formed by data points and the influence values between them. We evaluate our proposed approach against state-of-the-art backdoor poisoning attacks on benchmark datasets, demonstrating that it outperforms existing approaches for data cleaning in terms of reducing attack success and increasing model accuracy. We also demonstrate robustness against an adaptive attack that attempts to circumvent our defense by reducing the cross-influence of poisoned data.

1 INTRODUCTION

Machine learning models are increasingly deployed in high-stakes applications, making their robustness to adversarial manipulation a critical concern (Hartle III et al., 2025). *Training-time data poisoning* attacks are particularly insidious, as they compromise the model during the learning process (Bai et al., 2024; Hartle III et al., 2025). In targeted poisoning attacks, an adversary modifies a small subset of training data to induce specific errors at test time. A prominent type of such attacks are *backdoor attacks*, in which the attacker embeds a trigger pattern into training samples, causing the model to associate the trigger with an attacker-chosen target label (Bai et al., 2024; Abbasi et al., 2025). At inference time, the presence of the trigger leads to misclassifications, while the model behaves normally on clean inputs.

In this work, we focus on *data cleaning*, identifying and removing or repairing poisoned samples from the training set before training the final model. Unlike model-repair defenses, which modify an already trained model (Liu et al., 2018; Zhao et al., 2020; Li et al., 2021), and inference-time defenses, which detect triggered inputs at test time, data cleaning aims to prevent the model from learning the backdoor in the first place (Guo et al., 2023; Hou et al., 2024).

Existing data-cleaning defenses often identify poisoned samples by exploiting separability in learned representations or training-derived statistics. Activation Clustering (Chen et al., 2019) and spectral-signature methods (Tran et al., 2018; Hayase et al., 2021) separate poisoned samples from clean samples in feature space, while recent methods leverage gradients, representation consistency, or training dynamics (Pan et al., 2023; Qi et al., 2023; Hou et al., 2025). These methods often depend on the poisoned samples exhibiting separable behavior under a particular representation, statistic, or training signal. But such separability can be weak under stealthy attacks, where poisoned samples are designed to remain close to clean samples under representation-based inspection.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

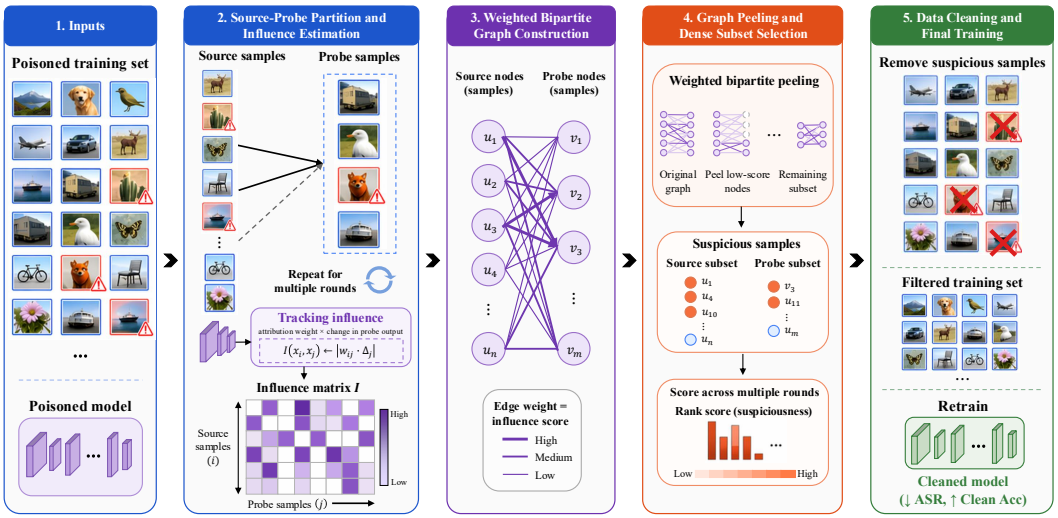

Figure 1: Overview of the proposed data-cleaning approach, TIDE.

In this work, we take a different perspective. Instead of asking whether individual samples look anomalous, we ask how the inclusion of one sample during training affects model predictions on other samples. This distinction is important because poisoned samples may not look suspicious when considered individually, but they can reveal a pattern of shared influence during training. Our hypothesis is that backdoor-poisoned samples exert a disproportionately strong influence on other poisoned samples, but not on clean samples. Thus, we shift detection from identifying isolated anomalies to finding groups of samples with unusually strong inter-sample influence.

To do so, we propose **TIDE** (**T**racking **I**nfluence for **D**ense subgraph **E**xtraction), a two-stage framework that estimates inter-sample influence during training and uses the resulting influence matrix to identify a dense high-influence region corresponding to suspicious samples. In principle, complete inter-sample influence could be estimated by measuring influence between every pair of samples in the dataset. However, computing a complete matrix is expensive for large datasets. For computational tractability, we instead use a bipartite approximation. We randomly partition the dataset into a source set and a probe set, and measure how samples in the source set affect model predictions on samples in the probe set during training. This produces a *source-to-probe influence* matrix, which we use to identify a dense high-influence region corresponding to suspicious samples.

In the first stage, we track stepwise source-to-probe influence throughout training. At each optimization step, we compute how each mini-batch sample changes the prediction of probe samples, attributing these changes to individual training samples. The resulting per-step influence scores are accumulated into a source-to-probe influence matrix. In the second stage, we interpret the resulting influence matrix as a weighted bipartite graph between source samples and probe samples, where each edge represents the influence of a source sample on a probe sample. This bipartite formulation preserves the directional, cross-set nature of influence by separating the source side used for influence estimation from the probe side on which the effect of each update is measured. Because poisoned samples are constructed to encode a shared trigger-target association, poisoned source samples tend to induce larger and more consistent output changes for poisoned probe samples than for clean ones. This creates a dense subgraph of high influence values across the two partitions. We therefore identify poisoned samples by extracting this subgraph with a greedy bipartite subgraph search algorithm. Figure 1 illustrates the overall pipeline.

Experiments on benchmark datasets and strong backdoor attacks show that TIDE achieves a favorable trade-off between poisoned-sample detection and clean-data preservation, reducing attack success while maintaining clean accuracy, compared to state-of-the-art baseline defenses. Therefore, our main contributions are (1) formulating data cleaning as the problem of identifying dense subgraphs of inter-sample influence, (2) introducing TIDE, which combines efficient source-to-probe influence estimation with weighted bipartite subgraph search (Section 4), and (3) demonstrating that TIDE reduces attack success while preserving clean accuracy without requiring a known-clean reference set (Section 5), even against an adaptive attacker (Appendix H). We also study and demonstrate the disproportionate cross-influence of poisoned samples empirically (Appendix A).

2 Related Works

2.1 Backdoor Attacks

Backdoor attacks are training-time poisoning attacks in which an adversary manipulates a small fraction of the training data so that the trained model behaves normally on clean inputs but predicts an attacker-specified target label when a trigger is present Gu et al. (2019); Li et al. (2024); Gao et al. (2020). Early work such as BadNets uses visible patch triggers and target-label poisoning Gu et al. (2019). Later attacks improve stealthiness by designing less perceptible triggers, such as blended triggers Chen et al. (2017) and signal-based triggers Barni et al. (2019).

Beyond dirty-label attacks, clean-label attacks preserve the original labels of poisoned samples, making poisoned data more difficult to detect through label inconsistency or manual inspection Turner et al. (2018); Saha et al. (2020); Souri et al. (2022). Recent attacks further increase flexibility and stealthiness through input-aware, feature-space, or adaptive trigger construction Nguyen & Tran (2020); Zeng et al. (2023); Huynh et al. (2024); Nguyen et al. (2025). These attacks motivate defenses that can identify poisoned data or suppress backdoor behavior in more challenging settings, where triggers may be imperceptible, label-consistent, input-dependent, or embedded in feature space.

2.2 Backdoor defenses

Existing backdoor defenses can be categorized by operational stage. Training-time data filtering methods identify and remove poisoned samples before or during training. Representation-based methods, such as Activation Clustering (Chen et al., 2019) and SPECTRE (Hayase et al., 2021), detect anomalies using clustering or robust statistics applied to learned representations.

Recent methods use richer model-derived or training-time signals. ASSET (Pan et al., 2023), Confusion Training (Qi et al., 2023), GCD (Yuan et al., 2025), and FLARE (Hou et al., 2025) leverage training dynamics, gradients, or multi-layer representations, while PureGen (Bhat et al., 2024) and AIBD (Yin et al., 2025), use generative purification or adversarially inspired training signals.

Model-repair defenses mitigate backdoors after training by modifying, fine-tuning, pruning, or unlearning model components (Liu et al., 2018; Wang et al., 2019; Chen et al., 2025; Lin et al., 2025). Inference-time defenses, such as STRIP (Gao et al., 2019), SentiNet (Chou et al., 2020), SCALE-UP (Guo et al., 2023), TeCo (Liu et al., 2023), and IBD-PSC (Hou et al., 2024), detect or reject triggered inputs at test time. These approaches can suppress backdoor behavior, but they do not directly identify and remove poisoned training samples.

Our work is more closely related to influence-based analysis, including influence graphs for backdoor defense (Sun et al., 2021), trajectory-based influence estimation (Pruthi et al., 2020), and cosine-similarity-based poisoning detection (Hammoudeh & Lowd, 2021). Unlike prior approaches that mainly use marginal influence scores or pairwise relationships, we use source-to-probe influence to identify a dense high-influence region across two data partitions and produce suspiciousness rankings for data cleaning.

3 Threat Model and Defense Setting

We consider the standard backdoor-based poisoning setting studied in prior work (Gu et al., 2019; Chen et al., 2017; Turner et al., 2018; Li et al., 2024; Zeng et al., 2023). In this setting, an attacker perturbs a small portion of the training data so that a model trained on the perturbed data learns an unintended association between an attack pattern (i.e., trigger) and an attacker-specified target class.

Backdoor Attack Pipeline The attacker constructs a poisoned training set by modifying a small subset of training data while leaving the remaining unchanged. We consider clean-label settings, where the original labels of poisoned samples are preserved. The attacker's objective is to produce a poisoned model that simultaneously achieves a high attack success rate, causing trigger induced inputs to be classified as a target class, while maintaining high clean accuracy on benign inputs. This results in a latent vulnerability activated only when the trigger pattern is present at inference time.

Attacker's Capabilities and Knowledge The attacker can modify a small subset of the training set, but does not control the model architecture or the training process. We do not restrict the attack to a specific poisoning mechanism. Within the clean-label setting, the attacker can use local, blended, sinusoidal, input-aware, or optimization-based triggers and perturbations (Gu et al., 2019; Chen et al., 2017; Barni et al., 2019; Huynh et al., 2024; Turner et al., 2018; Zeng et al., 2023). The poisoning rate ϵ is defined as the ratio of poisoned samples to the total size of the training set. Consistent with standard backdoor settings, ϵ is assumed to be small, typically ranging from 0.5% to 5% (Li et al., 2024), to ensure that the attack remains efficient but difficult to detect.

Defender's Goal The defender aims to identify and remove poisoned samples from a poisoned training set before training the model, so that training on the sanitized dataset yields a backdoor-free model. We focus on dataset filtering because it addresses the source of the backdoor, i.e., the poisoned training samples. This approach is complementary to defenses that aim to repair a trained model (Liu et al., 2018) or detect input with triggers at inference time (Gao et al., 2019).

Defender's Capabilities and Knowledge The defender has access to the model architecture, the training algorithm, and the potentially poisoned training set. The defender controls the training procedure, including hyperparameters, and can observe model parameters, gradients, and predictions during training. However, the defender has no prior knowledge of the specific poisoning mechanism, the exact poisoning rate ϵ, or which samples have been compromised. Importantly, the defender does *not* have access to a known-clean reference dataset, a trusted validation set, the trigger pattern, or the target class. Thus, our setting avoids relying on external clean data or attack-specific information, while also not requiring a known upper bound on the poisoning rate.

4 TIDE: Tracking Influence for Dense Subgraph Detection

TIDE combines trajectory-based influence estimation with graph-based subset search. As shown in Figure 1, we first randomly partition the dataset into a *source set* and a *probe set*. We then track how samples in the source set affect model outputs on held-out probe samples during training, producing a source-to-probe influence matrix. This matrix is interpreted as a weighted bipartite graph, where both source and probe samples are represented as nodes and edge weights correspond to influence scores. TIDE then searches this graph for a dense high-influence region and identifies suspicious samples from both sides of the bipartite graph. Sections 4.1 and 4.2 describe influence estimation and dense subgraph search, respectively. Appendix B provides detailed pseudocode for each.

4.1 Tracking Influence

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ denote the poisoned dataset. For data points $x_i, x_j \in \mathcal{D}$, the influence score $I(x_i, x_j)$ measures how much removing x_i from the training set would change the trained model's outputs for x_j. Formally, let θ denote the parameters obtained by training a model f on $\mathcal{D}$, and let θ_{-i} denote the parameters obtained by training on $\mathcal{D} \setminus \{x_i\}$. Then, the influence of x_i on x_j is defined as

$$I(x_i, x_j) = \left\| f_\theta(x_j) - f_{\theta_{-i}}(x_j) \right\|_1 . \tag{1}$$

This provides an idealized definition of inter-sample influence as the total change in the final trained model output when a training sample is removed. Computing this exact quantity for each pair x_i, x_j is infeasible because it would require retraining the model separately for each removed sample x_i. Therefore, we do not directly estimate Equation 1. Instead, we introduce a trajectory-based influence score that accumulates the magnitude of stepwise model-output changes attributable to each training sample during optimization. This score measures cumulative training-trajectory variation rather than a simple final net difference. This formulation is useful because poisoned samples may repeatedly induce large and consistent changes on other poisoned samples during training.

We focus on neural networks trained by gradient descent and estimate influence by measuring how each source sample contributes to changes in model outputs on a held-out probe set over the process of optimization. Unlike classical influence functions (Koh & Liang, 2017), which estimate the impact of a training point after the model has already been trained, our approach tracks influence throughout the training process, building on prior work that reformulates influence in terms of the

optimization trajectory (Hara et al., 2019; Pruthi et al., 2020). Our method follows this principle—attributing changes in model behavior to the training samples that drive them during optimization—but tracks changes in model outputs on a held-out probe set rather than changes in test loss. This distinction is important because classical influence functions typically rely on a locally optimal trained model and provide a post-hoc estimate of influence. In contrast, our method does not assume that the model has reached an optimum; instead, it directly measures how training samples shape model behavior throughout learning. We compare our method with detection based on classical influence functions in Appendix D.2. These results demonstrate that classical influence functions provide a less accurate signal for detecting poisoned data than our method.

Data Partitioning As we measure how each sample affects model behavior on held-out samples during learning, we temporarily partition the dataset for influence estimation. Given a potentially poisoned dataset $\mathcal{D}$, we randomly partition it into two disjoint subsets: a source set $\mathcal{D}_{\text{src}}$ and a probe set $\mathcal{D}_{\text{probe}}$, where $\mathcal{D} = \mathcal{D}_{\text{src}} \cup \mathcal{D}_{\text{probe}}$ and $\mathcal{D}_{\text{src}} \cap \mathcal{D}_{\text{probe}} = \emptyset$. The source set contains the samples whose influence will be estimated, while the probe set provides held-out samples on which we observe changes in model behavior. This random partition allows both subsets to be treated as samples from the same underlying data distribution.

This partition is used only to measure inter-sample influence. After suspicious samples are removed, the final model can be trained on the cleansed dataset $\mathcal{D}^* \subseteq \mathcal{D}_{\text{src}} \cup \mathcal{D}_{\text{probe}}$. Specifically, during influence estimation, a model f_θ with parameters θ is trained on $\mathcal{D}_{\text{src}}$, while $\mathcal{D}_{\text{probe}}$ is used only to observe changes in model outputs. This allows us to estimate how strongly each sample in $\mathcal{D}_{\text{src}}$ affects the model outputs for each sample in $\mathcal{D}_{\text{probe}}$ during training.

Estimation of Influence The overall procedure is summarized in Algorithm 1 in Section B. Rather than estimating influence only after training, we track *stepwise source-to-probe influence* throughout the learning process. Specifically, we train a temporary model from scratch on the source set $\mathcal{D}_{\text{src}}$ using stochastic gradient descent. At each gradient-descent step, we estimate how each source sample in the current mini-batch contributes to changes in model outputs for probe samples. We then accumulate these impacts over epochs, yielding pairwise influence scores between source samples and probe samples.

At gradient-descent step t, let $B_t \subset \mathcal{D}_{\text{src}}$ denote the current mini-batch of samples. In step t, the parameters of the model are updated from θ_t to θ_{t+1} based on the samples in B_t. To measure the impact of this update on a probe sample $x_j \in \mathcal{D}_{\text{probe}}$, we consider the change in logit values:

$$\Delta_j^{(t)} = f_{\theta_{t+1}}(x_j) - f_{\theta_t}(x_j). \tag{2}$$

This change reflects the combined effect of all samples in the mini-batch B_t. Therefore, to measure pairwise influence, we estimate how much each source sample $x_i \in B_t$ contributes to this change.

We use a last-layer approximation to attribute the mini-batch-induced probe-logit change to individual source samples. The rationale is that for the last linear layer, the update induced by a source sample is proportional to the product of its last-layer representation and class-wise prediction error.

Let $h_i^{(t)}$ and $h_j^{(t)}$ denote the the last layer of f for source sample x_i and probe sample x_j, respectively. We compute a class-specific attribution weight $w_{ijc}^{(t)}$, which estimates how much source sample x_i contributes to the change in model output for class c for probe sample x_j in step t:

$$w_{ijc}^{(t)} = \frac{h_i \cdot h_j}{\|h_i\| \cdot \|h_j\|} \left(\mathrm{softmax}(f_{\theta_t}(x_i))[c] - \mathbf{I}[y_i = c]\right). \tag{3}$$

Here, $\mathrm{softmax}(f_{\theta_t}(x_i))[c]$ is the predicted probability of class c for x_i, and $\mathbf{I}[y_i = c] \in \{0, 1\}$ is a binary indicator of whether the true label of source sample x_i is class c.

Because the change $\Delta_j^{(t)}$ is caused by the entire mini-batch B_t, we normalize the attribution weights w_{ijc} across all samples in mini-batch B_t for each probe sample x_j and each class c:

$$\tilde{w}_{ijc}^{(t)} = \frac{w_{ijc}^{(t)}}{\sum_{i' \in B_t} |w_{i'jc}^{(t)}|}. \tag{4}$$

These normalized weights $\tilde{w}_{ijc}$ are then used to attribute the measured probe-logit change $\Delta_j^{(t)}$ to each source sample. The per-step pairwise influence score is

$$I^{(t)}(x_i, x_j) = \sum_{c=1}^{C} \left| \tilde{w}_{ijc}^{(t)} \Delta_{x_j}^{(t)}[c] \right|. \tag{5}$$

Finally, we accumulate the per-step influence scores over all gradient-descent steps in which a training sample x_i appears:

$$I(x_i, x_j) = \sum_{t:\, x_i \in B_t} I^{(t)}(x_i, x_j). \tag{6}$$

After training, $I(x_i, x_j)$ represents the accumulated influence of source sample x_i on probe sample x_j throughout the learning trajectory. This influence matrix is then used by the peeling procedure to rank both source samples in $\mathcal{D}_{\text{src}}$ and probe samples in $\mathcal{D}_{\text{probe}}$, so that the final cleaning step can remove suspicious samples from the entire dataset $\mathcal{D}$.

Computational Cost Our approach avoids retraining the model for each source sample. At each gradient-descent step, the additional cost comes from a single forward pass over $\mathcal{D}_{\text{probe}}$ to measure the change in model outputs (requiring no additional backpropagation). Moreover, since $\mathcal{D}_{\text{probe}}$ can be chosen to be relatively small (even if $\mathcal{D}$ and $\mathcal{D}_{\text{src}}$ are large), this provides a scalable estimation of pairwise influence. We present empirical running-time results for our method in Appendix D.1.

4.2 Dense Subgraph Search

To detect poisoned samples, we analyze the influence matrix I, where each entry $I(x_i, x_j)$ measures the influence of source sample $x_i \in \mathcal{D}_{\text{src}}$ on probe sample $x_j \in \mathcal{D}_{\text{probe}}$. Because poisoned samples are constructed to encode a shared trigger-target association, poisoned source samples are expected to have consistently high influence on poisoned probe samples. As a result, poisoned samples in $\mathcal{D}_{\text{src}}$ and $\mathcal{D}_{\text{probe}}$ should form a dense high-influence region in I. Appendix A provides empirical support for this assumption: poison-to-poison influence is higher on average than between other pairs.

We interpret the influence matrix as a weighted bipartite graph between the source and probe samples. Each source or probe sample is represented as a node, and each entry $I(x_i, x_j)$ defines the edge weight between them. This graph representation allows us to search for a group of samples that are strongly connected through influence, rather than evaluating each sample independently.

Our goal is therefore to identify subsets of $\mathcal{D}_{\text{src}}$ and $\mathcal{D}_{\text{probe}}$ that maximize internal influence while minimizing connections to the remaining samples. Since the poisoning rate is unknown, recovering a single subset is insufficient. Instead, we aim to produce a suspiciousness ranking over all samples.

Subgraph Search Objective Let $S_{\text{src}} \subset \mathcal{D}_{\text{src}}$ and $S_{\text{probe}} \subset \mathcal{D}_{\text{probe}}$ denote the selected subsets, and let $\bar{S}_{\text{src}} = \mathcal{D}_{\text{src}} \setminus S_{\text{src}}$ and $\bar{S}_{\text{probe}} = \mathcal{D}_{\text{probe}} \setminus S_{\text{probe}}$ denote removed subsets.

We define the objective:

$$\begin{aligned} J(S_{\text{src}}, S_{\text{probe}}) = & \frac{\sum_{x_i \in S_{\text{src}}} \sum_{x_j \in S_{\text{probe}}} I(x_i, x_j)}{|S_{\text{src}}|\,|S_{\text{probe}}|} \\ & - \frac{\sum_{x_i \in S_{\text{src}}} \sum_{x_j \in \bar{S}_{\text{probe}}} I(x_i, x_j)}{|S_{\text{src}}|\,|\bar{S}_{\text{probe}}|} - \frac{\sum_{x_i \in \bar{S}_{\text{src}}} \sum_{x_j \in S_{\text{probe}}} I(x_i, x_j)}{|\bar{S}_{\text{src}}|\,|S_{\text{probe}}|}. \end{aligned} \tag{7}$$

The first term measures the average internal influence within the active subsets, encouraging dense connections. The second and third terms penalize influence between active and removed samples, discouraging connections to samples that are likely clean. Together, this objective favors subsets that are internally coherent, while being weakly connected to the rest of the dataset.

We achieve this using a greedy bipartite peeling algorithm. Starting from a complete graph, the algorithm iteratively removes the most weakly connected samples one-by-one. Samples that remain longer during peeling are assigned higher suspiciousness scores, as they are more connected to the dense high-influence region.

Greedy peeling offers two advantages. First, it produces a complete suspiciousness ranking over samples, rather than a binary selection. Second, it scales efficiently to large datasets. To justify this choice, we provide an optimality analysis in Appendix F by comparing greedy objective values against Integer Programming (IP) and Simulated Annealing (SA) baselines. This shows that greedy peeling closely approximates optimization baselines while remaining computationally efficient.

Greedy Peeling A detailed pseudocode for the greedy peeling algorithm is provided as Algorithm 2 in Appendix B. Starting from $S_{\text{src}} = \mathcal{D}_{\text{src}}$ and $S_{\text{probe}} = \mathcal{D}_{\text{probe}}$, we iteratively remove samples from either S_{src} or S_{probe} according to their effect on the peeling objective. For a source-side candidate $x_i \in S_{\text{src}}$, we select a sample to remove that maximizes $J(S_{\text{src}} \setminus \{x_i\}, S_{\text{probe}}) - J(S_{\text{src}}, S_{\text{probe}})$. Similarly, for a probe-side candidate $x_j \in S_{\text{probe}}$, we select a sample to remove that maximizes $J(S_{\text{src}}, S_{\text{probe}} \setminus \{x_j\}) - J(S_{\text{src}}, S_{\text{probe}})$. At each step, we peel from the side with the smaller fraction of samples removed so far, which maintains the relative size of the two sides S_{src} and S_{probe}.

Because $\mathcal{D}_{\text{src}}$ and $\mathcal{D}_{\text{probe}}$ are obtained via a random partitioning of the same dataset $\mathcal{D}$, we expect the two sides to have approximately the same ratio of poisoned samples. Therefore, the dense poisoned subset should have a similar relative size on the two sides of the bipartite graph. To reflect this, proportional peeling balances removals between the two sides, preventing one side from being eliminated earlier than the other and avoiding degenerate solutions that select a disproportionally small subset on one side.

The change in the objective caused by a removal may be positive or negative; therefore, removal does not necessarily increase the objective at every step. Intuitively, samples that are removed early are those whose exclusion is least detrimental, or most beneficial, to the dense high-influence structure. Thus, these samples are weakly connected to the dense structure, and we should consider them to be less suspicious.

During the peeling process, we track the objective value after each removal and retain the subsets that maximize it. The resulting subsets S^*_{src} and S^*_{probe} form the densest high-influence region of the bipartite graph, and they are returned as the detected poisoned samples. However, the peeling order also provides a complete suspiciousness ranking over all samples. Samples removed later receive higher scores because they remain longer in the dense high-influence region. Thus, a defender can clean more aggressively or conservatively by choosing a different threshold along this ranking.

Multiple Random Partitions Since the partition into $\mathcal{D}_{\text{src}}$ and $\mathcal{D}_{\text{probe}}$ is random, the estimated influence scores can vary between different splits. Consequently, the suspicious subsets identified by peeling, S^*_{src} and S^*_{probe}, may also vary between different runs. To improve robustness, we run TIDE multiple times with independently sampled partitions and aggregate suspiciousness rankings. This reduces split-specific variance and allows each sample to be evaluated both as a source sample and as a probe sample across runs.

5 Experiments

5.1 Experimental Setup

Datasets We evaluate TIDE on two classification benchmarks commonly used in the literature on data poisoning and cleaning. **CIFAR-10** (Krizhevsky et al., 2009; Krizhevsky, 2009) consists of 50,000 training images and 10,000 test images from 10 classes, each image being 32×32 pixels in size and 3 colors. **GTSRB** (Stallkamp et al., 2011) contains 39,270 images across 43 traffic sign classes, resized to 32×32 pixels for consistency with prior work. Further results with **Tiny ImageNet** (Le & Yang, 2015) are presented in Appendix G.

TIDE and Model Training Settings For each poisoned dataset, we repeat influence estimation and graph peeling over two randomly sampled source-probe partitions, and aggregate the results to identify the samples to be removed. For influence estimation, we use probe sets of $|\mathcal{D}_{\text{probe}}| =$ 1,000 samples, and we train ResNet-18 (He et al., 2016) models for 10 epochs. Further results with VGG-16 architecture (Simonyan & Zisserman, 2015) are reported in Appendix G. After removing the suspected poisoned samples, we train the final ResNet-18 model from scratch for 200 epochs. We train models based on cross-entropy loss. For optimization, we use SGD with a momentum

of 0.9, a weight decay of 5×10^{-4}, and a batch size of 128. The learning rate is scheduled using cosine annealing over the 200 training epochs, with an initial learning rate of 0.1. We conduct all experiments on a server with NVIDIA RTX A5000 GPUs with 24GB GPU memory. We provide the source code of our implementation as supplementary material to support reproducibility.

Attack Methods and Settings We consider five clean-label backdoor attacks: **Narcissus** (Zeng et al., 2023), **COMBAT** (Huynh et al., 2024), and three variants of **Wicked** (Nguyen et al., 2025). Narcissus uses optimized clean-label perturbations, COMBAT constructs input-aware clean-label perturbations, and Wicked strengthens trigger-based attacks through selective poisoning. We instantiate Wicked with patch-based, blended, and signal-based triggers, denoted Wicked-BadNets, Wicked-Blended, and Wicked-SIG, respectively (Gu et al., 2019; Chen et al., 2017; Barni et al., 2019). For each attack, we construct ten poisoned datasets by varying the target class from class 0 to 9. We poison 0.5% of the CIFAR-10 training set and 0.03–0.3% of the GTSRB training set, which corresponds to poisoning 5% of the samples in the target class. The overall rate varies across GTSRB target classes due to its class imbalance. For attacks with an ℓ_∞ perturbation constraint (i.e., budget), we set the perturbation constraint to $\epsilon = 16$, a value commonly used in the literature.

5.2 Baseline Defenses

We compare TIDE with four dataset cleaning defenses that all aim to remove suspected poisoned samples before training. **Activation Clustering (AC)** (Chen et al., 2019) clusters feature representations within each class and removes samples from the minority cluster. **SPECTRE** (Hayase et al., 2021) uses robust covariance estimation to detect poisoned samples with anomalous spectral signatures. **Confusion Training** (Qi et al., 2023) trains with random labels and flags samples that remain highly confident, indicating reliance on backdoor shortcuts. **FLARE** (Hou et al., 2025), combines feature clustering from all hidden layers to identify poisoned samples across diverse trigger types.

5.3 Evaluation Metrics

We evaluate defense performance using four metrics. **True Positive Rate (TPR)** measures the fraction of actual poisoned samples correctly identified and removed by the defense. **False Positive Rate (FPR)** measures the fraction of clean samples incorrectly flagged as poisoned; a low FPR is critical for preserving model utility, as removing too many clean samples degrades accuracy. After the defense removes its flagged samples, we retrain the model on the sanitized dataset and report **Clean Accuracy (Acc)**, the test accuracy on the original clean test set, and **Attack Success Rate (ASR)**, the fraction of triggered test inputs misclassified to the attacker's target class. An effective defense should minimize ASR while maintaining high Acc, achieving both high TPR and low FPR.

5.4 Results

Tables 1 and 2 report detection and post-defense performance, averaged over 10 runs (mean $\pm$ standard deviation is reported in Appendix C). On CIFAR-10, TIDE reduces ASR from 0.9120 to 0.0792 for Narcissus and from 0.8402 to 0.1789 for COMBAT, with clean accuracies of 0.9660 and 0.9491, respectively. Wicked attacks remain more challenging, with residual ASRs ranging from 0.2480 to 0.4910. FLARE achieves a lower ASR but suffers from very low clean accuracy, making it impractical. On GTSRB, TIDE achieves the lowest ASR across all five attacks, ranging from 0.0049 to 0.0864. These results demonstrate a favorable mitigation–utility trade-off, while showing that filtering does not eliminate every backdoor. We also evaluate against an additional attack, PureGen (Bhat et al., 2024), separately in Appendix E, since this attack purifies the entire training set instead of detecting poisoned samples, so only post-defense performance can be compared.

Hyperparameters and Ablation Appendix D studies the sensitivity of TIDE to its hyperparameters and design choices (probe-set size $|\mathcal{D}_{\text{probe}}|$, repeated partitions, and influence estimators), along with further ablations. We compare our method to classical influence-function-based detection in Appendix D.2, which shows that classical influence signals are noisier for poisoned data detection.

Additional Dataset and Architecture We extend our evaluation to VGG-16 on CIFAR-10 and ResNet-18 on Tiny ImageNet; complete results and per-setting baseline comparisons are reported in Appendix G. Across all five attacks in each setting, TIDE achieves the lowest ASR among the

Table 1: Detection performance under different attacks and defenses. For each attack, the best result is shown in **bold**, and the second-best result is underlined. For defenses that do not perform poison detection, TPR and FPR are reported as 0 and excluded from ranking. Results are averaged over ten runs. Note that FLARE's high TPR comes at the cost of an excessively high FPR.

Attack	TIDE (Ours)		Activation Clustering		SPECTRE		Confusion Training		FLARE	
	TPR ↑	FPR ↓	TPR ↑	FPR ↓	TPR ↑	FPR ↓	TPR ↑	FPR ↓	TPR ↑	FPR ↓
CIFAR-10										
Narcissus	0.8472	**0.2696**	0.8420	0.5528	0.5980	0.4989	0.8000	0.4902	**1.0000**	0.9069
COMBAT	0.6352	**0.0729**	0.0000	0.0000	0.0664	0.2470	0.1000	0.3597	**0.8620**	0.6523
Wicked-BadNets	0.5564	**0.1097**	0.0000	0.0000	0.2800	0.5011	0.4100	0.2460	**0.9224**	0.9169
Wicked-Blended	0.5840	**0.0990**	0.0000	0.0000	0.2300	0.4995	0.4000	0.1573	**1.0000**	0.9240
Wicked-SIG	0.5030	**0.0495**	0.0000	0.0000	0.3500	0.5015	0.1310	0.1495	**0.9852**	0.9551
GTSRB										
Narcissus	**1.0000**	**0.0809**	0.6668	0.1621	0.5775	0.6530	0.6530	0.2662	0.4848	0.3949
COMBAT	**1.0000**	**0.1218**	0.0000	0.0000	0.3320	0.2476	0.4288	0.2664	0.9784	0.4704
Wicked-BadNets	**1.0000**	0.3994	0.8666	**0.0891**	0.6225	0.4999	0.7696	0.4750	0.9400	0.6206
Wicked-Blended	**1.0000**	**0.0802**	0.2000	0.1161	0.5333	0.5001	0.6007	0.2306	**1.0000**	0.5516
Wicked-SIG	**0.9962**	0.1564	0.5333	**0.0747**	0.8667	0.4998	0.6252	0.3776	0.8682	0.8224

Table 2: Post-defense model performance under different attacks and defenses. For each attack, the best result is shown in **bold**, and the second-best result is underlined. Results are averaged over ten runs. Note that FLARE's low ASR comes at the cost of excessively low clean accuracy.

Attack	No Defense		TIDE (Ours)		Active Clustering		SPECTRE		Confusion Training		FLARE	
	Acc ↑	ASR ↓	Acc ↑	ASR ↓	Acc ↑	ASR ↓	Acc ↑	ASR ↓	Acc ↑	ASR ↓	Acc ↑	ASR ↓
CIFAR-10												
Narcissus	0.9521	0.9120	**0.9660**	0.0792	0.9023	**0.0431**	0.9045	0.1235	0.9149	0.0987	0.5364	0.0452
COMBAT	0.9482	0.8402	**0.9491**	0.1789	0.9477	0.8403	0.9342	0.6632	0.9168	0.3020	0.5829	**0.1713**
Wicked-BadNets	0.9467	0.8970	0.9413	0.3890	0.9469	0.8915	0.9425	0.8959	**0.9475**	0.8065	0.5507	**0.0699**
Wicked-Blended	0.9471	0.9383	**0.9451**	0.4910	0.9447	0.9381	0.9428	0.5608	0.9408	0.5675	0.4586	**0.1023**
Wicked-SIG	0.9451	0.9372	0.9419	0.2480	0.9407	0.9382	0.9400	0.7613	**0.9452**	0.7612	0.4941	**0.0717**
GTSRB												
Narcissus	0.9875	0.2309	**0.9902**	**0.0112**	0.9653	0.0589	0.9696	0.0948	0.9745	0.0554	0.9880	0.1353
COMBAT	0.9126	0.7955	0.9247	**0.0049**	0.9169	0.7917	**0.9305**	0.4480	0.9240	0.3776	0.9060	0.0597
Wicked-BadNets	0.9889	0.5993	**0.9918**	**0.0864**	0.9891	0.2831	0.9901	0.4075	0.9874	0.1337	0.8403	0.1640
Wicked-Blended	0.9870	0.5130	**0.9915**	**0.0360**	0.9913	0.4002	0.9910	0.4499	0.9859	0.1181	0.8942	0.0445
Wicked-SIG	0.9876	0.4899	**0.9978**	**0.0452**	0.9857	0.2876	0.9835	0.0828	0.9622	0.1731	0.8470	0.0969

evaluated defenses, while its clean accuracy remains within 0.0124 and 0.90 of the corresponding undefended models, respectively. On Tiny ImageNet, TIDE reduces ASR from 0.9899 to 0.2917 for Narcissus and from 0.2571 to 0.0133 for Wicked-BadNets.

Adaptive Attacks We formulate a TIDE-aware, adaptive variant of Narcissus that aims to reduce cross-influence among poisoned samples, making the shared influence patterns exploited by TIDE less pronounced. Under matched poisoning and perturbation budgets, the adaptive attack achieves an ASR of 1.0000 without defense and 0.1045 after TIDE, with a post-defense clean accuracy of 0.9051. Appendix H provides implementation details and results across various levels of adaptation.

6 Conclusion

We introduced TIDE, a novel data-cleaning defense for detecting data poisoning without a known-clean reference set. Our findings demonstrate that shared influence patterns provide a useful signal for identifying poisoned samples collectively, highlighting the value of examining relationships among training samples rather than assessing each sample in isolation. In all evaluations, *TIDE significantly reduces attack success while preserving test accuracy on clean data*, making it a practical defense. The evaluation of adaptive attacks also shows that attackers cannot suppress cross-influence among poisoned samples without significantly diminishing the effectiveness of their attacks.

Limitations Our experiments are limited to computer vision, which is the commonly used modality in data poisoning, e.g., BackdoorBench (Wu et al., 2022; 2025), FLARE (Hou et al., 2025), Wicked (Nguyen et al., 2025). Extending the evaluation to other modalities, such as audio or video, remains a direction for future work. TIDE incurs additional training time and storage costs (Table 6), and its attribution relies on a last-layer approximation rather than an exact counterfactual influence.

AI DISCLOSURE

In this paper, we used generative AI tools to improve the presentation of our work, including revising the manuscript text (e.g., checking grammar and improving clarity) and formatting LaTeX tables. We also used these tools for limited code cleanup and debugging in our implementation. Generative AI tools were not used to develop research ideas or methodology, design experiments, generate experimental data or results, or draw conclusions. We take responsibility for the final content of this work, including all text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the complete source code of our implementation of TIDE and our experiments as supplementary material; we will release the source code under open-source licenses upon publication. We provide complete pseudocode for influence estimation and dense subgraph search in Appendix B. We provide complete hyperparameter specifications for TIDE, the attacks, and the baseline defenses, including training configurations and poisoning rates, in Section 5.1 and Appendix G. All experiments use publicly available benchmark datasets and standard architectures.

ETHICS STATEMENT

This study uses only publicly available benchmark datasets and does not involve human subjects or personally identifiable information.

REFERENCES

Bilal Hussain Abbasi, Yanjun Zhang, Leo Zhang, and Shang Gao. Backdoor attacks and defenses in computer vision domain: A survey. *arXiv preprint arXiv:2509.07504*, 2025.

Naman Agarwal, Brian Bullins, and Elad Hazan. Second-order stochastic optimization for machine learning in linear time. *Journal of Machine Learning Research*, 18(116):1–40, 2017.

Yang Bai, Gaojie Xing, Hongyan Wu, Zhihong Rao, Chuan Ma, Shiping Wang, Xiaolei Liu, Yimin Zhou, Jiajia Tang, Kaijun Huang, et al. Backdoor attack and defense on deep learning: A survey. *IEEE Transactions on Computational Social Systems*, 12(1):404–434, 2024.

Mauro Barni, Kassem Kallas, and Benedetta Tondi. A new backdoor attack in cnns by training set corruption without label poisoning. In *Proceedings of the 26th IEEE International Conference on Image Processing (ICIP)*, pp. 101–105. IEEE, September 2019.

Sunay Bhat, Jeffrey Jiang, Omead Pooladzandi, Alexander Branch, and Gregory Pottie. PureGen: Universal data purification for train-time poison defense via generative model dynamics. In *Proceedings of the 38th International Conference on Neural Information Processing Systems (NeurIPS)*, pp. 135380–135414, December 2024.

Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. In *Proceedings of the 2019 AAAI Workshop on Artificial Intelligence Safety, co-located with the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, January 2019.

Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

Yukun Chen, Shuo Shao, Enhao Huang, Yiming Li, Pin-Yu Chen, Zhan Qin, and Kui Ren. REFINE: Inversion-free backdoor defense via model reprogramming. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, April 2025.

Edward Chou, Florian Tramer, and Giancarlo Pellegrino. SentiNet: Detecting localized universal attacks against deep learning systems. In *Proceedings of the 2020 IEEE Security and Privacy Workshops (SPW)*, pp. 48–54. IEEE, May 2020.

Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. STRIP: A defence against trojan attacks on deep neural networks. In *Proceedings of the 35th Annual Computer Security Applications Conference (ACSAC)*, pp. 113–125, December 2019.

Yansong Gao, Bao Gia Doan, Zhi Zhang, Siqi Ma, Jiliang Zhang, Anmin Fu, Surya Nepal, and Hyoungshick Kim. Backdoor attacks and countermeasures on deep learning: A comprehensive review. *arXiv preprint arXiv:2007.10760*, 2020.

Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019. doi: 10.1109/ACCESS.2019.2909068.

Junfeng Guo, Yiming Li, Xun Chen, Hanqing Guo, Lichao Sun, and Cong Liu. SCALE-UP: An efficient black-box input-level backdoor detection via analyzing scaled prediction consistency. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*, May 2023.

Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*, 2023. URL `https://www.gurobi.com`.

Zayd Hammoudeh and Daniel Lowd. Simple, attack-agnostic defense against targeted training set attacks using cosine similarity. In *ICML 2021 Workshop on Uncertainty and Robustness in Deep Learning (UDL)*, July 2021.

Satoshi Hara, Atsushi Nitanda, and Takanori Maehara. Data cleansing for models trained with SGD. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*, December 2019.

Frank Hartle III, Steve Mancini, and Emily Kerry. Data poisoning 2018–2025: A systematic review of risks, impacts, and mitigation challenges. *Issues in Information Systems*, 25(4):433–442, 2025. doi: 10.48009/4_iis_2025_135.

Jonathan Hayase, Weihao Kong, Raghav Somani, and Sewoong Oh. SPECTRE: Defending against backdoor attacks using robust statistics. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pp. 4129–4139. PMLR, July 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Linshan Hou, Ruili Feng, Zhongyun Hua, Wei Luo, Leo Yu Zhang, and Yiming Li. IBD-PSC: input-level backdoor detection via parameter-oriented scaling consistency. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pp. 18992–19022, July 2024.

Linshan Hou, Wei Luo, Zhongyun Hua, Songhua Chen, Leo Yu Zhang, and Yiming Li. FLARE: Toward universal dataset purification against backdoor attacks. *IEEE Transactions on Information Forensics and Security*, 20:6459–6473, 2025. doi: 10.1109/TIFS.2025.3581719.

Tran Huynh, Dang Nguyen, Tung Pham, and Anh Tran. Combat: Alternated training for effective clean-label backdoor attacks. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI)*, volume 38, pp. 2436–2444, 2024.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pp. 1885–1894. PMLR, August 2017.

Alex Krizhevsky. Learning multiple layers of features from tiny images. University of Toronto, `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`, 2009.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR-10 dataset (Canadian Institute for Advanced Research). University of Toronto, `http://www.cs.toronto.edu/~kriz/cifar.html`, 2009.

Ya Le and Xuan Yang. Tiny ImageNet visual recognition challenge. Technical report, Stanford University, 2015. URL https://cs231n.stanford.edu/reports/2015/pdfs/yle_project.pdf. CS231n Course Project Report.

Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma. Neural attention distillation: Erasing backdoor triggers from deep neural networks. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, May 2021.

Yiming Li, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(1):5–22, 2024.

Weilin Lin, Li Liu, Jianze Li, and Hui Xiong. Fusing pruned and backdoored models: Optimal transport-based data-free backdoor mitigation. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI)*, volume 39, pp. 26299–26307, 2025.

Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-Pruning: Defending against backdooring attacks on deep neural networks. In *Proceedings of the 21st International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*, pp. 273–294. Springer, September 2018.

Xiaogeng Liu, Minghui Li, Haoyu Wang, Shengshan Hu, Dengpan Ye, Hai Jin, Libing Wu, and Chaowei Xiao. Detecting backdoors during the inference stage based on corruption robustness consistency. In *Proceedings of the 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16363–16372, June 2023.

Quang H Nguyen, Nguyen Ngoc-Hieu, Thanh Nguyen-Tang, Kok Seng Wong, Hoang Thanh-Tung, Khoa D Doan, et al. Wicked oddities: Selectively poisoning for effective clean-label backdoor attacks. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, pp. 3033–3055, 2025.

Tuan Anh Nguyen and Anh Tran. Input-aware dynamic backdoor attack. In *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 3454–3464, December 2020.

Minzhou Pan, Yi Zeng, Lingjuan Lyu, Xue Lin, and Ruoxi Jia. ASSET: Robust backdoor data detection across a multiplicity of deep learning paradigms. In *Proceedings of the 32nd USENIX Security Symposium (USENIX Security)*, pp. 2725–2742, August 2023.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*, pp. 8026–8037, December 2019.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. In *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)*, pp. 19920–19930, December 2020.

Xiangyu Qi, Tinghao Xie, Jiachen T Wang, Tong Wu, Saeed Mahloujifar, and Prateek Mittal. Towards a proactive ML approach for detecting backdoor poison samples. In *Proceedings of the 32nd USENIX Security Symposium (USENIX Security)*, pp. 1685–1702, August 2023.

Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden trigger backdoor attacks. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pp. 11957–11965, February 2020.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*, 2015.

Hossein Souri, Liam Fowl, Rama Chellappa, Micah Goldblum, and Tom Goldstein. Sleeper agent: Scalable hidden trigger backdoors for neural networks trained from scratch. In *Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS)*, pp. 19165–19178, December 2022.

Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *Proceedings of the 2011 International Joint Conference on Neural Networks (IJCNN)*, pp. 1453–1460, 2011.

Xiaofei Sun, Jiwei Li, Xiaoya Li, Ziyao Wang, Tianwei Zhang, Han Qiu, Fei Wu, and Chun Fan. A general framework for defending against backdoor attacks via influence graph. *arXiv preprint arXiv:2111.14309*, 2021.

Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS)*, pp. 8011–8021, December 2018.

Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Clean-label backdoor attacks. 2018.

Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *Proceedings of the 40th IEEE Symposium on Security and Privacy (IEEE S&P)*, pp. 707–723. IEEE, May 2019.

Baoyuan Wu, Hongrui Chen, Mingda Zhang, Zihao Zhu, Shaokui Wei, Danni Yuan, and Chao Shen. BackdoorBench: A comprehensive benchmark of backdoor learning. In *Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS), Track on Datasets and Benchmarks*, volume 35, pp. 10546–10559, 2022.

Baoyuan Wu, Hongrui Chen, Mingda Zhang, Zihao Zhu, Shaokui Wei, Danni Yuan, Mingli Zhu, Ruotong Wang, Li Liu, and Chao Shen. BackdoorBench: A comprehensive benchmark and analysis of backdoor learning. *International Journal of Computer Vision*, 133(8):5700–5787, 2025.

Jia-Li Yin, Weijian Wang, Wei Lin, Ximeng Liu, et al. Adversarial-inspired backdoor defense via bridging backdoor and adversarial attacks. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI)*, volume 39, pp. 9508–9516, 2025.

Danni Yuan, Mingda Zhang, Shaokui Wei, Li Liu, and Baoyuan Wu. Activation gradient based poisoned sample detection against backdoor attacks. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, April 2025.

Yi Zeng, Minzhou Pan, Hoang Anh Just, Lingjuan Lyu, Meikang Qiu, and Ruoxi Jia. Narcissus: A practical clean-label backdoor attack with limited information. In *Proceedings of the 30th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 771–785, November 2023.

Pu Zhao, Pin-Yu Chen, Payel Das, Karthikeyan Natesan Ramamurthy, and Xue Lin. Bridging mode connectivity in loss landscapes and adversarial robustness. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, April 2020.

A Empirical Evidence for the Assumption of Disproportionate Poison-to-Poison Influence

Our detection approach relies on the assumption that poisoned samples exert disproportionally strong mutual influence on each other during training relative to clean samples. In addition to demonstrating that our approach performs well in Section 5, we also test this assumption directly. We test the assumption on two backdoor attacks that differ in how the trigger is constructed: Narcissus, where all poisoned images share a single fixed trigger pattern, and COMBAT, where the trigger is synthesized per image from the input itself. For each attack, we compute accumulated influence scores across all four types of sample pairs (i.e., poison $\rightarrow$ poison, poison $\rightarrow$ clean, clean $\rightarrow$ poison, and clean $\rightarrow$ clean) and report the mean, median, standard deviation, and 95th percentile.

Table 3: Accumulated influence by sample-pair type on CIFAR-10.

Pair	Mean	Median	Std. dev.	95th pct.
Narcissus				
Poison $\rightarrow$ Poison	0.1193	0.0109	0.4080	0.5865
Poison $\rightarrow$ Clean	0.0415	0.0016	0.2094	0.1642
Clean $\rightarrow$ Poison	0.0089	0.0031	0.1022	0.0100
Clean $\rightarrow$ Clean	0.0053	0.0026	0.0530	0.0099
COMBAT				
Poison $\rightarrow$ Poison	0.6100	0.5746	0.3081	1.3138
Poison $\rightarrow$ Clean	0.5616	0.5114	0.3292	1.1697
Clean $\rightarrow$ Poison	0.2739	0.1715	0.3054	1.0526
Clean $\rightarrow$ Clean	0.3236	0.1895	0.3881	1.0830

Under Narcissus, the influence of poison $\rightarrow$ poison is the highest among all pair types, roughly $22\times$ higher than the clean $\rightarrow$ clean baseline in the mean and $59\times$ higher in the 95th percentile, indicating a significantly heavier high-influence tail between poison-poison pairs than between clean-clean pairs. COMBAT presents a more challenging case for our assumption due to its input-conditioned trigger, but our assumption holds nonetheless: poison $\rightarrow$ poison influence remains the strongest among all pair types, in both the mean and in the tail. However, the margin is narrower, and poison $\rightarrow$ clean influence has a comparable magnitude, which may be explained by the trigger direction being less consistent between poisoned samples when the trigger is synthesized per input. This narrower margin is why we detect at the subgraph level rather than scoring either individual samples or individual pairs or samples. TIDE's peeling procedure selects the subset that maximizes average influence density, rather than relying on any single pairwise gap; because high-influence pairs are concentrated among poisoned samples, the selected subset can remain internally dense even when the pairwise margin is modest. On COMBAT, this translates into a large empirical gain despite the weaker margin: attack success rate drops from 0.8402 to 0.1789 at a false-positive rate of 0.0729.

Mechanistic Explanation of the Assumption The trigger acts as a shared low-loss direction across poisoned samples: each gradient update on a poisoned sample moves the model along this direction, lowering the loss on other poisoned samples more than on clean ones. This predicts that poison $\rightarrow$ poison influence should exceed the clean $\rightarrow$ clean baseline, consistent with the above measurements. The effect is pronounced under Narcissus, where the trigger pattern is fixed and shared across all poisoned samples, and weaker under COMBAT, where the input-conditioned trigger makes this direction less consistent from one poisoned sample to the next.

B Algorithm Pseudocode

This section provides detailed pseudocode for the two main components of our framework, *influence estimation* (Algorithm 1) and *dense subgraph search* (Algorithm 2).

Algorithm 1 describes the procedure for estimating source-to-probe influence scores. During training, we track how each mini-batch update changes model outputs on a held-out probe set. These

output changes are attributed to individual source samples using a last-layer approximation based on feature similarity and prediction error. The resulting per-step contributions are accumulated across training steps to form a pairwise influence matrix I, where each entry I_{ij} measures the influence of a source sample x_i on a probe sample x_j.

Algorithm 1 Influence Estimation

Input: source set $\mathcal{D}_{\text{src}}$, probe set $\mathcal{D}_{\text{probe}}$, trainable model f_θ, number of epochs E
Initialize influence matrix $I \in \mathbb{R}^{|\mathcal{D}_{\text{src}}| \times |\mathcal{D}_{\text{probe}}|}$ to all zeros
Initialize model parameters θ_0
Compute initial model outputs $f_{\theta_0}(x_j)$ for $x_j \in \mathcal{D}_{\text{probe}}$
for epoch $e = 1, \ldots, E$ **do**
  for mini-batch $B_t \subset \mathcal{D}_{\text{src}}$ **do**
    Compute model outputs $f_{\theta_t}(x_i)$ and last-layer features $h_i^{(t)}$ for all $x_i \in B_t$
    Update model parameters by gradient descent:

$$\theta_{t+1} \leftarrow \theta_t - \eta_t \nabla_\theta \ell(B_t; \theta_t)$$

    Compute updated probe outputs $f_{\theta_{t+1}}(x_j)$ and probe features $h_j^{(t)}$ for all $x_j \in \mathcal{D}_{\text{probe}}$
    Compute probe-output shift for each $x_j \in \mathcal{D}_{\text{probe}}$:

$$\Delta_j^{(t)} = f_{\theta_{t+1}}(x_j) - f_{\theta_t}(x_j)$$

    For each $x_i \in B_t$, $x_j \in \mathcal{D}_{\text{probe}}$, and class c, compute attribution weight:

$$w_{ijc}^{(t)} = \frac{h_i^{(t)} \cdot h_j^{(t)}}{\|h_i^{(t)}\| \, \|h_j^{(t)}\|} \left(\text{softmax}(f_{\theta_t}(x_i))[c] - \mathbf{I}[y_i = c]\right)$$

    Normalize attribution weights:

$$\tilde{w}_{ijc}^{(t)} = \frac{w_{ijc}^{(t)}}{\sum_{i' \in B_t} |w_{i'jc}^{(t)}|}$$

    Update influence scores:

$$I_{ij} \leftarrow I_{ij} + \sum_{c=1}^{C} \left| \tilde{w}_{ijc}^{(t)} \Delta_j^{(t)}[c] \right|$$

  end for
end for
Output: influence matrix I

Algorithm 2 presents the dense subgraph search procedure, which operates on the influence matrix I. Starting from the complete set of source and probe samples, the algorithm iteratively removes samples, greedily maximizing the objective function J. At each step, removal is performed on either the source or probe side to maintain balance between the two sets. The algorithm keeps track of the pair of subsets that achieve the highest objective value, which corresponds to a dense subgraph that captures strongly interacting samples.

Together, these two algorithms enable the efficient identification of subsets of samples that exhibit disproportionally strong mutual influence, which we leverage to detect and remove poisoned data.

C Statistical Variability of Main Results

We report complete detection and post-defense results as mean $\pm$ standard deviation across ten runs in Tables 1 and 2.

Algorithm 2 Dense Subgraph Search

Input: Influence matrix $I \in \mathbb{R}^{N_t \times P}$
Initialize selected and removed subsets: $(S_{\text{src}}, S_{\text{probe}}) \leftarrow (\mathcal{D}_{\text{src}}, \mathcal{D}_{\text{probe}})$ and $(\bar{S}_{\text{src}}, \bar{S}_{\text{probe}}) \leftarrow (\emptyset, \emptyset)$
Initialize optimal selected subsets: $(S^*_{\text{src}}, S^*_{\text{probe}}) \leftarrow (S_{\text{src}}, S_{\text{probe}})$
while $|S_{\text{src}}| > 0$ **and** $|S_{\text{probe}}| > 0$ **do**
  if $|\bar{S}_{\text{src}}|/|\mathcal{D}_{\text{src}}| \leq |\bar{S}_{\text{probe}}|/|\mathcal{D}_{\text{probe}}|$ **then**
    Select the source sample x_i^* whose removal would lead to the highest objective value:
    $x_i^* \leftarrow \arg\max_{x_i \in S_{\text{src}}} J(S_{\text{src}} \setminus \{x_i\}, S_{\text{probe}}) - J(S_{\text{src}}, S_{\text{probe}})$
    Remove sample x_i^*:
    $S_{\text{src}} \leftarrow S_{\text{src}} \setminus \{x_i^*\}$, $\bar{S}_{\text{src}} \leftarrow \bar{S}_{\text{src}} \cup \{x_i^*\}$
  else $|\bar{S}_{\text{src}}|/|\mathcal{D}_{\text{src}}| > |\bar{S}_{\text{probe}}|/|\mathcal{D}_{\text{probe}}|$
    Select the probe sample x_j^* whose removal would lead to the highest objective value:
    $x_j^* \leftarrow \arg\max_{x_j \in S_{\text{probe}}} J(S_{\text{src}}, S_{\text{probe}} \setminus \{x_j\}) - J(S_{\text{src}}, S_{\text{probe}})$
    Remove sample x_j^*:
    $S_{\text{probe}} \leftarrow S_{\text{probe}} \setminus \{x_j^*\}$, $\bar{S}_{\text{probe}} \leftarrow \bar{S}_{\text{probe}} \cup \{x_j^*\}$
  end if
  if $J(S_{\text{src}}, S_{\text{probe}}) > J^* \leftarrow J(S^*_{\text{src}}, S^*_{\text{probe}})$ **then**
    Update optimal selected subsets:
    $(S^*_{\text{src}}, S^*_{\text{probe}}) \leftarrow (S_{\text{src}}, S_{\text{probe}})$
  end if
end while
Output: Optimal subsets S^*_{src} and S^*_{probe}

Table 4: Detection performance on CIFAR-10 and GTSRB, reported as mean ± standard deviation across ten runs. For each attack, the best result is shown in **bold**, and the second-best result is underlined. For defenses that do not perform poison detection, TPR and FPR are reported as 0 and excluded from ranking. Note that FLARE's high TPR comes at the cost of an excessively high FPR.

Attack	TIDE (Ours)		Activation Clustering		SPECTRE		Confusion Training		FLARE	
	TPR ↑	FPR ↓	TPR ↑	FPR ↓	TPR ↑	FPR ↓	TPR ↑	FPR ↓	TPR ↑	FPR ↓
CIFAR-10										
Narcissus	0.8472 ± 0.0019	**0.2696** ± 0.0020	0.8420 ± 0.0025	0.5528 ± 0.0085	0.5980 ± 0.0134	0.4989 ± 0.1244	0.8000 ± 0.0277	0.4902 ± 0.0219	**1.0000** ± 0.0000	0.9069 ± 0.0005
COMBAT	0.6352 ± 0.0042	**0.0729** ± 0.0073	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0664 ± 0.0025	0.2470 ± 0.0061	0.1000 ± 0.0013	0.3597 ± 0.0038	**0.8620** ± 0.0193	0.6523 ± 0.0131
Wicked-BadNets	0.5564 ± 0.0121	**0.1097** ± 0.0077	0.0000 ± 0.0000	0.0000 ± 0.0000	0.2800 ± 0.0583	0.5011 ± 0.0032	0.4100 ± 0.3150	0.2460 ± 0.0190	**0.9224** ± 0.0010	0.9169 ± 0.0336
Wicked-Blended	0.5840 ± 0.0379	**0.0990** ± 0.0006	0.0000 ± 0.0000	0.0000 ± 0.0000	0.2300 ± 0.0543	0.4995 ± 0.0031	0.4000 ± 0.0044	0.1573 ± 0.0027	**1.0000** ± 0.0000	0.9240 ± 0.0111
Wicked-SIG	0.5030 ± 0.0264	**0.0495** ± 0.0012	0.0000 ± 0.0000	0.0000 ± 0.0000	0.3500 ± 0.0767	0.5015 ± 0.0041	0.1310 ± 0.0231	0.1495 ± 0.0031	**0.9852** ± 0.0702	0.9551 ± 0.0472
GTSRB										
Narcissus	**1.0000** ± 0.0000	**0.0809** ± 0.0026	0.6668 ± 0.0036	0.1621 ± 0.0007	0.5775 ± 0.0204	0.6530 ± 0.0151	0.6530 ± 0.0651	0.2662 ± 0.0164	0.4848 ± 0.0125	0.3949 ± 0.0258
COMBAT	**1.0000** ± 0.0000	**0.1218** ± 0.0037	0.0000 ± 0.0000	0.0000 ± 0.0000	0.3320 ± 0.0050	0.2476 ± 0.0115	0.4288 ± 0.0268	0.2664 ± 0.0128	0.9784 ± 0.0038	0.4704 ± 0.0002
Wicked-BadNets	**1.0000** ± 0.0000	0.3994 ± 0.0152	0.8666 ± 0.1440	**0.0891** ± 0.0175	0.6225 ± 0.0826	0.4999 ± 0.0004	0.7696 ± 0.1334	0.4750 ± 0.0185	0.9400 ± 0.0568	0.6206 ± 0.0176
Wicked-Blended	**1.0000** ± 0.0000	**0.0802** ± 0.0118	0.2000 ± 0.0175	0.1161 ± 0.0606	0.5333 ± 0.1333	0.5001 ± 0.0006	0.6007 ± 0.0085	0.2306 ± 0.0087	**1.0000** ± 0.0000	0.5516 ± 0.0089
Wicked-SIG	**0.9962** ± 0.0004	0.1564 ± 0.0015	0.5333 ± 0.2497	**0.0747** ± 0.0247	0.8667 ± 0.1136	0.4998 ± 0.0005	0.6252 ± 0.0170	0.3776 ± 0.0185	0.8682 ± 0.0590	0.8224 ± 0.0144

Table 5: Post-defense clean accuracy and attack success rate on CIFAR-10 and GTSRB, reported as mean ± standard deviation acorss ten runs. For each attack, the best result is shown in **bold**, and the second-best result is underlined. Note that FLARE's low ASR comes at the cost of excessively low clean accuracy.

Attack	No Defense		TIDE (Ours)		Active Clustering		SPECTRE		Confusion Training		FLARE	
	Acc ↑	ASR ↓	Acc ↑	ASR ↓	Acc ↑	ASR ↓	Acc ↑	ASR ↓	Acc ↑	ASR ↓	Acc ↑	ASR ↓
CIFAR-10												
Narcissus	0.9521 ± 0.0004	0.9120 ± 0.0010	**0.9660** ± 0.0007	0.0792 ± 0.0039	0.9023 ± 0.0020	**0.0431** ± 0.0025	0.9045 ± 0.0006	0.1235 ± 0.0290	0.9149 ± 0.0010	0.0987 ± 0.0061	0.5364 ± 0.0294	0.0452 ± 0.0011
COMBAT	0.9482 ± 0.0007	0.8402 ± 0.0034	**0.9491** ± 0.0005	0.1789 ± 0.0098	0.9477 ± 0.0153	0.8403 ± 0.0100	0.9342 ± 0.0004	0.6632 ± 0.0211	0.9168 ± 0.0015	0.3020 ± 0.0096	0.5829 ± 0.1297	**0.1713** ± 0.0456
Wicked-BadNets	0.9467 ± 0.0006	0.8970 ± 0.0035	0.9413 ± 0.0004	0.3890 ± 0.0054	0.9469 ± 0.0006	0.8915 ± 0.0067	0.9425 ± 0.0055	0.8959 ± 0.0174	**0.9475** ± 0.0049	0.8065 ± 0.0074	0.5507 ± 0.3928	**0.0699** ± 0.4694
Wicked-Blended	0.9471 ± 0.0002	0.9383 ± 0.0007	**0.9451** ± 0.0004	0.4910 ± 0.0117	0.9447 ± 0.0029	0.9381 ± 0.0014	0.9428 ± 0.0047	0.5608 ± 0.0758	0.9408 ± 0.0042	0.5675 ± 0.0054	0.4586 ± 0.0040	**0.1023** ± 0.0762
Wicked-SIG	0.9451 ± 0.0007	0.9372 ± 0.0018	0.9419 ± 0.0011	0.2480 ± 0.0022	0.9407 ± 0.0056	0.9382 ± 0.0262	0.9400 ± 0.0054	0.7613 ± 0.0475	**0.9452** ± 0.0041	0.7612 ± 0.0038	0.4941 ± 0.0354	**0.0717** ± 0.0206
GTSRB												
Narcissus	0.9875 ± 0.0006	0.2309 ± 0.0012	**0.9902** ± 0.0001	**0.0112** ± 0.0068	0.9653 ± 0.0012	0.0589 ± 0.0131	0.9696 ± 0.0008	0.0948 ± 0.0036	0.9745 ± 0.0007	0.0554 ± 0.0002	0.9880 ± 0.0006	0.1353 ± 0.0103
COMBAT	0.9126 ± 0.0005	0.7955 ± 0.0151	0.9247 ± 0.0031	**0.0049** ± 0.0008	0.9169 ± 0.0303	0.7917 ± 0.0179	**0.9305** ± 0.0116	0.4480 ± 0.0107	0.9240 ± 0.0076	0.3776 ± 0.0020	0.9060 ± 0.0274	0.0597 ± 0.0208
Wicked-BadNets	0.9889 ± 0.0019	0.5993 ± 0.0195	**0.9918** ± 0.0008	**0.0864** ± 0.0420	0.9891 ± 0.0009	0.2831 ± 0.0482	0.9901 ± 0.0004	0.4075 ± 0.0356	0.9874 ± 0.0021	0.1337 ± 0.0116	0.8403 ± 0.0920	0.1640 ± 0.0377
Wicked-Blended	0.9870 ± 0.0017	0.5130 ± 0.1043	**0.9915** ± 0.0017	**0.0360** ± 0.0010	0.9913 ± 0.0016	0.4002 ± 0.0022	0.9910 ± 0.0014	0.4499 ± 0.1056	0.9859 ± 0.0016	0.1181 ± 0.0023	0.8942 ± 0.1238	0.0445 ± 0.0109
Wicked-SIG	0.9876 ± 0.0015	0.4899 ± 0.0574	**0.9978** ± 0.0017	**0.0452** ± 0.0016	0.9857 ± 0.0017	0.2876 ± 0.0485	0.9835 ± 0.0020	0.0828 ± 0.0232	0.9622 ± 0.0015	0.1731 ± 0.0615	0.8470 ± 0.2183	0.0969 ± 0.0910

D Ablation Study

D.1 Effect of Probe Set Size and Repetitions

In this section, we examine how the size of the probe set $|\mathcal{D}_{\text{probe}}|$ and the number of repetitions affect detection performance and computational cost. We perform this analysis on CIFAR-10 poisoned by the Narcissus attack.

At each gradient-descent step, our method requires only one additional forward pass over the probe set to measure the change in probe logits. Therefore, the additional computational overhead scales approximately linearly with $|\mathcal{D}_{\text{probe}}|$.

Table 6 reports detection performance and runtime across different probe-set sizes on a poisoned dataset with target class set to 0. As the probe size increases, the true positive rate consistently improves, indicating that larger probe sets provide more informative signals for identifying poisoned samples. However, this comes at the cost of a higher false positive rate, suggesting that overly large probe sets may introduce noise and lead to less selective detection. In addition, the runtime increases nearly linearly with $|\mathcal{D}_{\text{probe}}|$, confirming the expected computational scaling.

Table 6: Runtime under different probe-set sizes. The additional cost mainly comes from the forward pass over the probe set $\mathcal{D}_{\text{probe}}$ at each gradient-descent step. The setting used in the main experiments is shown in bold.

Probe Size $\lvert\mathcal{D}_{\text{probe}}\rvert$	**TPR**	**FPR**	**Runtime (minutes)**
500	0.63	0.19	≈ 4
1,000	0.70	0.22	≈ 8
2,000	0.79	0.30	≈ 16
5,000	0.86	0.40	≈ 39
10,000	0.92	0.50	≈ 78
20,000	0.96	0.65	≈ 152

We also examine the effect of varying the number of repetitions. Table 7 shows the effect of increasing the number of repetitions using disjoint probe sets. Increasing the number of repetitions could improve detection stability, leading to a gradual increase in TPR and a reduction in FPR. This suggests that aggregating influence estimates across multiple probe sets could help reduce variance and suppress spurious signals.

Table 7: Effect of the number of randomly sampled source–probe partitions on detection performance. The setting used in the main experiments is shown in bold.

# of Reps	**TPR**	**FPR**
1	0.61	0.20
2	0.70	0.22
5	0.73	0.20
10	0.76	0.14

Based on these results, we select a probe size of $|\mathcal{D}_{\text{probe}}| = 1{,}000$ and repeat it twice for the main experiments. For the probe size, we choose $|\mathcal{D}_{\text{probe}}| = 1{,}000$ because it provides a reasonable balance between detection performance and computational cost. Although larger probe sets further improve TPR, they also increase FPR and runtime substantially. For the number of repetitions, increasing the number of repetitions from 1 to 2 yields a clear improvement in TPR, while further increases provide only a gradual increase. Therefore, we use 2 as a practical choice that improves detection performance without adding excessive computational overhead.

D.2 Comparison with Classical Influence Functions

As an ablation study, we compare our proposed influence estimation method with classical influence functions (Koh & Liang, 2017), which approximate the effect of removing a training sample[1] on the loss of a test sample using an inverse Hessian-vector product.

Given a trained model with parameters θ, the classical influence score of a training (or source) sample x_i on a test (or probe) sample x_j is defined as the change in loss at x_j when x_i is removed during training. This can be approximated as

$$I(x_i, x_j) \approx -\nabla_\theta \ell(x_j, \theta)^\top H_\theta^{-1} \nabla_\theta \ell(x_i, \theta), \tag{8}$$

[1]Classical influence functions are often defined in terms of upweiging a sample during training; removal of the sample corresponds to upweighing by a factor of -1.

where $\ell(\cdot, \theta)$ is the loss function and H_θ is the Hessian of the empirical risk with respect to θ.

To obtain a source-to-probe influence matrix comparable to our method, we compute pairwise influence scores between randomly split $\mathcal{D}_{\text{src}}$ and $\mathcal{D}_{\text{probe}}$ using this formulation. In practice, we can approximate the inverse Hessian-vector product using stochastic estimation methods, such as LiSSA (Agarwal et al., 2017), which is a common approach.

We note that classical influence functions rely on strong assumptions, including local convexity and access to second-order information. In addition, the computational cost of estimating influence scores is expensive in large-scale settings.

This comparison allows us to evaluate whether our stepwise, training-dynamics-based influence estimation provides a more reliable and scalable alternative in the context of poisoned data detection.

Table 8 reports the detection results of two influence estimation approaches on a CIFAR-10 poisoned by the Narcissus attack. We use a probe set size of 1,000 and repeat the influence estimation over two disjoint probe sets.

Table 8: Detection results of different influence estimation methods on a CIFAR-10 poisoned by the Narcissus attack.

Influence Estimation	**TPR**	**FPR**
Classical Influence Function	0.96	0.68
Ours	0.70	0.22

The results show that classical influence functions achieve a high true positive rate, indicating that many poisoned samples are successfully identified. However, this comes at the cost of a substantially higher false positive rate, suggesting that a large number of clean samples are incorrectly flagged as poisoned.

In contrast, our method achieves a lower TPR but significantly reduces the FPR to 0.22, resulting in a more precise identification of poisoned samples. This trade-off indicates that while classical influence functions tend to produce noisy influence scores, our approach yields more reliable and selective detection signals. Therefore, our training-dynamics-based influence estimation provides a more robust alternative to classical influence functions for poisoned data detection, particularly in reducing samples being sacrificed.

E Comparison with Dataset Purification Defense

Unlike the defense methods evaluated in the main experiments, which aim to identify and discard suspicious samples, PureGen (Bhat et al., 2024) is a data purification approach that modifies the training dataset $\mathcal{D}$ to mitigate the effect of the backdoor trigger. In this setting, the entire dataset $\mathcal{D}$ is transformed into a purified version prior to final training, rather than selectively removing suspicious data points. Table 9 compares TIDE to PureGen against the Narcissus attack on the two benchmark datasets. For consistency with the main experiments, we report average results over 10 poisoned datasets generated by varying the target class from 0 to 9.

Table 9: Comparison of TIDE and PureGen against the Narcissus attack on benchmark datasets.

Dataset	**Defense**	**Acc.**	**ASR**
CIFAR-10	No Defense	0.9521	0.9120
	TIDE	0.9660	0.0792
	PureGen-DDPM	0.9138	0.0587
GTSRB	No Defense	0.9875	0.2309
	TIDE	0.9902	0.0112
	PureGen-DDPM	0.9484	0.1333

The results show that both approaches reduce ASR compared to the undefended baseline, but they exhibit different trade-offs between attack mitigation and clean accuracy. In CIFAR-10, PureGen-

DDPM achieves a lower ASR, reducing the attack success rate from 0.9120 to 0.0587. However, this reduction comes with a decrease in clean accuracy, from 0.9521 without defense to 0.9138 after purification. In contrast, TIDE reduces ASR to 0.0792 while achieving higher clean accuracy than both *No Defense* and PureGen-DDPM. This suggests that selective data cleaning can preserve clean-data utility more effectively than transforming the entire training set.

In GTSRB, TIDE achieves both a higher clean accuracy and a lower ASR than PureGen-DDPM. Specifically, it reduces ASR from 0.2309 to 0.0112 while maintaining an accuracy of 0.9902. PureGen-DDPM also reduces ASR compared to undefended one, but its attack success rate remains higher and its clean accuracy decreases to 0.9484. These results indicate that dataset purification may be effective when the generative purification process removes trigger-related artifacts, but it can also alter clean semantic features and reduce accuracy.

This comparison shows the distinction between purification-based and filtering-based defenses. PureGen modifies all training samples and can strongly suppress attacks, but may introduce distributional changes that affect clean accuracy. By contrast, TIDE directly ranks suspicious samples and removes only a small subset of the training data, leading to a more favorable balance between ASR reduction and clean-accuracy preservation.

F Alternative Approaches for Dense Subgraph Search

In this section, we compare the proposed *greedy peeling* to two alternative approaches for dense subgraph search: *integer programming* (Section F.1) and *simulated annealing* (Section F.2). We present experimental results that compare these different approaches in Section F.3.

F.1 Integer Programming

Finding a maximum-density bipartite subgraph is generally NP-hard. Moreover, the objective $J(S_{\text{src}}, S_{\text{probe}})$ is non-linear because the normalization terms depend on the subset sizes $|S_{\text{src}}|$ and $|S_{\text{probe}}|$. To find optimal dense subgraphs using integer programming, we fix the subset sizes $|S_{\text{src}}|$ and $|S_{\text{probe}}|$ and solve for the resulting linear objective.

Let $u_i \in \{0,1\}$ indicate whether a source sample $x_i \in \mathcal{D}_{\text{src}}$ is included in selection S_{src}, and let $v_j \in \{0,1\}$ indicate whether a probe sample $x_j \in \mathcal{D}_{\text{probe}}$ is included in selection S_{probe}. The removed subsets $\bar{S}_{\text{src}}$ and $\bar{S}_{\text{probe}}$ are implicitly represented by $(1-u_i)$ and $(1-v_j)$, respectively.

The optimization problem is given by

$$\begin{aligned}\max_{\boldsymbol{u},\boldsymbol{v}} \quad & \frac{\sum_{x_i\in\mathcal{D}_{\text{src}}}\sum_{x_j\in\mathcal{D}_{\text{probe}}} I(x_i,x_j)\cdot u_i\cdot v_j}{|S_{\text{src}}|\,|S_{\text{probe}}|} \\ & - \frac{\sum_{x_i\in\mathcal{D}_{\text{src}}}\sum_{x_j\in\mathcal{D}_{\text{probe}}} I(x_i,x_j)\cdot u_i\cdot (1-v_j)}{|S_{\text{src}}|\,|\bar{S}_{\text{probe}}|} \\ & - \frac{\sum_{x_i\in\mathcal{D}_{\text{src}}}\sum_{x_j\in\mathcal{D}_{\text{probe}}} I(x_i,x_j)\cdot (1-u_i)\cdot v_j}{|\bar{S}_{\text{src}}|\,|S_{\text{probe}}|}\end{aligned} \tag{9}$$

subject to

$$\sum_{x_i\in\mathcal{D}_{\text{src}}} u_i = |S_{\text{src}}|, \qquad \sum_{x_j\in\mathcal{D}_{\text{probe}}} v_j = |S_{\text{probe}}|, \qquad u_i, v_j \in \{0,1\}. \tag{10}$$

Since the objective contains products of binary variables (e.g., $u_i, \cdot v_j$), our implementation uses a standard linearization of binary products to linearize the objective. For example, we replace each product $u_i \cdot v_j$ with an auxiliary binary variable that is constrained to be equal to the logical *AND* of variables u_i and v_j. We apply the same linearization to $u_i \cdot (1-v_j)$ and $(1-u_i)\cdot v_j$ as well.

We solve this integer program using the Gurobi 11.0 solver Gurobi Optimization, LLC (2023) for each pair $(|S_{\text{src}}|, |S_{\text{probe}}|)$ within a search grid centered on the subset sizes identified by the greedy peeling algorithm. This exhaustive search provides a close lower bound for the subgraph search objective and provides a direct comparison with the greedy solution.

F.2 Simulated Annealing

We implement simulated annealing (SA) to explore the objective landscape beyond the removal-only search path of greedy peeling. The state space consists of the binary inclusion vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ for $\mathcal{D}_{\text{src}}$ and $\mathcal{D}_{\text{probe}}$, respectively. In each iteration, SA selects a sample at random from $\mathcal{D}_{\text{src}} \cup \mathcal{D}_{\text{probe}}$ and flips its state (i.e., includes the sample if it was removed, and removes it if it was included). SA accepts this proposed state transition at random according to the Metropolis criterion:

$$\Pr(\text{accept}) = \min\left(1, \exp\left(\frac{J(S'_{\text{train}}, S'_{\text{probe}}) - J(S_{\text{train}}, S_{\text{probe}})}{T_t}\right)\right), \tag{11}$$

where $(S_{\text{train}}, S_{\text{probe}})$ and $(S'_{\text{train}}, S'_{\text{probe}})$ denote the current and proposed states, respectively, and T_t is a temperature parameter that is lowered at each step t.

We use a geometric cooling schedule per-iteration with a temperature floor: $T_t = \max(T_{\min}, T_0\alpha^t)$, where $T_0 = 0.05$, $\alpha = 0.995$, and $T_{\min} = 10^{-6}$. Thus, the temperature is updated at every iteration, but clipped below by $T_{\min}$. The initial temperature is calibrated to ensure an initial acceptance rate of approximately 0.05 for deleterious moves. The algorithm is executed for $30{,}000$ iterations. The small gap observed between SA and greedy peeling (particularly in the Narcissus attack) suggests that the greedy solution is not significantly hindered by local optima (Table 10).

F.3 Experimental Results and Analysis

Table 10 shows performance results for two attacks, COMBAT and Narcissus, on CIFAR-10 ($N = 50{,}000$). For a fair comparison, we evaluate all three algorithms with a fixed selection size of 0.5% of the complete dataset. Results are averaged over 3 runs for COMBAT and 5 runs for Narcissus.

Table 10: Comparison of subgraph search algorithms in terms of optimality gap and runtime.

Attack	Method	Objective (J)	Gap to IP (%)	TPR	FPR	Runtime (minutes)
COMBAT	Integer Programming	0.4817	0.0%	0.3400	0.0485	≈ 68
	Simulated Annealing	0.4579	4.9%	0.3467	0.0485	5.5
	Greedy Peeling	**0.4250**	**11.8%**	**0.3467**	**0.0485**	**1.5**
Narcissus	Integer Programming	0.2248	0.0%	0.6344	0.0471	> 400
	Simulated Annealing	0.2035	9.5%	0.6696	0.0469	8.2
	Greedy Peeling	**0.2033**	**9.6%**	**0.6688**	**0.0469**	**1.5**

Discussion Two key insights emerge from these experimental results. First, although Greedy Peeling has a non-zero optimality gap in terms of the objective value J, this gap does not lead to worse detection performance under the fixed 0.5% selection size. Against COMBAT, Greedy Peeling achieves the same TPR and FPR as Simulated Annealing and the same FPR as Integer Programming. Against Narcissus, Greedy Peeling also closely matches Simulated Annealing, with nearly identical TPR and FPR. This indicates that for the purpose of detecting poisoned samples, the greedy solution is competitive despite not increasing the objective.

Second, Greedy Peeling has a significant advantage in terms of runtime. Integer Programming provides the exact optimum for each fixed-size problem, but its computational cost is substantially higher, requiring approximately 68 minutes for COMBAT and more than 400 minutes for Narcissus. In contrast, Greedy Peeling completes in approximately 1.5 minutes while providing comparable detection performance. For a forensic tool intended to run before final training or deployment, this efficiency makes Greedy Peeling a practical choice for datasets on a scale of $N \gg 50,000$.

Finally, we would like to highlight that Greedy Peeling provides a complete suspiciousness ranking over the samples (i.e., order in which they are iteratively removed from the selection), while the alternative algorithms output a single selection as the optimization solution (i.e., each sample is either included or not).

G Additional Dataset and Model Architecture

To broaden our evaluation, we extend the experiments to Tiny ImageNet (Le & Yang, 2015), a larger dataset, and VGG-16 (Simonyan & Zisserman, 2015), an architecture distinct from the ResNet-18

used in our main experiments. Specifically, we evaluate VGG-16 on CIFAR-10 and ResNet-18 on Tine ImageNet against the same five clean-label backdoor attacks. We compare TIDE with Activation Clustering, SPECTRE, and FLARE, reporting clean accuracy (Acc.) and attack success rate (ASR).

For the CIFAR-10 experiments, FLARE requires batch normalization layers, which are absent from standard VGG-16. Therefore, we use VGG-16-BN (Paszke et al., 2019) for FLASE instead of VGG-16. All methods use ResNet-18 in the Tiny ImageNet experiments.

Results on VGG-16 Table 11 shows that TIDE achieves the lowest ASR across all five attacks. Its clean accuracy decreases by only 0.0018 – 0.0090 relative to the corresponding undefended VGG-16 models. For Narcissus and COMBAT, TIDE reduces ASR from 1.0000 and 0.9580 to 0.1450 and 0.2710, respectively. Across the Wicked variants, TIDE achieves ASRs of 0.2540 – 0.3640, compared to 0.7110 – 0.8970 for Activation Clustering and 0.5040 – 0.6670 for SPECTRE.

FLARE, which is evaluated with VGG-16-BN model, reports similar ASRs for Narcissus and COMBAT but higher ASRs for the Wicked variants. TIDE also has higher clean accuracy than FLARE across all five attacks. We note that these direct comparisons should be interpreted with caution due to the difference in model architectures.

Table 11: Clean accuracy and attack success rate on CIFAR-10 with VGG-16, averaged over five runs. Note that FLARE uses VGG-16-BN to satisfy its batch normalization requirements. For all other methods, including undefended case, use VGG-16. For each attack, the highest clean accuracy and lowest ASR among the defenses are shown in **bold**.

	Undefended		TIDE (Ours)		Activation Clustering		SPECTRE		FLARE	
Attack	Acc.↑	ASR↓	Acc.↑	ASR↓	Acc.↑	ASR↓	Acc.↑	ASR↓	Acc.↑	ASR↓
Narcissus	0.9070	1.0000	0.9040	**0.1450**	**0.9070**	1.0000	0.9030	0.5080	0.9007	0.1494
COMBAT	0.9273	0.9580	**0.9228**	**0.2710**	0.9057	0.9120	0.9177	0.5560	0.9140	0.2740
Wicked-BadNets	0.9120	0.9680	**0.9030**	**0.3640**	0.9020	0.8860	0.8990	0.5040	0.8994	0.5907
Wicked-Blended	0.9110	0.8330	0.9092	**0.3490**	0.9060	0.7110	**0.9100**	0.6670	0.8107	0.3719
Wicked-SIG	0.9080	0.9140	**0.9050**	**0.2540**	0.9010	0.8970	0.9040	0.5040	0.8901	0.4678

Results on Tiny ImageNet Tiny ImageNet contains 100,000 training images and 10,000 validation images across 200 classes. For every attack, we poison 0.5% of the training set (250 images) and train ResNet-18 from scratch. Table 12 shows that TIDE achieves the lowest ASR among the evaluated defenses across all five attacks, with clean accuracy decreases of at most 0.0124 relative to the corresponding undefended models. For Narcissus, TIDE reduces ASR from 0.9899 to 0.2917, compared with 0.5914 for Activation Clustering, 0.9899 for SPECTRE, and 0.3830 for FLARE. For COMBAT, TIDE reduces ASR from 0.4811 to 0.1593. For Wicked-BadNets, Wicked-Blended, and Wicked-SIG, TIDE reduces ASR to 0.0133, 0.0536, and 0.0348, respectively. Compared with FLARE, TIDE achieves both lower ASR and higher clean accuracy across all five attacks. However, TIDE does not uniformaly achieve the highest clean accuracy. SPECTRE retains slightly higher accuracy for Narcissus and Wicked-BadNets.

Table 12: Clean accuracy and attack success rate on Tiny ImageNet with ResNet-18, averaged over five runs. All methods use the same architecture. For each attack, the highest clean accuracy and lowest ASR among the defenses are shown in **bold**.

	Undefended		TIDE (Ours)		Activation Clustering		SPECTRE		FLARE	
Attack	Acc.↑	ASR↓	Acc.↑	ASR↓	Acc.↑	ASR↓	Acc.↑	ASR↓	Acc.↑	ASR↓
Narcissus	0.5880	0.9899	0.5840	**0.2917**	0.5831	0.5914	**0.5844**	0.9899	0.3972	0.3830
COMBAT	0.5863	0.4811	**0.5739**	**0.1593**	0.4952	0.1818	0.5661	0.4680	0.4101	0.1696
Wicked-BadNets	0.5889	0.2571	0.5765	**0.0133**	0.5082	0.2448	**0.5865**	0.1992	0.4836	0.1764
Wicked-Blended	0.5838	0.3916	**0.5731**	**0.0536**	0.5457	0.1245	0.5433	0.1354	0.3839	0.0743
Wicked-SIG	0.5889	0.3470	**0.5774**	**0.0348**	0.5018	0.2809	0.5309	0.2885	0.3590	0.0447

H ADAPTIVE ATTACK

In this section, we examine whether an attacker who is aware of TIDE's detection can evade the defense while maintaining an effective backdoor trigger. To do this, we develop an adaptive variant of Narcissus attack and evaluate its effectiveness. We use Narcissus as the base attack because its optimization-based construction of a shared clean label trigger allows us to directly incorporate a penalty targeting cross-sample poison influence, while retaining the original trigger generation objective and perturbation constraints.

H.1 GENERATING ADAPTIVE ATTACK

We build a TIDE-aware variant of Narcissus that targets the shared influence structure exploited by our defense. On top of the original trigger generation objective that encourages a trigger–target association, our adaptive variant penalizes gradient alignment among poisoned target class samples. The adaptive attacker can therefore generate a successful backdoor while weakening the cross-sample relationships that enables TIDE to find poisoned samples.

Let c_{tar} denote the attacker-chosen target class, and let $\mathcal{D}_{\text{tar}}$ denote the target-class samples available to the attacker for trigger generation. This set is distinct from the defender's source and probe partitions, $\mathcal{D}_{\text{src}}$ and $\mathcal{D}_{\text{probe}}$. Let δ denote a shared additive trigger and $\mathcal{U}$ the set of triggers satisfying the perturbation constraints.

We use f_θ for the attacker's surrogate model in this subsection. Following the loss notation used elsewhere in the paper, $\ell(x_i + \delta, \theta)$ denotes the cross-entropy loss of f_θ on the triggered input, with the unchanged label $y_i = c_{\text{tar}}$. The original trigger-generation objective is

$$\delta^\star = \arg\min_{\delta \in \mathcal{U}} \sum_{x_i \in \mathcal{D}_{\text{tar}}} \ell(x_i + \delta, \theta). \tag{12}$$

Define the parameter gradient for a triggered sample as $g_i(\delta; \theta) = \nabla_\theta \ell(x_i + \delta, \theta)$.

We introduce the alignment penalty $\sum_{x_i \in \mathcal{D}_{\text{tar}}} \sum_{\substack{x_j \in \mathcal{D}_{\text{tar}} \\ i \neq j}} g_i(\delta; \theta)^\top g_j(\delta; \theta)$, where self-pairs are excluded.

The adaptive trigger is obtained by optimizing

$$\delta^\star = \arg\min_{\delta \in \mathcal{U}} \left[\sum_{x_i \in \mathcal{D}_{\text{tar}}} \ell(x_i + \delta, \theta) + \gamma \sum_{x_i \in \mathcal{D}_{\text{tar}}} \sum_{\substack{x_j \in \mathcal{D}_{\text{tar}} \\ i \neq j}} g_i(\delta; \theta)^\top g_j(\delta; \theta) \right]. \tag{13}$$

Here, $\gamma \geq 0$ controls the strength of the alignment penalty. Setting $\gamma = 0$ is equivalent to the original Narcissus objective.

Rationale for the Heuristic Directly optimizing against TIDE would require accounting for influence accumulation throughout the defender's training trajectory and the subsequent discrete peeling procedure. We instead use gradient inner products as a differentiable, local surrogate for cross-sample interaction.

For a hypothetical gradient-descent update induced by sample $x_i + \delta$, the first-order change in the loss on $x_j + \delta$ is

$$\ell(x_j + \delta, \theta - \eta g_i(\delta; \theta)) - \ell(x_j + \delta, \theta) \approx -\eta \, g_j(\delta; \theta)^\top g_i(\delta; \theta), \tag{14}$$

where η is the step size of the update. A positive inner product indicates that an update on one triggered sample locally decreases the loss on another. This motivates penalizing aggregate gradient alignment to weaken such interactions.

Note that the alignment penalty is not identical to TIDE's influence score $I(x_i, x_j)$: it measures signed loss-gradient alignment in a surrogate model, whereas TIDE accumulates magnitudes of attributed probe-logit changes along the training trajectory. Consequently, reducing the penalty does

not guarantee a reduction in TIDE's influence scores. Moreover, because the penalty uses signed gradient inner products, a lower value does not necessarily imply weaker cross-sample interactions. It may also reflect opposing gradient directions.

To optimize the adaptive objective, the trigger update uses

$$\nabla_\delta \left[\sum_{x_i \in \mathcal{D}_{\text{tar}}} \ell(x_i + \delta, \theta) + \gamma \sum_{x_i \in \mathcal{D}_{\text{tar}}} \sum_{\substack{x_j \in \mathcal{D}_{\text{tar}} \\ i \neq j}} g_i(\delta; \theta)^\top g_j(\delta; \theta) \right]. \tag{15}$$

Differentiating the alignment term requires mixed second-order derivatives through the parameter gradients.

H.2 Attack and defense performance

We evaluate the adaptive attack on CIFAR-10 with ResNet-18 and report results averaged over five runs. We use the same poisoning rate, perturbation budget, and TIDE hyperparameters as in the main experiments. Table 13 compares the adaptive variants with the original Narcissus attack. All evaluated variants achieve a pre-defense ASR of 1.0000, with clean accuracy between 0.9030 and 0.9044. Thus, the adaptive objective preserves the original Narcissus's attack effectiveness before filtering.

After applying TIDE, ASR decreases to 0.0261 – 0.1045 across the adaptive variants, compared to 0.0019 for the original Narcissus attack. TPR decreases from 0.9320 to 0.8245 – 0.8810, while FPR remains between 0.0469 and 0.1095, similar to the original attack's 0.0359. Compared to the original Narcissus attack, adaptive variants have lower detection rates and higher post-defense ASR, although TIDE still substantially reduces ASR from its pre-defense of 1.0000 while maintaining clean accuracy. Post-defense accuracy ranges from 0.9051 to 0.9065, with a decrease of at most 0.0016 relative to each variant's pre-defense accuracy.

Search over Adaptive Penalty Strengths We conduct a hyperparameter search over $\gamma \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ to examine the effect of the alignment penalty on adaptive attacks. Among the evaluated settings, all γ reach an attack success rate of 1.0000, but $\gamma = 10^{-3}$ shows the highest post-defense ASR of 0.1045 over an average of five runs, with a TPR of 0.8245 and an FPR of 0.1095. Notably, the effect is not linear in γ: setting $\gamma = 10^{-2}$ yields a post-defense ASR of 0.0503, while $\gamma = 10^{-1}$ yields 0.0871. So, increasing the penalty strength does not consistently improve evasion of TIDE.

Table 13: Adaptive Narcissus evaluation across penalty strengths γ. *Pre* and *post* denote performance before defense and after TIDE filtering and retraining, respectively. Results are averaged over five runs. The largest reported post-defense ASR (i.e., the most difficult case) among the adaptive variants is shown in **bold**.

γ	Pre Acc.	Pre ASR	TPR	FPR	Post Acc.	Post ASR
Narcissus	0.9066	1.0000	0.9320	0.0359	0.9037	0.0019
10^{-5}	0.9042	1.0000	0.8430	0.0469	0.9064	0.0261
10^{-4}	0.9030	1.0000	0.8810	0.0567	0.9056	0.0724
10^{-3}	0.9044	1.0000	0.8245	0.1095	0.9051	**0.1045**
10^{-2}	0.9036	1.0000	0.8810	0.0924	0.9065	0.0503
10^{-1}	0.9035	1.0000	0.8696	0.0655	0.9056	0.0871