META-SIFT: How to Sift Out a Clean Subset in the Presence of Data Poisoning?

Yi Zeng, *Virginia Tech and SONY AI;* Minzhou Pan, Himanshu Jahagirdar, and Ming Jin, *Virginia Tech;* Lingjuan Lyu, *SONY AI;* Ruoxi Jia, *Virginia Tech*

https://www.usenix.org/conference/usenixsecurity23/presentation/zeng

This paper is included in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

META-SIFT : How to Sift Out a Clean Subset in the Presence of Data Poisoning?

Yi Zeng*[1,2], Minzhou Pan*[1], Himanshu Jahagirdar[1], Ming Jin[1], Lingjuan Lyu[2] and Ruoxi Jia[1]

[1]Virginia Tech, Blacksburg, VA 24061, USA
[2]Sony AI, Tokyo, 108-0075, Japan

Abstract

External data sources are increasingly being used to train machine learning (ML) models as the data demand increases. However, the integration of external data into training poses data poisoning risks, where malicious providers manipulate their data to compromise the utility or integrity of the model. Most data poisoning defenses assume access to a set of clean data (referred to as the base set), which could be obtained through trusted sources. But it also becomes common that entire data sources for an ML task are untrusted (e.g., Internet data). In this case, one needs to identify a subset within a contaminated dataset as the base set to support these defenses.

This paper starts by examining the performance of defenses when poisoned samples are mistakenly mixed into the base set. We analyze five representative defenses that use base sets and find that their performance deteriorates dramatically with less than 1% poisoned points in the base set. These findings suggest that sifting out a base set with *high precision* is key to these defenses' performance. Motivated by these observations, we study how precise existing automated tools and human inspection are at identifying clean data in the presence of data poisoning. Unfortunately, neither effort achieves the precision needed that enables effective defenses. Worse yet, many of the outcomes of these methods are worse than random selection.

In addition to uncovering the challenge, we take a step further and propose a practical countermeasure, META-SIFT . Our method is based on the insight that existing poisoning attacks shift data distributions, resulting in high prediction loss when training on the clean portion of a poisoned dataset and testing on the corrupted portion. Leveraging the insight, we formulate a bilevel optimization to identify clean data and further introduce a suite of techniques to improve the efficiency and precision of the identification. Our evaluation shows that META-SIFT can sift a clean base set with 100% precision under a wide range of poisoning threats. The selected base set is large enough to give rise to successful defense when plugged into the existing defense techniques.

*M. Pan and Y. Zeng contributed equally. Corresponding **Y. Zeng**, **L. Lyu** or **R. Jia**. Work partially done during Y. Zeng's internship at Sony AI.

1 Introduction

Constructing high-performance machine learning (*ML*) systems requires large and diverse data. The data-hungry nature will inevitably force individuals and organizations to leverage data from external sources, the beginning of which is already evident. For instance, CLIP [1], the state-of-the-art image representation, is learned from 400 million image-text pairs collected from the Internet. Various data marketplaces and crowd-sourcing platforms also emerge to enable data exchange at scale. While incorporating external data sources into training has clear benefits, it exposes ML systems to security threats on account of data poisoning attacks, in which attackers modify training data to degrade model performance or control model prediction. In fact, data poisoning has been remarked as the top security concern regarding ML systems in the industry [2].

In this paper, the term "data poisoning" will be used in a broad sense, referring to attacks that involve training data manipulation. In particular, it includes both the attacks that interfere only with training data *and* backdoor attacks that embed a backdoor trigger during the training time and further inject the trigger into test-time inputs to control their corresponding predictions. Within the scope of this paper, we divide existing data poisoning attacks into three categories based on the attribute being manipulated:

- **Label-only attacks** that only alter labels, such as targeted [3] and Random Label-Flipping attacks [4] aimed at degrading model utility;
- **Feature-only attacks** that only manipulate features without changing the labels, such as feature collision attacks [5] and clean-label backdoor attacks [6, 7];
- **Label-Feature attacks** that change both feature and label, such as standard backdoor attacks [8–10].

Intensive efforts have been invested in mitigating data poisoning. The types of defenses in the prior work range from identifying poisoned samples in a training set [11] (**Poison Detection**) to detecting whether a model has been trained on a poisoned dataset [12] (**Trojan-Net Detection**) to removing backdoors from a poisoned model [13, 14] (**Backdoor**

Removal) to redesigning training algorithms to prevent poisoning from taking effect [4] (**Robust Training**)

Most existing defenses assume that *the defender can access a set of clean data* (referred to as the *base set* hereafter). Despite the prevalence of the assumption in existing literature, focused discussion about its validity is lacking. If the defender were capable of collecting a set of clean samples from trusted sources of data, then this assumption could be met easily. However, it has become increasingly common to learn solely from untrusted data sources, such as training with the data scraped from the Internet or purchased from specific vendors. In that case, the defender needs to identify a clean subset within the poisoned dataset to form the base set. *Many important questions remain unclear*:

How does the defense performance change if the identification is imperfect and some poisoned data are mixed into the base set? Are there any existing automated methods that can reliably identify a clean base set in the presence of various types of poisoning attacks? Can human inspection fulfill the need? If not, how can we reliably identify enough clean samples to support those defenses?

Takeaway #1: Defense performance is sensitive to the purity of the base set. We start by examining the sensitivity of defense performance to the ratio of poisoned points in the base set. We study five representative defense techniques that rely on access to a base set. The techniques considered either achieve state-of-the-art performance or are popular baselines. We find that their performance degrades significantly (e.g., attack success rate exceeding 80%) with less than 1% of poisoned points in the base set. Surprisingly, even a single poisoned point is sufficient to nullify the effect of a state-of-the-art poisoned data detector. These findings suggest that the ability to sift out a base set with *high precision* is critical to successfully applying these defenses.

Takeaway #2: Both existing automated methods and human inspection fail to identify a clean subset with high enough precision. We investigate how precise existing automated methods and human inspection can be in identifying clean data in the presence of data poisoning and the result is illustrated in Figure 1. The precision of both humans and existing automated methods varies a lot across different attack categories. Humans are proficient at identifying poisoned samples that involve label changes, including Label-only attacks and Label-Feature attacks, and outperform existing automated methods by a large margin. However, humans still miss many poisons and cannot realize a 100% success rate in sifting out a clean base set. Notably, for these two attack categories, several automated methods even underperform the random baseline.

On the other hand, for Feature-only attacks, human inspection results in a precision close to the random baseline. As these attacks inject small perturbations only to the features while not changing the overall semantics, human experts perform worse than most automated methods. *This finding is in direct contrast to the traditional wisdom that treats human supervision as the final backstop of data poisoning.* Besides being time-consuming and cost-intensive, human inspection becomes less trustworthy in identifying poisoned data given the fast-growing research on stealthy attacks. Overall, both existing automated methods and human inspection cannot reach the level of precision required to enable successful defense.

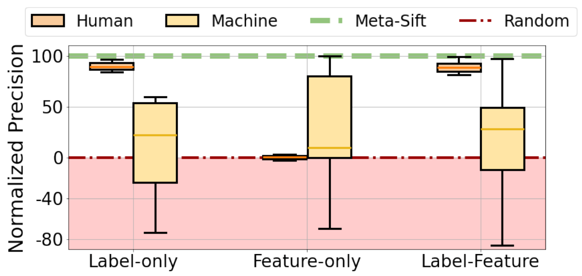


Figure 1: A comparison of the normalized precision of existing automated methods (Machine), Human, and META-SIFT in sifting out a clean subset from a poisoned CIFAR-10. Both human, machine-based, and META-SIFT results are normalized with the poison ratio to ensure comparability. A larger value indicates a stronger filtering capability. The **red** region depicts the filtering capability worse than random selection.

Takeaway #3: META-SIFT — a scalable and effective automated method to sift out a clean base set. We propose META-SIFT to sift a clean subset from the poisoned set. Our approach is based on a novel insight that data manipulation techniques exploited by existing poisoning attacks inevitably result in a distributional shift from the clean data. Hence, training on the clean portion of the contaminated dataset and testing the trained model on the other corrupted portion will lead to a high prediction loss. We formulate a bilevel optimization problem to split the contaminated dataset in a way that training on one split and testing on the other leads to the highest prediction loss. However, this splitting problem is hard to solve exactly as it has a combinatorial search space and at the same time, contains two nested optimization problems. To address the computational challenge, we first relax it into a continuous splitting problem, where we associate each sample with a continuous weight indicating their likelihood of belonging to one of the splits and then optimize the weights via gradient-based methods. Secondly, we adapt the online algorithm that was originally designed for training sample reweighting [15] to efficiently solve the continuous relaxation of the bilevel problem. Furthermore, we adopt the idea of "ensembling" to improve the precision of selection. In particular, we propose to apply random perturbations to each point, run the online algorithm on each perturbed version to obtain a weight, and aggregate the weights for final clean data selection. Our evaluation shows that META-SIFT can robustly sift out a clean base set with 100% precision under a wide range of poisoning attacks. The selected base set is large enough to give rise to successful defense when plugged into the existing defense techniques. It is worth noting that META-SIFT significantly outperforms the existing automated methods (illustrated in Figure 1) while being orders of magnitude faster (Table 5, 6, 15, 16).

Our contributions can be summarized as follows:

- We **identify an overlooked problem** of the accessibility of a clean base set in the presence of data poisoning.
- We **systematically evaluate the performance of existing automated methods and human inspection** in distinguishing between poisoned and clean samples;
- We **propose a novel splitting-based idea** to sift out a clean subset from a poisoned dataset and **formalize it into a bilevel optimization problem**.
- We **propose META-SIFT**, comprising an **efficient algorithm** to solve the bilevel problem as well as **a series of techniques to enhance sifting precision**.
- We **extensively evaluate META-SIFT** and compare with existing automated methods on four benchmark datasets under twelve different data poisoning attack settings. **Our method significantly outperforms existing methods** in both sifting **precision** and **efficiency**. At the same time, plugging our sifted samples into existing defenses achieves **comparable or even better performance** than plugging in randomly selected clean samples.
- We **open-source the project** to promote research on this topic and facilitate the successful application of existing defenses in settings without a clean base set [1].

2 Sifting Out a Clean Enough Base Set is Hard

The ability to acquire a clean base set was taken for granted in many existing data poisoning defenses [13, 14, 16–19]. For instance, a popular Trojan-Net Detection strategy is to first synthesize potential trigger patterns from a target model and then inspect whether there exists any suspicious pattern [13, 16]. Trigger synthesis is done by searching for a pattern that maximally activates a certain class output when it is patched onto the clean data. Hence, access to a clean set of data is indispensable to this defense strategy. Another example is defenses against Label-Flipping attacks (often referred to as mislabeled data detection in ML literature). State-of-the-art methods detect mislabeled data by finding a subset of instances such that when they are excluded from training, the prediction accuracy on a clean validation set is maximized. A clean set of instances are needed to enable these methods.

2.1 Defense Requires a Highly Pure Base Set

TABLE 1 summarizes some representative techniques that rely on access to a clean base set in each of the aforementioned defense categories, namely, Poison Detection, Trojan-Net Detection, Backdoor Removal, and Robust Training against label noise. These techniques either achieve the state-of-the-art performance (e.g., Frequency Detector [11], I-BAU [14], MW-Net [19]) or are widely-adopted baselines (e.g., MNTD [12] and Neural Cleanse (NC) [13]). In particular, MNTD is implemented as a base strategy in an ongoing competition for Trojan-Net Detection[2].

[1] `https://github.com/ruoxi-jia-group/Meta-Sift`
[2] `https://trojandetection.ai/`

While conventionally, these defense techniques only report their performance based on a completely clean base set, given the fast-advancing research on stealthy attacks, it is possible that some poisoned samples may go unnoticed and get selected into the base set by mistake. Hence, it is critical to evaluate how the performance of these defenses depends on the ratio of the poisoned samples in the base set.

We adopt widely used metrics to measure defense performance for each defense category. Specifically, for Poison Detection, we use **Poison Filtering Rate (PFR)**, which measures the ratio of poisoned samples that are correctly detected. For Trojan-Net Detection, we follow the original work of MNTD and use the **Area Under the ROC Curve (AUC)** as a metric, which measures the entire two-dimensional area underneath the ROC curve[3]. *The most naive baseline for poison detection and Trojan-Net detection is random deletion, which ends up with a PFR of 50% and an AUC of 50%.* The closer the performance of the defense in the Poison Detection and the Trojan-Net Detection category gets to 50%, the weaker the defense is. For backdoor removal, we use the **Attack Success Rate (ASR)**, which calculates the frequency with which non-target-class samples patched with the backdoor trigger are misclassified into the attacker-desired target class. For Robust Training, we use the **Test Accuracy (ACC)**, which measures the accuracy of the trained model on a clean test set. *The baselines for Backdoor Removal and Robust Training are simply the deployment of no defenses at all.* We report ASR or ACC that is obtained directly from training on the poisoned dataset. The closer the performance of defense in these two categories gets to these baselines, the weaker the defense is.

We compare the resulting defense performance against standard attacks (e.g., BadNets [8], Random Label-Flipping) between clean and corrupted base sets (Table 1). For Poisoned Detection with Frequency Detector, even one poisoned example sneaking into the base set is sufficient to nullify the defensive effect, leading to a performance worse than the random baseline. For MNTD, with 1% of poisoned examples mixed into the base set, the AUC drops by almost 40%. Comparing the two techniques for Backdoor Removal, we can find that I-BAU is more sensitive to corruption of the base set than NC. Both techniques patch a trigger to partial samples in the base set to fine-tune the poisoned model, aimed at forcing the model to "forget" the wrong association between the trigger and the target label. Compared to NC, the design of I-BAU selects fewer samples in the base set to be patched with a trigger. Hence, the positive "forgetting" effect introduced by these samples is more likely to be overwhelmed by the negative effect caused by poisoned examples sneaking into the base set. This explains the larger sensitivity of I-BAU to corruption of the base set. For both techniques, less than 3% of corruption in the base set is adequate to bring the ASR back above 60%. For Robust Training with MW-Net, 20 mislabeled samples in

[3] An ROC curve plots the true positive rate vs. the false positive rate at different classification thresholds

	Poison Detetcion	**Trojan-Net Detetcion**	**Backdoor Removal**		**Robust Training**
	Frequency Detector [11]	MNTD [12]	NC [13]	I-BAU [14]	MW-Net [19]
Task/ Settings	Detecting BadNets;	BadNets 5%; Target: 2;	BadNets 5%; Target: 38;		20% Random Label-Flipping;
Base Set	100-CIFAR-10	1000-MNIST	1000-GTSRB		100-CIFAR-10
Metric	PFR (↑ %)	AUC (↑ %)	ASR (↓ %)		ACC (↑ %)
Baseline	Random: 50	Random: 50	No Def: 97.43		No Def: 69.99
# poison	0/100	0/1000	0/1000	0/1000	0/100
Original	99.95	99.92	18.83	12.58	91.18
# poison	1/100	10/1000	30/1000	8/1000	20/100
After	3.11	62.78	62.67	81.82	81.84

Table 1: Defenses in the case of using a corrupted base set. For each category of defense, we use different metrics according to these original works. **Baseline** results are the settings with random guessing as defense or without defenses; **Original** results is the settings assuming an access to clean base set; **After** shows the results of using a contaminated base set.

the base set can reduce the accuracy by about 10%. Overall, we can see that base sets with high purity are crucial to enable the successful application of these popular defenses requiring access to base sets.

2.2 The Data Sifting Problem

The sensitivity of defense performance to the purity of the underlying base set motivates us to study the **Data Sifting Problem**: *How to sift out a clean subset from a given poisoned dataset?* We highlight some unique challenges and opportunities towards answering this question.

- (Challenge) **High precision**: The empirical study in Section 2.1 demonstrates that the defensive performance could drop significantly with a small portion of corruption in the base set. Hence, sifting out clean samples with high precision is crucial to ensure defense effectiveness.
- (Challenge) **Attack-Agnostic**: In practice, the defender usually does not know the underlying attack mechanisms that the attacker used to generate the poisoned samples. Hence, it is important to ensure high precision across different types of poisoning attacks.
- (Opportunity) **Mild requirement on the size of the sifted subset.** In contrast to the high requirement on the purity, the requirement of the size of the sifted subset is generally mild. The size of the base set required to enable an effective defense is usually much **smaller** compared to the size of the poisoned set. For instance, a clean base set of size less than 1% of the whole poisoned dataset suffice to enable effective defenses [14, 20] .

Note that some attempts have been made in the prior work to lift the requirement on base sets in data poisoning defenses [15, 21–23]. However, these works are focused on specific defense categories against specific attack settings. By contrast, when solving the data sifting problem one can obtain a highly pure base set that can be plugged into *any* defense technique that requires the base-set-access. Hence, we argue that **solving the data sifting problem provides a more flexible pathway to address the base-set-reliance issue in current data-poisoning defense literature.**

2.3 How Effective are Existing Methods

We first consider automated methods that can potentially solve the data sifting problem. Note that the data sifting problem is similar to the traditional outlier detection problem, wherein the goal is to sift out the abnormal instances in a contaminated dataset. Ideally, if one could filter out all the abnormal instances perfectly, then the complement set can be taken to solve the data sifting problem. There are two **key differences** between data sifting and outlier detection. *Firstly*, the data sifting problem is contextualized in data poisoning defense applications, where it is not necessary to sift out all the clean instances, but instead, a small subset of clean instances in the training set often suffices to support an effective defense. *Secondly*, in outlier detection, it is often more important to achieve high recall at a given selection budget (i.e., a high proportion of true outliers is marked as outliers by the detection algorithm), whereas in the data sifting problem, high precision is the key to realize a successful defense (i.e., a high proportion of the "marked-as-clean" points is truly clean).

At a technical level, the outlier detection algorithms are still applicable to approach the data sifting problem. In particular, existing outlier detection algorithms assign an "outlier score" to each data point, indicating their likelihood to be an outlier. To re-purpose these algorithms for data sifting, we select the points with the lowest "outlier scores" of each class to form the balanced base sets. We evaluate some representative outlier detection methods that do not rely on additional clean data to function and examine their potential to solve the identified problem. Specifically, we evaluate:

- Distance to the Class-Means (*DCM*): We compute the mean of each class at the input-space (pixel-level) as the class center and assign the outlier score to a point based on the distance to its center.
- Distance to Model-Inversion-based CM (*MI-DCM*): MI-DCM differs from DCM in the choice of the class center. Here, each class center is obtained by conducting the model inversion attack [24] on the model trained on the entire dataset. Model inversion is a type of privacy attack aimed at reconstructing the representative points for each class from the trained model.
- Spectral Filtering's Least Scores (*SF-Least*) [25]: Spectral Filtering is an advanced outlier detection method in robust statistics. Its key idea is that the outliers will result in larger eigenvalues than expected eigenvalues of sample covariance. This idea has been applied to the detection of backdoored samples. To start with, we extract features for each sample using the model trained over the contaminated dataset and the outlier score of a sample is calculated based on the dot product between its feature and the top eigenvector of the sample covariance matrix.
- Loss-based Poison Scanning (*Loss-Scan*) [26]: Recent work has identified the difference between the losses of benign and backdoored samples [26]. For most backdoor attacks, the losses of the poisoned sample would be lower

Poison attack \ Method	Random	DCM	MI-DCM*	SF-Least*	Loss-Scan*	Self-IF*
Targeted Label-Flipping [3]	83.3	92	93.3±0.5	**82.7±0.5**	**69.0±1.7**	**72.0±0.8**
Narcissus Backdoor [7]	90	**90**	**0.0±0.0**	94.00±0.0	**82.0±0.3**	**88.3±0.9**
Poison Frogs [5]	90	**90**	**72.7±2.9**	**87.0±0.0**	92.4±0.7	91.7±0.5
BadNets Backdoor [8]	66.7	**63**	**62.0±0.0**	76.0±0.8	67.0±0.5	70.3±0.5
Overhead (seconds)	NA	10	**5411+300**	**5411+15**	210	**5411+19832**

Table 2: Automated methods' precision for sifting out a size-1000 CIFAR-10 subset (each class 100 samples). Precision is calculated using # correct clean samples in the selected subset divided by the total # samples for the respective target class (100). The **red** highlights the methods that perform even worse than random selection. The **yellow** highlights the methods that are not time-efficient. The methods with * denote that the setting includes randomness (e.g., resulting from deep learning model training); thus, we report the corresponding means and the standard deviations.

than those of the benign sample during the early learning period (5 epochs). We adopt this notion and record the negative value of early training losses as the outlier score, i.e., the smaller the original loss, the greater the likelihood that the sample contains potentially hazardous information.

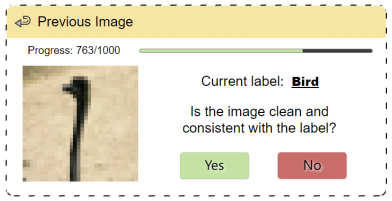
- Self-Influence-Function (*Self-IF*) [27]: Influence functions are designed to measure the impact of each training point on the loss of the trained model and have been widely used to enhance training [28] and repair mislabeled training data [29]. In particular, Self-Influence-Function is a variant that measures the influence of a point on the training loss. The outlier score of a point here is taken to be the negative of the result returned by Self-Influence-Function.

TABLE 2 shows the sifting results of existing methods on CIFAR-10. We include four popular attack strategies from the three categories of data poisoning in our evaluation, i.e., Targeted Label-Flipping, Narcissus backdoor, Poison Frogs, and BadNets backdoor. Settings are detailed in Appendix 6.1.

As shown in TABLE 2, none of the existing automated methods can sift out a clean subset from a poisoned dataset with high enough precision to support effective defense (recall the results in Table 1 even the purity of 97% can already significantly weaken the defenses under the BadNets attack). These methods employ simple criteria to calculate the outlier scores, which can be evaded by advanced attack techniques. Also, we observe that the sifting precision of each method varies largely over different attacks. Worse yet, many of these approaches produce results even inferior to random selection. These negative results motivate us to analyze the commonality of existing poisoning strategies in order to devise a sifting strategy effective under different attacks. Lastly, note that MI-DCM, SF-Least, and Self-IF require a well-trained model using the contaminated dataset and incur high computational costs. For instance, in our experiments, such training requires 5411 GPU seconds (on one RTX 2080 Ti).

2.4 How Effective is Human Inspection

Given that the automated methods fall short of sifting out a clean base set, we ask the question: can humans accurately



Is the image clean and consistent with the label?

Figure 2: The user interface designed for evaluating humans' ability to identify clean data.

differentiate between clean and poisoned instances?

Traditionally, human inspection is considered a final backstop of data poisoning. For instance, in the highly-cited survey of data poisoning [30], it is argued that it is the lack of human supervision on the dataset that opens the door for poisoning attacks and the subtext is that data poisoning attacks can be prevented with human inspection in place. However, as more and more stealthy poisoning attacks are developed, can human still serve a final backstop for data poisoning?

For completeness, we evaluate all three types of data poisoning: Label-only, Feature-only, and Label-Feature attacks. We consider the most representative and advanced attacks for each type. We extensively study 16 data poisoning attacks (1 Label-only attack, 5 Feature-only assaults, and 10 Label-Feature attacks). The details of the attacks and their settings are deferred to Appendix 6.2 due to the space limit. We create 16 poisoned datasets in total using CIFAR-10, each corresponding to a different attack. The size of all poisoned datasets are fixed to 1000 and the poison ratio is set to be 10%. To ensure comparability, we use the same random seed to select the poisoned samples (i.e., the poisoned samples in each dataset share the same indices). Finally, each poisoned dataset is sent to 3 human experts for inspection and each human expert only inspects one dataset to avoid possible bias introduced by repeated inspection. In total, we recruit 48 human experts from 3 different data labeling companies. Figure 2 illustrates a screenshot of the web portal that we develop for human experts to identify poisoned data. In particular, we provide the image itself and the label of that image to the human experts. They are asked to discern whether a given image is clean and consistent with the label.

We use false positive rate (FPR) or false negative rate (FNR) to measure the human's capability of distinguishing between clean and poisoned samples. FPR is defined as follows:

$$FPR = \frac{\text{\# of clean samples flagged as poisoned}}{\text{Total size of the clean samples}}. \tag{1}$$

FPR indicates the ratio of clean samples that are mistakenly flagged as poisoned. Meanwhile, FNR is the metric we are more interested in, which measures the ratio of poisoned samples that bypass human inspection and sneak into the sifted dataset. FNR is defined as:

$$FNR = \frac{\text{\# of poisoned samples marked as clean}}{\text{Total size of the poisoned samples}}. \tag{2}$$

We report the average and standard deviation of both metrics based on the results from multiple human experts. The results are summarized in Figure 3. Surprisingly, 9.13% of the clean samples are mistakenly recognized as the poisoned.

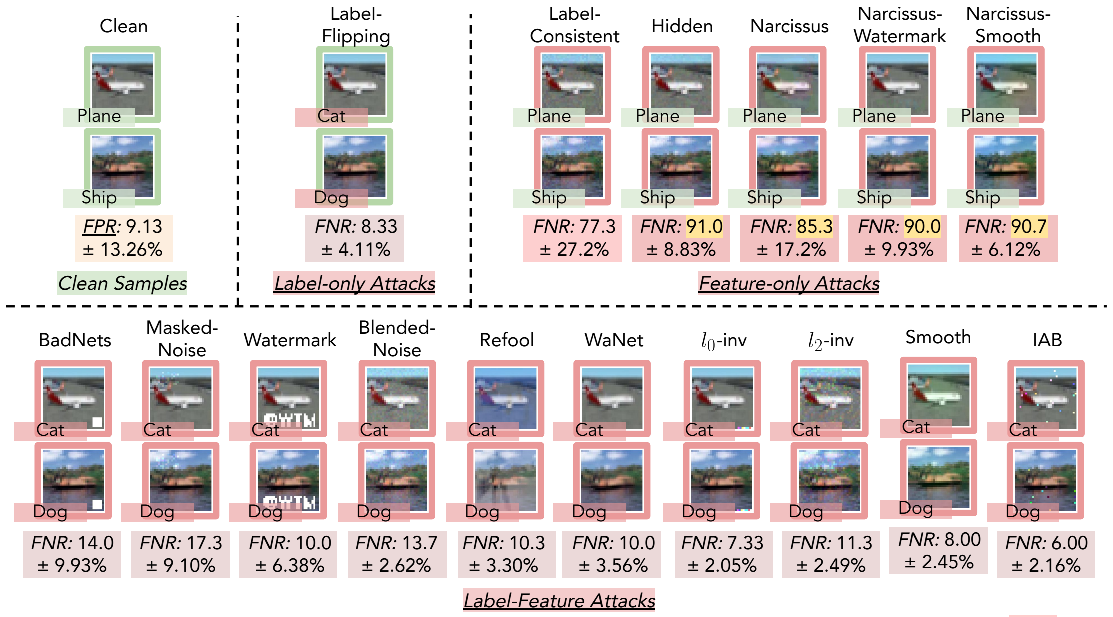


Figure 3: Human inspection results regarding data poisoning attacks. The labels and images marked in **red** depict potential manipulations under that attack category, and the **green** represents that the attribute remains intact. The evaluation uses 16 poisoned datasets, with 10% of data being manipulated following the existing attack designs. Among the three different categories of attacks, we report the error rate of misclassifying clean samples into poisoned ones (FPR) or poisoned ones into clean samples (FNR). Each poisoned dataset is inspected by at least three human experts from three different professional data labeling companies. We report the average results with standard deviations. The **highlighted** FNRs remark on the dangerous settings where human inspection failed in identifying 85% or more poisoned data points.

Humans are more capable of capturing manipulations on labels than on features. For Label-only attacks, human experts can filter out 91.67% of the poisoned samples. Such a result implies that despite the costs incurred, manual inspection can help avoid most label-flipped samples in the base set.

On average, the results on the evaluated 10 Label-Feature attacks are similar to those on the Label-only attacks, achieving an averaged FNR of 10.8%. However, Label-Feature attacks are much more powerful than Label-Flipping attacks. A small amount of the poisoned samples (e.g., a poison ratio of 0.01% suffices in existing literature [31, 32]) may successfully mislead the downstream model. At the same time, a small amount of such poisoned samples in the base set (less than 3% based on Table 1) may already significantly hinder defense performance. Hence, while humans achieve a relatively low FNR on Label-Feature attacks, the missed poisoned samples may still harm the downstream model or compromise the defense.

Even worse, when it comes to attacks that only manipulate data features, human expert's performance degrades sharply. The averaged FNR is above 86% as shown in Figure 3. Such a result is almost close to random guessing, which achieves a FNR of around 90% as the poison ratio is 10%. To conclude, humans have limited capability of identifying samples corrupted by feature manipulation. Finally, it is worth noting that clean-label backdoor attacks—an instance of Feature-only Attacks—are particularly dangerous. They can not only largely evade human inspection, but also backdoor a system with an extremely small poison ratio with state-of-the-art techniques (e.g., 0.05% for attacking Tiny ImageNet [7]).

It's noteworthy that our human study is conducted using only one dataset (CIFAR-10). Further exploration into the impact of image resolution and concept on humans' capability of poison detection is an interesting future work.

3 Meta-Sift

3.1 Reducing to a Splitting Problem

This paper presents a new but intuitive idea to solve the data sifting problem. Given a contaminated dataset, one can expect that an ML model trained on the clean portion of the dataset will perform poorly on the other corrupted portion and vice versa. Take standard backdoor attacks as an example. The poisoned instances are constructed by first applying a backdoor trigger to some clean inputs and then changing their labels to some target classes. A model trained on the clean data would return large prediction loss when it observes such poisoned instances because they are mislabeled, and their features only contain a trigger that does not appear in the clean data. This insight applies to general data poisoning attacks because they all involve feature and/or label manipulation, thereby resulting in a distributional shift from the clean instances. Thus, we solve the data sifting problem by finding a split of the given contaminated dataset such that the model trained on one split produces a large loss on the other.

We formulate the splitting problem as a bilevel optimization. Formally, given the contaminated dataset $D = D_1 \cup \ldots \cup D_K$, where D_k is the subset of D but only containing samples with class label $k \in \{1, \cdots, K\}$, our goal is to divide each class-wise subset, D_k, into two splits B_k and $D_k \setminus B_k$, such that

$$\mathcal{B}^* = \arg\max_{\mathcal{B}} \sum_{k=1}^{K} \sum_{z_i \in D_k \setminus B_k} L(\theta^*(\mathcal{B}), z_i), \quad (3)$$

$$\text{s.t. } \theta^*(\mathcal{B}) = \arg\min_{\theta} \sum_{k=1}^{K} \sum_{z_j \in B_k} L(\theta, z_j), \quad (4)$$

$$|B_1| = \ldots = |B_K|, \quad (5)$$

$$|B_1| + \ldots + |B_K| \geq (1-\varepsilon)|D|. \quad (6)$$

where $\mathcal{B} = B_1 \cup \ldots \cup B_K$ is the union of the class-wise training splits, θ represents the parameters of an ML model, and $\theta^*(\mathcal{B})$ denotes the parameters obtained from training on $\mathcal{B}$. L is the loss function, e.g., cross-entropy loss for classification tasks. The inner function (4) acquires a model that is trained over one split. The outer optimization (3) evaluates the model performance on the other split – $D \setminus \mathcal{B}$. The formulation seeks for the subset $\mathcal{B}$ that leads to the highest loss when evaluated on $D \setminus \mathcal{B}$. Additionally, (5) constrains the size of the split of each class-wise subset so that the union $\mathcal{B}$ is a perfectly label-balanced subset of D. (6) enforces the acquired $\mathcal{B}$ to have at least $(1-\varepsilon)|D|$ samples, where ε is an upper bound of the poison ratio. If we assume the number of poisoned samples in a given dataset to be smaller than that of clean samples (a standard assumption in robust statistics [33]), i.e., ε will be 0.5. Thus, (6) is also encourages $\mathcal{B}$ to be the clean split.

3.2 Relaxation of the Splitting Problem

Exactly solving Eqn. (3-6) requires training on every possible $\mathcal{B}$ and testing on their complement. This is clearly intractable. The computational challenge in part arises from the combinatorial nature of the outer optimization problem. To address this challenge, we propose to relax Eqn. (3-6) to a *continuous* splitting problem. We start by rewriting (3-6) as

$$\mathcal{V}^* = \arg\max_{\mathcal{V}} \sum_{k=1}^{K} \sum_{z_i \in D_k} (1 - v_{(k,i)}) L(\theta^*(\mathcal{V}), z_i), \quad (7)$$

$$\text{s.t. } \theta^*(\mathcal{V}) = \arg\min_{\theta} \sum_{k=1}^{K} \sum_{z_i \in D_k} v_{(k,i)} \cdot L(\theta, z_i), \quad (8)$$

$$\|V_1\|_1 = \ldots = \|V_K\|_1, \quad (9)$$

$$\|V_1\|_1 + \ldots + \|V_K\|_1 \geq (1-\varepsilon)|D|, \quad (10)$$

where instead of directly optimizing the split, we optimize the tuple of binary variables indicating each sample's presence, $\mathcal{V} = [V_1, \ldots, V_K]$,where $k \in \{1, \ldots K\}$. Each $V_k = [v_{(k,1)}, \ldots, v_{(k,|D_k|)}]$ encodes the splitting result of D_k, where $v_{(k,i)} = 1$ if $z_i \in \mathcal{B}$ (a class-k sample z_i assigned to the split for training) and $v_{(k,i)} = 0$ if $z_i \in D \setminus \mathcal{B}$ (z_i assigned to the split for testing). $\|V_k\|_1$ computes the l_1 norm of V_k, i.e., $\|V_k\|_1 = \sum_{i=1}^{|D_k|} v_{(k,i)} = |B_k|$. To solve Eqn. (7-10), we relax the binary variables $v_{(k,i)}$ as continuous ones, i.e., $v_{(k,i)} \in [0,1]$. Hereinafter, $v_{(k,i)}$ represents the likelihood of z_i being assigned to the split for training. This continuous relaxation enables us to leverage gradient-based methods to search for an approximate solution, i.e., each class-wise clean split B_k can be obtained by collecting the samples with larger $v_{(k,i)}$.

However, when the size of the dataset is large, the outer optimization needs to optimize over a continuous space of the same large dimension, which could be slow. Inspired by advances in meta learning [15,19], we propose to further learn a weight-assigning network to assign weight to each point instead of directly optimizing the weights. We found that the number of parameters of such a network required to produce useful weights is much smaller compared to the data size. For instance, in our experiments, we found a network with 100 nodes is sufficient for high-quality weight assignment, but on the other hand, the size of the datasets considered in this paper is orders of magnitude larger. Hence, through adapting a weight-assigning network, we effectively reduce the number of variables that are optimized in the outer optimization.

Specifically, let $\mathcal{S}(\cdot)$ denote the weight-assigning network and let ψ denote its parameters. Furthermore, $\forall k$ and $z_i \in D_k$, we let $v_{(k,i)} = \mathcal{S}(L(\theta, z_i); \psi)$, i.e., the weight-assigning network determines the weight for z_i based on its associated learning loss from θ. For simplicity, we use $L_i(\theta)$ as a shorthand for $L(\theta, z_i)$. Then, the continuous splitting problem that we will solve can be expressed as:

$$\psi^* = \arg\max_{\psi} \sum_{i=1}^{|D|} (1 - \mathcal{S}(L_i(\theta^*(\psi)); \psi)) L_i(\theta^*(\psi)), \quad (11)$$

$$\text{s.t. } \theta^*(\psi) = \arg\min_{\theta} \sum_{i=1}^{|D|} \mathcal{S}(L_i(\theta); \psi) L_i(\theta). \quad (12)$$

Since samples from each class are weighted by a shared weight-assigning network, in (11, 12), we no longer specify which class z_i is from. Moreover, note that the optimization problem above does not have explicit constraints to ensure that the training split is class-balanced and contains the majority of the samples. Instead, we will adopt two heuristics to enforce these constraints. **1)** We apply $\mathcal{S}(\cdot)$ to each sample to obtain their weight and then select the same amount of highest-weight samples within each class; **2)** We initialize the weights, $v_{(k,i)}$, to be all ones, i.e., assigning all the samples to the training split and terminate the optimization with limited rounds. The limited round of the optimization will leave most of the samples with large $v_{(k,i)}$ values, thus resulting in an implicit satisfaction of the constraint of selecting more samples for the training split. Each hidden node in $\mathcal{S}(\cdot)$ is equipped with ReLU activation function [34], making it possible to approximate non-linear functions. The output node produces a one-dimensional value indicating the weight assigned to the input to the weight-assigning network. The output node utilizes a Sigmoid activation function to ensure that the output is within the range $[0,1]$.

3.3 Overall Algorithm

<u>Overview.</u> We now describe the full algorithm to sift out a clean subset, which we call META-SIFT . Given a poisoned dataset and a selection budget, META-SIFT aims to select a subset that is most likely to be clean. The subset output by META-SIFT can then be used as a base set in different

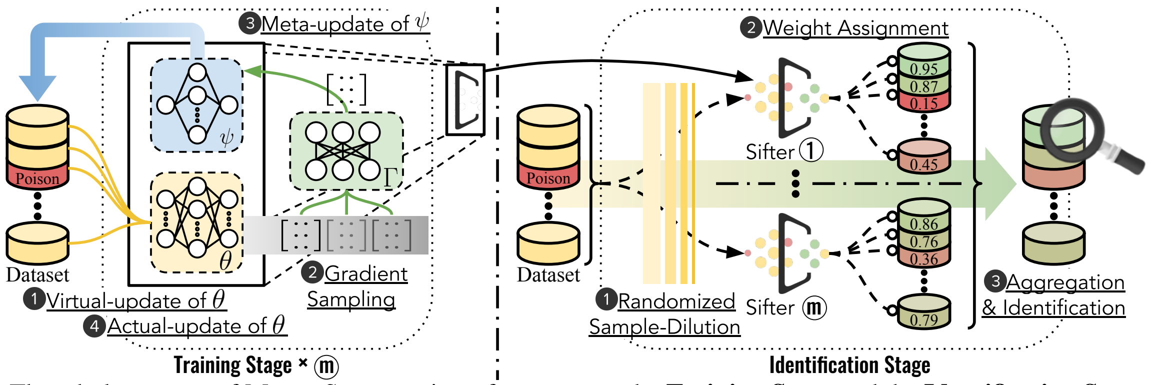


Figure 4: The whole process of META-SIFT consists of two stages: the **Training Stage** and the **Identification Stage**. Multiple (m) Sifters will be included during the **Identification Stage** to reduce the randomness resulting from SGD and randomized sample-dilution. As such, the **Training Stage** will be repeated m times with different random seeds to obtain m Sifters. In each Sifter, there are two different structures working as a pair: model θ and the weight-assigning network ψ. In one iteration of the **Training Stage**, there are four steps: Virtual-update of θ; Gradient Sampling using the meta-gradient-sampler Γ; Meta-update of ψ; then the Actual-update of θ. After only one iteration, **Training Stage** will terminate. The trained Sifters will be adopted in the **Identification Stage** to assign weights to the diluted data from the dataset. Finally, META-SIFT aggregates the results from multiple Sifters, and the clean samples will be sifted by inspecting the high-value end.

downstream defense algorithms to fulfill their corresponding defense goals. At a high level, META-SIFT consists of two stages: **Training** and **Identification**. In the Training stage, we adopt an online algorithm to solve (11, 12), and the algorithm will produce optimized parameters ψ^* for $\mathcal{S}(\cdot)$ as well as model parameters θ^* trained on the weighted samples. The definition of θ^* will be more clear after we explicate the algorithm. We will refer to the combination of $\mathcal{S}(\cdot)$ parameterized by ψ^* and the classification model parameterized by θ^* as a **Sifter**. To mitigate the randomness of batch selection in the online algorithm, we run the online algorithm independently for m times, which results in m Sifters. In the Identification stage, we feed the dataset into each Sifter and aggregate the weights produced by all Sifters to obtain the final weight for each sample in the dataset. The full process of META-SIFT is depicted in Figure 4.

Training Stage. We present how to train a single Sifter in META-SIFT . The training process starts by "warming up" θ—training θ on the entire dataset for one epoch. Essentially, we update θ while setting all weights to be one. This "warm-up" step helps promote $\mathcal{S}(\cdot)$ to assign large weight (close to 1) to samples, which in turn, allows most of the samples to be placed into the split for training, as mentioned above. Then, the Sifter (with θ and ψ) are updated via an online algorithm (illustrated in Figure 4). At each round of the online algorithm, we perform the following four steps: *Virtual-update of a warmed-up model* θ, *Gradient Sampling* using the meta-gradient-sampler Γ, *Meta-update of* ψ, and finally the *Actual-update of* θ.

❶ *Virtual-update of* θ. This step takes a virtual update of θ, the same as the traditional stochastic gradient descent (SGD). Formally, a mini-batch of data $\{z_i\}_{i=1}^n$ is sampled, where n is the batch size, and the following update is performed:

$$\theta' = \theta - \alpha \mathbf{g}_\theta, \tag{13}$$

where $\mathbf{g}_\theta = \frac{1}{n}\sum_{i=1}^n \mathcal{S}(L_i(\theta);\psi)\frac{\partial L_i(\theta)}{\partial \theta}$, representing the weighted gradient of the training loss with respect to θ and α is the learning rate. Suppose θ is a neural network consisting of $\mathcal{L}$ layers. Let the parameters in layer l be denoted by w_l. Then, we can rewrite $\mathbf{g}_\theta$ as

$$\mathbf{g}_\theta = \left(\frac{1}{n}\sum_{i=1}^n \mathcal{S}(L_i(\theta);\psi)\right) \times [g_1,\dots,g_{\mathcal{L}}], \tag{14}$$

where $g_l = \frac{\partial L_i(\theta)}{\partial w_l}$, representing a scaled gradient with respect to the parameters in layer l.

❷ *Gradient Sampling.* Inspired by [15], we parallelly train a list of u gradient-samplers to select partial gradients from the last u layers, $\{g_{\mathcal{L}-u},\dots,g_{\mathcal{L}}\}$ for subsequent update on ψ. The goal of gradient sampling is to accelerate and achieve more stable results on the subsequent update on ψ [15].

In particular, we associate a different gradient-sampler $\gamma_l(\cdot)$ for each layer $l \in \{\mathcal{L}-u,\dots,\mathcal{L}\}$. γ_l takes as input the gradient g_l and returns a value indicating whether g_l is selected. The reason why each layer has a different gradient-sampler is that their corresponding gradients are of different dimensions and $\gamma_l(\cdot)$'s input dimension needs to be tailored to the size of g_l. Following [15], we implement each gradient-sampler as a two-layer fully-connected neural network. The activation units in the first and second layer are PReLU [35] and "Gumbel-softmax" [36], respectively.

The l-th gradient sampler, γ_l, can be updated using the gradient of the outer optimization objective w.r.t. the gradient sampler's parameters. The dependency of the outer optimization objective on the sampler's parameters is via ψ. The gradient can be calculated automatically via auto-differentitaion in standard deep learning frameworks [37]. We then update each sampler according to the respected gradients before selecting layers. The gradient of a layer is selected if the output of the respect sampler exceeds 0.5. We denote the collection of the weights for layers that are selected by a gradient sampler as

$$w_C = \{w_l : \gamma_l(\frac{\partial L_i(\theta)}{\partial w_l}) > 0.5\}. \tag{15}$$

❸ *Meta-update of* ψ. In this step, we update the parameter ψ of the weight-assigning network by calculating the gradient of the outer optimization objective function with respect to ψ and performing the gradient descent: $\hat{\psi} = \psi - \beta \mathbf{g}_\psi$, where $\mathbf{g}_\psi$

$$\mathbf{g}_\psi = \frac{\partial}{\partial \psi}\left(\frac{1}{n}\sum_{i=1}^{n}(1-\mathcal{S}(L_i(\theta');\psi))L_i(\theta')\right) = \frac{1}{n}\sum_{i=1}^{n}\left(-\frac{\partial \mathcal{S}(L_i(\theta');\psi)}{\partial \psi}L_i(\theta') + (1-\mathcal{S}(L_i(\theta');\psi))\frac{\partial L_i(\theta')}{\partial \psi}\right). \quad (16)$$

$\frac{\partial L_i(\theta')}{\partial \psi}$ is often referred to as *indirect gradient*, and expands as

$$\frac{\partial L_i(\theta')}{\partial \psi} = \frac{\partial L_i(\theta')}{\partial \theta}\left(-\frac{\alpha}{n}\sum_{j=1}^{n}\frac{\partial \mathcal{S}(L_j(\theta);\psi)}{\partial \psi}\frac{\partial L_j(\theta)}{\partial \theta}\right). \quad (17)$$

The process of calculating the indirect gradient of ψ has been shown to be computationally intensive [38, 39]. However, in our case, by using the gradient samplers, we only consider the selected layers to calculate the indirect gradient, i.e.,

$$\frac{\partial L_i(\theta')}{\partial \psi} = \frac{\partial L_i(\theta')}{\partial w_C}\left(-\frac{\alpha}{n}\sum_{j=1}^{n}\frac{\partial \mathcal{S}(L_j(\theta);\psi)}{\partial \psi}\frac{\partial L_j(\theta)}{\partial w_C}\right). \quad (18)$$

Note that $\mathbf{g}_\psi$ does not need to be calculated manually; instead, it can also be computed via automated differentiation.

❹ *Actual-update of* θ. Finally, we update θ in place based on the weights assigned by the updated $\mathcal{S}(\cdot;\widehat{\psi})$:

$$\widehat{\theta} = \theta - \alpha\frac{1}{n}\sum_{i=1}^{n}\mathcal{S}(L_i(\theta);\widehat{\psi})\frac{\partial L_i(\theta)}{\partial \theta}. \quad (19)$$

In each round of our online algorithm, the four steps are performed on one batch of data. The algorithm terminates when it has gone through all the samples in D. The pseudocode of the Training Stage is in Algorithm 1, Appendix 6.3. We define the updated θ and ψ in the final round as θ^* and ψ^*, respectively. The pair of θ^* and ψ^* defines one Sifter's parameters. We will run the online Training algorithm multiple times to obtain the Sifters we will use in the Identification Stage.

Identification Stage. Now we aggregate the results from all the Sifters to identify the clean samples in three steps: *Randomized Sample-Dilution*, *Weight Assignment*, and *Weight Aggregation & identification*.

❶ *Randomized Sample-Dilution:* Instead of directly feeding a sample to the Sifter, we propose to first randomly perturb the sample. Existing works [40, 41] show that the prediction associated with a clean sample is robust to random perturbation. Hence, a sample that consistently receives high weights under different perturbations is more likely to be clean than those samples experiencing large variance under perturbations. The perturbation strategies considered in this paper include random cropping, random rotation, random horizontal flipping, and Gaussian blurring. For each sample, we apply all the perturbation strategies simultaneously, and each strategy is configured randomly (e.g., we randomly sample a rotation angle and rotate the sample correspondingly). Ablation study on these random perturbations is presented in Appendix 6.7.

❷ *Weight Assignment:* The random sample-dilution step produces m perturbed datasets and each dataset contains samples perturbed by a different random configuration of the perturbations strategies. Then, each Sifter (m in total) takes as input a perturbed dataset and produces the weights for all samples. In particular, the classification model (parameterized by θ^*) in the Sifter will be used to extract features from each sample and $\mathcal{S}(\cdot)$ (parameterized by ψ^*) will assign weight to a sample based on the extracted features.

❸ *Aggregation & identification:* We average the weights for each sample from multiple Sifters and use the top-weighted samples of each class k to form the sifted class-wise clean subset B_k. Finally, the clean split $\mathcal{B}$ is obtained by combining the class-wise results.

4 Evaluation

4.1 Experimental Setup

Metrics. One crucial factor in evaluating automated data sifting methods is the precision of the selection. We use the Corruption Ratio (CR) to measure the precision of each method's selection performance. Formally, let D_{sub} denote the selected subset from D using an automated method, and N_{poi} denote the total number of the poisoned samples in D_{sub}. Then, the CR of D_{sub} is defined as $CR = \frac{N_{poi}}{|D_{sub}|} \times 100\%$. While CR seems a natural performance metric, it is not well suited for comparing the precision of selection across different attack settings because they normally employ different poison ratios. For example, consider two attacks with poison ratios 5% and 20% and set the selection strategy to be a random one. Then, the CRs on the two attacks are different (roughly 5% and 20%, respectively) despite the same selection strategy. To facilitate the comparison of the sifting performance across different attacks, we propose the Normalized Corruption Ratio (NCR), defined as follows

$$NCR = \frac{CR}{CR_{rand}} \times 100\%, \quad (20)$$

where CR_{rand} is the corruption ratio of the random selection.

General Settings. We use two servers equipped with a total of 16 GTX 2080 Ti GPUs as the hardware platform. PyTorch [37] is adopted as the implementation framework.

We consider four popular benchmark datasets, namely, CIFAR-10 [42], GTSRB [43], PubFig [44], and the ImageNet [45]. The details of the datasets, the architecture of the model trained on each dataset, training algorithms, and performance of the models (trained on clean datasets as baselines) are provided in Table 3. For ImageNet, directly applying META-SIFT leads to high computational costs because a large-capacity classification model θ is needed to ensure the quality of extracted features. For large-scale datasets such as ImageNet, we adopt a practical and reliable alternative where we split the ImageNet dataset into multiple subsets and then apply META-SIFT on each one separately. In our evaluation, we will showcase the sifting performance on three non-overlapping random 10-class subsets of ImageNet.

In addition to the five attacks in Section 2, we add seven more representative attacks from different categories of data poisoning, the detailed attack settings are provided in TABLE

Dataset	CIFAR-10 [42]	GTSRB [43]	PubFig [44]	ImageNet [45]
# of Classes	10	43	83	1000
# of Samples	50,000	39,209	12,454	1,281,167
Input Shape	(3,32,32)	(3,32,32)	(3,224,224)	(3,224,224)
Target Class	5 (Dog)	38 (Keep Right)	60 (Miley Cyrus)	762; 578; 897
Model	PreActResNet-18	VGG-16	ResNet-18	ResNet-18
Epochs	200	50	60	200
Optimizer	SGD [46]	Adam [47]	RAdam [48]	SGD [46]
ACC	95.33	97.55	94.01	92.60; 86.00; 89.80
Tar-ACC	93.50	99.48	96.33	92.00; 92.00; 86.90

Table 3: Hyperparameters and settings to obtain clean baseline models. The target class of each dataset is fixed across all attacks with one target label. ACC and Tar-ACC represent the value of these metrics without any attacks.

4. We follow the poison ratio in the original attack papers. We mark the settings where the attacks are not successful (i.e., the attack settings that failed to result in an ASR above 50%) in red and remove them from comparison. We also exclude PubFig and ImageNet for the Smooth attack (colored in gray) as they were not considered in the original paper [11].

As size requirements for base sets in existing defenses are *mostly less than 1000 samples*, we will mainly examine each method's ability to sift out a **1000-size** base set on each poisoned dataset. However, we will show that META-SIFT can sift out more clean samples, but the performance may vary from attack to attack. Finally, we run the defense 3 times with different random seeds for all the results that include randomness (e.g., SGD for optimizations or random augmentations) and present the respective means and standard deviations.

META-SIFT Settings. META-SIFT is equipped with a classifier model θ, weight-assigning network ψ, and a list of gradient samplers Γ. We will use the same ψ and Γ on all datasets. But for θ, we will select the model architecture based on the dataset size. θ functions as a feature extractor in META-SIFT and a larger, more complex dataset naturally requires a larger model for feature extraction. Specifically, for CIFAR-10 and GTSRB, we adopt ResNet-18; for PubFig and ImageNet, we use ResNet-34 and ResNet-50, respectively.

4.2 Sifting Performance

Now we evaluate and compare META-SIFT with the five existing automated methods previously discussed in Section 2—DCM, MI-DCM, SF-Least [25], Loss-Scan [26], and Self-IF [27]. Table 4 compares NCR of the five baselines and META-SIFT across different datasets with richer attack settings than Table 2. An NCR value above 100% indicates that the selection performance is worse than the naive random baseline. By contrast, an NCR value below 100% implies better selection than random. The best possible NCR is 0%. We mark the results worse than random selection in red, the time-consuming selection results in yellow, and the best baseline among the four existing automated methods in blue. If our result is better than those of the existing methods and achieves the maximum precision, we mark them in green. Note that across all the baselines, no augmentations technique is included (i.e., sample-dilution). To ablate the effect of sample dilution, we present additional results in Table 18, Appendix 6.7, and show that sample-dilution is not the decisive factor that can help other techniques resolve the sifting problem.

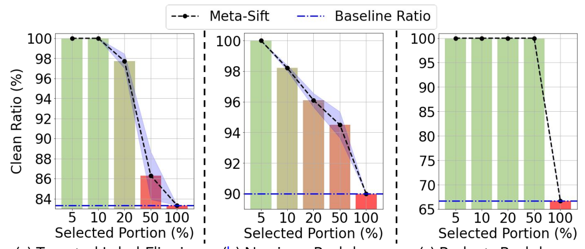


Figure 5: Sifting results of META-SIFT under representative settings of each attack category, GTSRB. Shades depict standard deviation among different runs.

Table 5 evaluates the sifting methods on CIFAR-10 aginst different attacks. CIFAR-10 is a balanced dataset containing 10 classes. Hence, we select 100 samples per class to form the base set of a total size 1000. In most settings, the five baselines do not yield a completely clean base set, and the performance varies largely from one attack to another (even within the same category). By contrast, META-SIFT produces a clean subset (NCR = 0%) in all the evaluated settings while enjoying a lower computational overhead than most of the baselines. Note that many attacks listed in Table 4 can achieve similar attack effects but with a smaller poison ratio. Primarily, many defense works have found attacks with lower poison ratios are harder to be identified [20, 50]. Thus, we present additional results of META-SIFT on CIFAR-10 with smaller poison ratios in Table 11, Appendix 6.5. Similar results can be found on GTSRB and PubFig, which are deferred to the Appendix 6.6 due to the page limitations.

Finally, the ImageNet results are presented in Table 6. As discussed, we break down the ImageNet dataset into smaller subsets and sift each subset sequentially to accelerate the sifting process. Table 6 demonstrates the sifting performance on three randomly selected 10-class subsets from ImageNet. The most important contribution of META-SIFT is that it's the only method that can reliably sift out a completely clean base set (NCR = 0%) across different attacks.

While we set the base set size to 1000 samples for the above experiments, we also study the behavior of META-SIFT for selecting a larger base set. Figure 5 depicts the percentage of clean samples under different sifting budgets on GTSRB. META-SIFT can select around 4000 (10%) clean samples under the Targeted Label-Flipping case. Notably, META-SIFT can achieve almost a perfect split when the poisoned instances are generated by the BadNets backdoor attack. On GTSRB, a general observation is that META-SIFT achieves a better split on attacks that manipulate labels. This is because the features for every class in GTSRB samples have relatively low variance and are distinct from other classes. Since incorporating data corrupted by label manipulations into the split for testing largely increases the outer loss in Eqn. (11,12), these data will be assigned a small weight and thus get unselected. Meanwhile, META-SIFT 's effectiveness is worse on attacks that only involve feature manipulation such as the Narcissus backdoor attack, but it can still maintain a high

		Label-only		Feature-only			Label-Feature						
		Targeted Label-Flipping [3]	Random Label-Flipping [4]	Clean-Label [6]	Narcissus Backdoor [7]	Poison Frogs [5]	BadNets One-Tar [8]	Smooth One-Tar [11]	IAB One-Tar [49]	Blended One-Tar [9]	BadNets All-to-all [8]	Smooth All-to-all [11]	Blended All-to-all [9]
CIFAR-10	**Attack Settings**	$[3 \rightarrow 5]$; Tar: 16.67%	[*all*]; All: 20%	[5]; Tar: 10%	[5]; Tar: 10%	[5]; Tar: 10%	[5]; Tar: 33%	[5]; Tar: 33%	[5]; Tar: 33%	[5]; Tar: 33%	[*all*]; All: 20%	[*all*]; All: 20%	[*all*]; All: 20%
	Results (%)	ACC: 91.77 ASR: 27.56 Tar-ACC: 83.78	ACC: 71.8	ACC: 91.52 ASR: 99.98	ACC: 93.26 ASR: 100	ACC: 92.88 ASR: 100	ACC: 84.03 ASR: 95.78	ACC: 85.63 ASR: 96.17	ACC: 92.21 ASR: 91.20	ACC: 89.43 ASR: 91.26	ACC: 85.57 ASR: 84.15	ACC: 84.05 ASR: 78.67	ACC: 84.53 ASR: 88.74
GTSRB	**Attack Settings**	$[2 \rightarrow 38]$; Tar: 16.67%	[*all*]; All: 20%	[38]; Tar: 10%	[38]; Tar: 10%	[38]; Tar: 10%	[38]; Tar: 33%	[38]; Tar: 33%	[38]; Tar: 33%	[38]; Tar: 33%	[*all*]; All: 20%	[*all*]; All: 20%	[*all*]; All: 20%
	Results (%)	ACC: 97.43 ASR: 91.72 Tar-ACC: 98.84	ACC: 95.28	ACC: 98.12 ASR: 6.41	ACC: 97.83 ASR: 100	ACC: 97.81 ASR: 100	ACC: 97.10 ASR: 97.43	ACC: 98.08 ASR: 98.92	ACC: 97.88 ASR: 95.72	ACC: 97.89 ASR: 98.11	ACC: 96.87 ASR: 95.49	ACC: 96.64 ASR: 95.75	ACC: 96.825 ASR: 95.84
PubFig	**Attack Settings**	$[52 \rightarrow 60]$; Tar: 16.67%	[*all*]; All: 20%	[60]; Tar: 10%	[60]; Tar: 10%	[60]; Tar: 10%	[60]; Tar: 33%		[60]; Tar: 33%	[60]; Tar: 33%	[*all*]; All: 20%		[*all*]; All: 20%
	Results (%)	ACC: 91.33 ASR: 89.00 Tar-ACC: 93.75	ACC: 77.89	ACC: 93.42 ASR: 33.18	ACC: 93.50 ASR: 100	ACC: 93.28 ASR: 98.00	ACC: 91.19 ASR: 89.88		ACC: 92.73 ASR: 84.53	ACC: 93.57 ASR: 93.64	ACC: 76.45 ASR: 66.69		ACC: 91.84 ASR: 90.03
ImageNet-1	**Attack Settings**	$[385 \rightarrow 762]$; Tar: 16.67%	[*all*]; All: 20%	[762]; Tar: 10%	[762]; Tar: 10%	[762]; Tar: 10%	[762]; Tar: 33%		[762]; Tar: 33%	[762]; Tar: 33%	[*all*]; All: 20%		[*all*]; All: 20%
	Results (%)	ACC: 89.80 ASR: 88.80 Tar-ACC: 87.25	ACC: 77.40	ACC: 91.25 ASR: 5.33	ACC: 92.60 ASR: 100	ACC: 91.80 ASR: 100	ACC: 90.10 ASR: 98.20		ACC: 89.90 ASR: 95.20	ACC: 89.20 ASR: 92.00	ACC: 78.00 ASR: 76.60		ACC: 78.40 ASR: 72.40
ImageNet-2	**Attack Settings**	$[48 \rightarrow 578]$; Tar: 16.67%	[*all*]; All: 20%	[578]; Tar: 10%	[578]; Tar: 10%	[578]; Tar: 10%	[578]; Tar: 33%		[578]; Tar: 33%	[578]; Tar: 33%	[*all*]; All: 20%		[*all*]; All: 20%
	Results (%)	ACC: 84.00 ASR: 82.20 Tar-ACC: 88.00	ACC: 75.60	ACC: 86.00 ASR: 7.55	ACC: 85.60 ASR: 100	ACC: 86.00 ASR: 100	ACC: 86.80 ASR: 99.20		ACC: 85.80 ASR: 97.40	ACC: 85.80 ASR: 92.60	ACC: 74.90 ASR: 71.30		ACC: 72.40 ASR: 68.40
ImageNet-3	**Attack Settings**	$[830 \rightarrow 897]$; Tar: 16.67%	[*all*]; All: 20%	[897]; Tar: 10%	[897]; Tar: 10%	[897]; Tar: 10%	[897]; Tar: 33%		[578]; Tar: 33%	[897]; Tar: 33%	[*all*]; All: 20%		[*all*]; All: 20%
	Results (%)	ACC: 86.00 ASR: 96.00 Tar-ACC: 84.00	ACC: 75.20	ACC: 89.40 ASR: 6.22	ACC: 89.60 ASR: 99.83	ACC: 88.40 ASR: 100	ACC: 89.60 ASR: 95.20		ACC: 89.00 ASR: 91.56	ACC: 88.80 ASR: 99.20	ACC: 75.80 ASR: 72.30		ACC: 78.80 ASR: 76.00

Table 4: Effectiveness of different attacks on given datasets. We detail the attack settings by listing out the target labels (e.g., $[3 \rightarrow 5]$ indicates that samples from class 3 are manipulated to the target-class 5, and $[all]$ indicates that samples are manipulated to all classes). Here, 'Tar' and 'All' denote the poison ratio in that particular setting, for the target class and the whole dataset respectively. Finally, attack performance is studied using ACC, ASR, or Tar-ACC as applicable. Note that the ASR of the Poison Frogs denotes the attack confidence. For ImageNet, we report results over 3 separate subsets.

	Label-only		Feature-only			Label-Feature							
	Targeted Label-Flipping [3]	Random Label-Flipping [4]	Clean-Label [6]	Narcissus Backdoor [7]	Poison Frog [5]	BadNets One-Tar [8]	Smooth One-Tar [11]	IAB One-Tar [49]	Blended One-Tar [9]	BadNets All-to-all [8]	Smooth All-to-all [11]	Blended All-to-all [9]	**Overhead (s)**
DCM	48.0	96.5	20.0	**100.0**	**100.0**	**111.0**	**9.00**	**105.1**	**18.0**	57.0	**3.00**	32.0	10
MI-DCM*	**40.0±3.45**	**134.8±31.1**	83.3±3.45	**1000±0**	**273.3±35.1**	**114.0±0**	**135.0±0**	**120.0±0**	**123.4±0**	**100.0±13.2**	5.68±0.58	**10.8±6.78**	**5411+300**
SF-Least*	**104.0±3.45**	**20.0±5.50**	**166.7±3.45**	**60.0±0**	**130.0±0**	**72.0±3.00**	68.0±1.73	**48.0±10.2**	65.2±4.52	**51.8±6.79**	65.6±3.28	58.3±7.89	**5411+15**
Loss-Scan*	**185.9±67.2**	**451.0±53.2**	80.0±0	**180.0±10.0**	**80.0±0**	99.0±16.0	**135.0±19.9**	84.0±22.4	**161.0±123.9**	69.5±15.7	68.5±7.80	**733.1±307.5**	210
Self-IF*	**168.0±6.00**	**114.7±1.61**	0±0	**116.7±11.5**	83.3±3.45	89.0±1.73	29.0±6.24	**108.1±23.1**	36.3±6.88	**119.5±3.46**	**142.2±38.2**	**137.6±31.6**	**5411+19832**
META-SIFT	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	5×100

Table 5: NCR results under the CIFAR-10 settings.

		Label-only		Feature-only		Label-Feature					
		Targeted Label-Flipping [3]	Random Label-Flipping [4]	Narcissus Backdoor [7]	Poison Frog [5]	BadNets One-Tar [8]	IAB One-Tar [49]	Blended One-Tar [9]	BadNets All-to-all [8]	Blended All-to-all [9]	**Overhead (s)**
Subset-1	DCM	82.5	**132**	76.9	**115.4**	**127.9**	72.0	**129.8**	**118.5**	**118.5**	10
	MI-DCM*	**59.8±2.80**	**94.0±5.20**	**56.4±8.88**	**115.4±20.3**	**104.1±4.0**	**108.0±1.9**	**122.8±4.7**	**97.9±2.9**	**127.5±0**	**7200+300**
	SF-Least*	97.1±4.11	**123.4±3.80**	**113.3±5.20**	**78.1±4.10**	**132.3±6.83**	**54.0±7.51**	**108.7±15.4**	**112.6±13.6**	**109.2±12.5**	**7200+15**
	Loss-Scan*	**185.9**	**366.1±65.4**	80.0±0.0	80.0±0.0	**51.0±16.3**	63.1±18.4	**35.9±8.36**	**271.3±18.9**	**238.8±68.1**	240
	META-SIFT	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	5×120
Subset-2	DCM	**4.95**	**156.0**	**61.5**	**115.4**	85.1	78.0	85.1	**137.0**	**132.0**	10
	MI-DCM*	**118.8±0**	**102.8±4.52**	91.5±4.17	**133.5±6.78**	**39.8±10.6**	**111.0±54.3**	68.3±3.65	**71.5±4.67**	**102.1±0.89**	**7200+300**
	SF-Least*	54.2±1.45	**83.2±0.92**	76.9±0	**0.0±0**	71.3±5.31	66.1±3.58	49.4±10.6	**124.1±3.82**	**128.3±5.67**	**7200+15**
	Loss-Scan*	**221.9±24.1**	**290.5±32.1**	80.0±16.9	60.0±0.0	45.0±6.34	**63.0±12.33**	**39.8±5.44**	**316.0±54.6**	**288.7±32.1**	240
	META-SIFT	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	5×120
Subset-3	DCM	**148.5**	**106.5**	46.1	76.9	91.1	84.1	91.1	**107.0**	**107.0**	10
	MI-DCM*	99.0±0	99.0±0	**31.2±3.43**	98.72±2.34	93.8±5.87	93.1±4.66	62.3±6.78	93.3±10.9	**113.5±4.62**	**7200+300**
	SF-Least*	92.1±10.9	**82.0±0**	63.5±5.65	**61.5±3.23**	61.2±4.67	**69.0±3.69**	79.2±8.75	90.8±4.53	**68.3±12.0**	**7200+15**
	Loss-Scan*	**85.9±12.6**	**233.5±23.1**	90.0±8.20	70.0±0.23	**42.0±4.32**	84.0±15.6	**35.7±0.60**	**89.0±6.89**	**105.5±19.2**	240
	META-SIFT	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	5×120

Table 6: NCR results under the ImageNet settings. We dropped Self-IF due to an exploding computational time.

precision up to the selection budget of about 2000 samples (accounting for 5% of the total size).

We want to highlight that *there is a trade-off between* META-SIFT*'s precision and the base set size to be selected.* However, existing defenses usually require only a small base set (i.e., often less than 1% of the total size of the poisoned set). The fact that META-SIFT can reliably sift out a pure clean set of more than 5% of the total size across different attacks makes it useful for defense applications (see detailed evaluation in Section 4.3). By contrast, the precision of human inspection does not change with the selection budget, because each sample is examined separately. Moreover, the existing automated methods constantly produce a very low precision regardless of the selection budget.

4.3 Downstream Defense Evaluation

Now, we evaluate the performance of defenses with the selected base set plugged in. Note that the data used to build the Sifters is the poisoned training set. Downstream defenses use the subset selected by the Sifters as the base set to achieve their specific defense goals. The performance of each defense is assessed on separate held-out clean and poisoned datasets. We consider the defense categories introduced in Section 2. We present the results of each defense when 1) the base set is

	BadNets [8]	Trojan-WM [51]	Trojan-SQ [51]	l_2 inv [52]	l_0 inv [52]
Clean	89.27	100.0	99.80	100	99.99
Random	**44.88**	**25.83**	**30.55**	**53.42**	**0.23**
META-SIFT	88.54	99.99	99.78	100.0	95.38

Table 7: PFRs (↑) of the Detector under CIFAR-10 attacks.

completely clean (**Clean**); 2) the base set is randomly selected from the contaminated training set (**Random**); and 3) the base set is selected by **META-SIFT** .

Poison Detection: Similar to the evaluation in Section 2, we consider the state-of-the-art frequency-based poison detection method [11]. Following the original work, we consider five attack settings with a poison ratio of 5%. The original setting of the frequency detector requires a base set of size 100. Table 7 compares the defense performance between the Clean, Random, and META-SIFT -selected base set. META-SIFT enables the frequency detector to obtain a defense result similar to the result using the original clean set.

	BadNets [8]	Blended [9]
Clean	99.20	99.85
Random	**47.34**	**49.83**
META-SIFT	99.50	99.92

Table 8: AUCs (↑) of MNTD on the MNIST.

Trojan-Net Detection: Now we evaluate how META-SIFT may help MNTD [12]. In the original work, it require a base set of size 1000 over the MNIST dataset in the jumbo training procedure to train 256 clean and 256 randomly poisoned models. Like the original work, we incorporate two sets of backdoor attacks with a poison ratio of 5%, namely, BadNets and Blended, and report the results of MNTD with different base sets. The results are listed in Table 8, showing that the performance with META-SIFT -generated base set is even higher than that with a clean base set. This implies that META-SIFT can help MNTD to work in a situation without access to a clean base set.

	BadNets [8]	Trojan-WM [51]	Trojan-SQ [51]	l_2 inv [52]	l_0 inv [52]
No Defense	97.43	99.37	98.90	98.36	98.11
Clean	18.83	19.21	18.34	16.78	16.33
Random	**62.67**	**71.34**	**68.78**	**59.06**	**61.32**
META-SIFT	16.42	17.56	16.88	16.23	14.98

Table 9: ASRs (↓) of NC purified models on the GTSRB.

	BadNets [8]	Trojan-WM [51]	Trojan-SQ [51]	l_2 inv [52]	l_0 inv [52]
No Defense	97.43	99.37	98.90	98.36	98.11
Clean	12.58	7.28	5.30	18.82	12.82
Random	**96.55**	**98.04**	**93.78**	**93.29**	**98.78**
META-SIFT	10.65	4.50	5.62	6.75	5.77

Table 10: ASRs (↓) of I-BAU purified models on the GTSRB.

Backdoor Removal: Table 9 and 10 show the result of NC and I-BAU using different 1000-size base sets on GTSRB. We use the same attack settings as the original works. Interestingly, by using the base set selected by META-SIFT , both NC and I-BAU can achieve a slightly higher defense efficacy than using a randomly selected clean base set. We conjecture that META-SIFT is optimized to select more consistent and robust information (as those selected samples have withstood randomized sample-dilution and still end up at the high-value end). Thus META-SIFT -selected data are naturally more robust to noises and harder to be misclassified when patched with weak noise. This in turn helps both methods synthesize more accurate triggers that can be used for backdoor removal. Visual examples of the selection results over different datasets are shown in Appendix, Figure 9; however, it is hard to conclude the sample consistency directly from visual observations. We further compare the synthesized triggers of using NC equipped with the two different base sets (randomly selected clean and the META-SIFT selected). We evaluate the averaging loss of clean samples patching with the synthesized triggers towards the target label. A much smaller loss indicates a more accurate trigger synthesis. Averaging from five different executions, we find the triggers synthesized using META-SIFT base sets achieved an averaging loss of 0.2774. The loss of trigger synthesized using randomly selected clean base sets is only 0.3321, which confirms our conjecture.

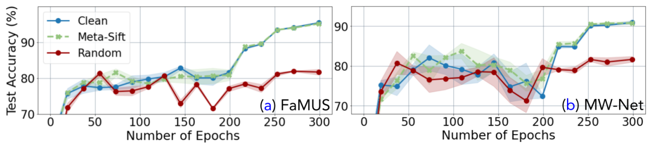


Figure 6: ACCs (↑) vs. the number of epochs of FaMUS and MW-Net on learning label-noisy CIFAR-10. Shades depict standard deviation among different runs.

Robust Training: Finally, we include the result of using META-SIFT to help existing robust training methods work in situations without additional clean samples. The representative works of this line of defense include MW-Net [19] and FaMUS [15]. As shown in Figure 6, both methods utilizing META-SIFT -generated base sets can achieve a similar level of effectiveness to using an additional clean validation set.

Note that, in the original paper of FaMUS, they also introduced a training strategy that does not need access to a clean base set. However, it requires training two paralleled neural networks and working in a manner that assigns pseudo labels for each other throughout the training procedure, which we find introduced 7× more GPU computational overhead than the setting with a clean base set access. By using META-SIFT , we get away with the additional computational overhead while still maintaining good performance in situations without access to the additional clean base set.

4.4 Adaptive Attack Analysis

To evaluate the robustness of META-SIFT against adaptive attacks, we consider a white-box scenario with the attacker having full knowledge of META-SIFT and the access to the entire training data, D. The evaluation settings will follow the settings on the CIFAR-10 with PreActResNet-18 listed in Table 3. With D, the defender will follow the workflow of META-SIFT to train m Sifters (parameterized by m pairs of (θ^*, ψ^*)) and use them to acquire the base set of size 1000. An adaptive attacker seeks to update D in a way that, upon providing the updated dataset, denoted as $\tilde{D}$, to the defender, the base set selected by META-SIFT will contain poisons.

Poisoning the Majority: An intuitive but less practical adaptive attack is to poison the majority of D, thereby violating the assumption stated in Section 3.1 (the number of poisons in each class is less than 50%). As this attack has limited practicality, we defer its analysis to Appendix 6.5.

Adversarial Noise using a Clean Model: Given the access to the whole dataset, D, the attacker knows which portion of D is clean and which will be poisoned when presenting to the defender. Another intuitive but more practical idea is to perturb poisoned samples generated by existing attacks in ways that cause minimal loss on the classifier $f(\cdot|\theta_{\mathrm{cln}})$ trained using the clean portion of data:

$$\delta^* = \arg\min_{\delta} L(f(x+\delta|\theta_{\mathrm{cln}}), y), \tag{21}$$

where L denotes the loss function. x and y are the feature and the label of a poisoned sample $z_{poi} = (x, y)$. We optimize the objective using SGD till convergence and find that the resulting perturbation can successfully minimize the loss without introducing large visual artifacts. To obtain $\tilde{D}$, we compute δ^* for each z_{poi}, and update $z_{poi} = (x, y)$ with $\tilde{z}_{poi} = (x+\delta^*, y)$. We evaluate META-SIFT on three poisoned datasets that updated with this adaptive design, each of which was originally poisoned using a representative attack from one of three categories: Targeted Label Flipping, Narcissus, and BadNets. The details of the implementation and $\tilde{D}$'s attacking performances are provided in Appendix 6.4 and 6.5. Our results show that the adaptively perturbed poisons can indeed achieve higher weights, compared to the non-adaptive setting; yet, the 1000 samples with top weights are still pure clean, and thus META-SIFT achieves a 0% NCR at a selection budget of 1000 (i.e., a size that empowers downstream defenses, Section 4.3).

Adversarial Noise using Sifters: The third way to adaptively attack META-SIFT is to disguise each poisoned sample by adding adversarial noise that allows these samples to obtain a high average score when passing through m Sifters (parameterized by m pairs of (θ^*, ψ^*) obtained from D):

$$\delta^* = \arg\max_{\delta} \frac{1}{m} \sum_{s=1}^{m} \mathcal{S}\left(L(f(x+\delta|\theta_s^*), y); \psi_s^* \right), \tag{22}$$

where θ_s^* and ψ_s^* denotes the parameters of the Sifter indexed by s. We employ stochastic gradient ascent to solve the above optimization till convergence and again found that the optimized perturbation does not introduce much visual change. To obtain $\tilde{D}$, we compute δ^* for each z_{poi}, and subsequently update z_{poi} with $\tilde{z}_{poi} = (x+\delta^*, y)$. We evaluate META-SIFT with the same three attack settings as above, and the details of the implementation and $\tilde{D}$'s attacking performances are provided in Appendix 6.4 and 6.5. From our results, we find the sifting performance over $\tilde{D}$ at the selection budget of 1000 remains 0% NCR, although each $\tilde{z}_{poi}$ can obtain a high score using the old Sifters parameterized by (θ^*, ψ^*) obtained from D.

Remark: The above analysis suggests the difficulty of adaptively attacking our method. We attribute the robustness to adaptive attacks to three factors: (**1**) META-SIFT is based on a robust insight that attacks, regardless of their mechanisms and whether they are adaptive or not, introduce some perturbations that eventually lead to a distributional shift from clean samples. A subset of clean samples, as long as poisons remain the minority, can be found with a combinatorial splitting process. (**2**) META-SIFT is, by design, a bilevel optimization; thus, it is difficult to synthesize "optimal" attacks that require optimizing through a multi-level optimization. (**3**) META-SIFT is designed to select a relatively small set, as existing defenses only require a small base set. Thus, a successful adaptive attack would need to alter the poisons to achieve higher weights than the majority of clean samples to be selected. *How to design such an attack remains an open question.*

5 Conclusion and Outlook

This work presents the first focused study of the data sifting problem, aimed at sifting out a clean subset from a dataset potentially contaminated by unknown poisoning attacks. This problem is of critical importance to successfully implement and apply existing defenses against poisoning attacks, as most of them require a clean base set to initiate the defense mechanisms. We first show that the performance of the popular and many state-of-the-art defenses is sensitive to the precision of clean data selection. Our study of the existing automated methods and human inspection shows that both cannot reach the precision required to achieve effective defenses. We further propose META-SIFT as a first solution to the data sifting problem. META-SIFT is based on an intuitive and reliable insight that regardless of how poisoned samples are generated, when one trains on only the clean data, the prediction loss of the trained model on these poisoned examples is large. Our evaluation shows that META-SIFT is faster than existing automated methods by significant orders of magnitude while achieving much higher sifting precision. In particular, training META-SIFT only requires going through the dataset for two epochs, and using META-SIFT to sift data can be done in minutes. Plugging the sifted samples into existing defenses achieves comparable or even better performance than using randomly selected clean data.

Limitations & Outlook. Despite the efficacy and time-efficiency, META-SIFT can cause high memory overhead due to the use of multiple perturbed datasets and Sifters. To support memory-constrained applications, further reduction of overhead is necessary. Additionally, there is a trade-off between META-SIFT 's precision and the size of the set to be sifted, as shown in Figure 5, META-SIFT can confidently sift out approximately 5% of the size of the dataset to be clean across different settings, datasets, and attacks, which is sufficient for most existing defenses (as discussed in Section 4.3). However, the size is insufficient for any model trained directly on the sifted samples to perform well. Therefore, expanding the scale of META-SIFT 's performance and enabling direct training is future work that needs to be addressed.

Additionally, there are a number of interesting venues for future exploration. For example, is it possible to design successful and practical adaptive attacks against META-SIFT that enabling the infiltration of poisoned samples past our sifting? Also, it might be interesting to explore other methods for resolving the proposed sifting or splitting problem and generalize the idea to more settings beyond image classification.

Acknowledgement

This work is partially supported by Sony AI. RJ and the ReDS lab appreciate the support of the Amazon - Virginia Tech Initiative for Efficient and Robust Machine Learning and the Cisco Award. YZ is supported by the Amazon Fellowship.

References

[1] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *ICML*, 2021, pp. 8748–8763.

[2] R. S. S. Kumar, M. Nyström, J. Lambert, A. Marshall, M. Goertzel, A. Comissoneru, M. Swann, and S. Xia, "Adversarial machine learning-industry perspectives," in *2020 IEEE SPW*, 2020, pp. 69–75.

[3] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *ESORICS*, 2020, pp. 480–501.

[4] M. Ren, W. Zeng, B. Yang, and R. Urtasun, "Learning to reweight examples for robust deep learning," in *ICML*, 2018, pp. 4334–4343.

[5] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," in *NeurIPS*, vol. 31, 2018.

[6] A. Turner, D. Tsipras, and A. Madry, "Label-consistent backdoor attacks," *arXiv:1912.02771*, 2019.

[7] Y. Zeng, M. Pan, H. A. Just, L. Lyu, M. Qiu, and R. Jia, "Narcissus: A practical clean-label backdoor attack with limited information," *arXiv:2204.05255*, 2022.

[8] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "Badnets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019.

[9] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv:1712.05526*, 2017.

[10] T. A. Nguyen and A. T. Tran, "Wanet-imperceptible warping-based backdoor attack," in *ICLR*, 2020.

[11] Y. Zeng, W. Park, Z. M. Mao, and R. Jia, "Rethinking the backdoor attacks' triggers: A frequency perspective," in *ICCV*, 2021.

[12] X. Xu, Q. Wang, H. Li, N. Borisov, C. A. Gunter, and B. Li, "Detecting ai trojans using meta neural analysis," in *2021 IEEE S&P*, 2021, pp. 103–120.

[13] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *2019 IEEE S&P*, 2019, pp. 707–723.

[14] Y. Zeng, S. Chen, W. Park, Z. Mao, M. Jin, and R. Jia, "Adversarial unlearning of backdoors via implicit hypergradient," in *ICLR*, 2022.

[15] Y. Xu, L. Zhu, L. Jiang, and Y. Yang, "Faster meta update strategy for noise-robust deep learning," in *CVPR*, 2021, pp. 144–153.

[16] W. Guo, L. Wang, X. Xing, M. Du, and D. Song, "Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems," *arXiv:1908.01763*, 2019.

[17] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *RAID*, 2018, pp. 273–294.

[18] Z. Xiang, D. J. Miller, H. Wang, and G. Kesidis, "Revealing perceptible backdoors in dnns, without the training set, via the maximum achievable misclassification fraction statistic," in *MLSP*, 2020, pp. 1–6.

[19] J. Shu, Q. Xie, L. Yi, Q. Zhao, S. Zhou, Z. Xu, and D. Meng, "Meta-weight-net: Learning an explicit mapping for sample weighting," in *NeurIPS*, vol. 32, 2019.

[20] Z. Xiang, D. J. Miller, and G. Kesidis, "Post-training detection of backdoor attacks for two-class and multi-attack scenarios," *arXiv:2201.08474*, 2022.

[21] R. Wang, G. Zhang, S. Liu, P.-Y. Chen, J. Xiong, and M. Wang, "Practical detection of trojan neural networks: Data-limited and data-free cases," in *ECCV*, 2020, pp. 222–238.

[22] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, "Strip: A defence against trojan attacks on deep neural networks," in *ACSAC '19*, 2019, p. 113–125.

[23] H. Chen, C. Fu, J. Zhao, and F. Koushanfar, "Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks." in *IJCAI*, vol. 2, no. 5, 2019, p. 8.

[24] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *SIGSAC*, 2015, pp. 1322–1333.

[25] B. Tran, J. Li, and A. Madry, "Spectral signatures in backdoor attacks," in *NeuIPS*, 2018, pp. 8000–8010.

[26] Y. Li, X. Lyu, N. Koren, L. Lyu, B. Li, and X. Ma, "Anti-backdoor learning: Training clean models on poisoned data," in *NeurIPS*, vol. 34, 2021.

[27] P. W. Koh and P. Liang, “Understanding black-box predictions via influence functions,” in *ICML*, 2017, pp. 1885–1894.

[28] X. Shao, A. Skryagin, W. Stammer, P. Schramowski, and K. Kersting, “Right for better reasons: Training differentiable models by constraining their influence functions,” in *AAAI*, vol. 35, no. 11, 2021, pp. 9533–9540.

[29] S. Kong, Y. Shen, and L. Huang, “Resolving training biases via influence-based data relabeling,” in *ICLR*, 2021.

[30] M. Goldblum, D. Tsipras, C. Xie, X. Chen, A. Schwarzschild, D. Song, A. Madry, B. Li, and T. Goldstein, “Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses,” *IEEE TPAMI*, 2022.

[31] N. Carlini and A. Terzis, “Poisoning and backdooring contrastive learning,” in *ICLR*, 2021.

[32] M. Pan, Y. Zeng, L. Lyu, X. Lin, and R. Jia, “Asset: Robust backdoor data detection across a multiplicity of deep learning paradigms,” *arXiv:2302.11408*, 2023.

[33] J. Steinhardt, *Robust learning: Information theory and algorithms*, 2018.

[34] A. F. Agarap, “Deep learning using rectified linear units (relu),” *arXiv:1803.08375*, 2018.

[35] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *ICCV*, 2015, pp. 1026–1034.

[36] E. Jang, S. Gu, and B. Poole, “Categorical reparameterization with gumbel-softmax,” *arXiv:1611.01144*, 2016.

[37] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” *NeurIPS*, vol. 32, pp. 8026–8037, 2019.

[38] R. Liu, J. Gao, J. Zhang, D. Meng, and Z. Lin, “Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond,” *arXiv:2101.11517*, 2021.

[39] R. Grazzi, L. Franceschi, M. Pontil, and S. Salzo, “On the iteration complexity of hypergradient computation,” in *ICML*, 2020, pp. 3748–3758.

[40] Y. Li, T. Zhai, B. Wu, Y. Jiang, Z. Li, and S. Xia, “Rethinking the trigger of backdoor attack,” *arXiv:2004.04692*, 2020.

[41] H. Qiu, Y. Zeng, S. Guo, T. Zhang, M. Qiu, and B. Thuraisingham, “Deepsweep: An evaluation framework for mitigating dnn backdoor attacks using data augmentation,” in *ASIACCS*, 2021, pp. 363–377.

[42] A. Krizhevsky, G. Hinton *et al.*, “Learning multiple layers of features from tiny images,” 2009.

[43] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition,” *Neural networks*, vol. 32, pp. 323–332, 2012.

[44] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar, “Attribute and simile classifiers for face verification,” in *ICCV*, 2009, pp. 365–372.

[45] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, “Imagenet large scale visual recognition challenge,” *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.

[46] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv:1609.04747*, 2016.

[47] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *ICLR*, Y. Bengio and Y. LeCun, Eds., 2015.

[48] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, “On the variance of the adaptive learning rate and beyond,” in *ICLR*, 2019.

[49] T. A. Nguyen and A. Tran, “Input-aware dynamic backdoor attack,” *NeurIPS*, vol. 33, pp. 3454–3464, 2020.

[50] Y. Liu, W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang, “Abs: Scanning neural networks for back-doors by artificial brain stimulation,” in *SIGSAC*, 2019, pp. 1265–1282.

[51] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, “Trojaning attack on neural networks,” in *NDSS*, 2018.

[52] S. Li, M. Xue, B. Zhao, H. Zhu, and X. Zhang, “Invisible backdoor attacks on deep neural networks via steganography and regularization,” *IEEE TDSC*, 2020.

[53] A. Saha, A. Subramanya, and H. Pirsiavash, “Hidden trigger backdoor attacks,” in *AAAI*, vol. 34, 2020, pp. 11 957–11 965.

[54] Y. Liu, X. Ma, J. Bailey, and F. Lu, “Reflection backdoor: A natural backdoor attack on deep neural networks,” in *ECCV, 2020*, 2020, pp. 182–199.

[55] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *CVPR*, 2016, pp. 770–778.

6 Appendix

6.1 Detailed Attack Settings in Section 2.3

We detail the 4 attacks in evaluating existing methods:

- For Label-only attacks, we use Targeted Label-Flipping [3]. We randomly selected 1000 "cat" samples and mislabeled them into "dog" samples to launch the attack. Compared to the standard Random Label-Flipping, which is considered in most label-noise defenses, our study finds that Targeted Label-Flipping is much harder to be mitigated.
- For Feature-only attacks, we consider two representative works, namely, Narcissus[4] [7], a clean-label backdoor attack, and Poison Frogs[5] [5], a targeted poisoning attack. For both attacks, we poison 10% of the dog-class samples.
- For the Label-Feature attacks, we consider BadNets[6] [8]. Following the setting in the original paper, we manipulate 5% of the CIFAR-10 by patching the trigger and changing the label to the target class.

6.2 Detailed Attack Settings in Section 2.4

We detail the 16 attacks in human studies as follows:

- For Label-only attacks, Random and Targeted Label-Flipping leads to mislabeled samples. However, since humans can only classify at a sample-wise level, these attacks present the same level of difficulty to humans. Hence, we only evaluate Random Label-Flipping, where we randomly shuffle a poisoned sample's label to a different class.
- For Feature-only attacks, we consider three representative works: Label-Consistent attack [7] [6], Hidden Trigger backdoor[8] [53], and the Narcissus attack [7]. We include three different designs of triggers from the original paper: vanilla Narcissus with $l_\infty = 16/255$; watermarked Narcissus, which used a mask to constrain the trigger within a pre-defined shape; and Narcissus-Smooth trigger, which is constrained to be low-frequency.
- For Label-Feature attacks, we study a comprehensive list of dirty-label backdoor attacks with different trigger designs: 1) Patch-based triggers that adopt an arbitrary trigger pattern in small regions: BadNets, and Masked random noise (Masked-Noise) [12], Watermark [41]; 2) Blending-based triggers that blend the trigger into the original image for better stealthiness: Blended random noise (Blended-Noise) [9] and Refool[9] [54]; 3) Attack based on elastic transformation: WaNet[10] [10]; 4) Optimized triggers within norm-balls: l_0 and l_2 invisible triggers [52]; 5) Optimized low-frequency triggers: the smooth trigger[11] [11]; 6) Input-specific triggers with trigger patterns customized to individual inputs: the input aware backdoor (IAB)[12] [49].

[4]https://github.com/ruoxi-jia-group/Narcissus
[5]https://github.com/LostOxygen/poison_froggo
[6]https://github.com/verazuo/badnets-pytorch
[7]https://github.com/MadryLab/label-consistent-backdoor-code
[8]https://github.com/UMBCvision/Hidden-Trigger-Backdoor-Attacks
[9]https://github.com/DreamtaleCore/Refool
[10]https://github.com/THUYimingLi/BackdoorBox
[11]https://github.com/YiZeng623/frequency-backdoor
[12]https://github.com/THUYimingLi/BackdoorBox

6.3 Pseudocode of the Training Stage

Algorithm 1: Training Algorithm of One Sifter

```
Input: $\theta$ (Classifier Model); $\psi$ (weight-assigning network);
       $\Gamma$ (meta-gradient-samplers);
       $D$ (dataset that requires sifting);
Output: $(\theta^*, \psi^*)$ (pair of parameters for one Sifter);
Parameters: $\alpha, \beta > 0$ (step sizes)
/* A.Warming-up $\theta$, i.e., normal training with
   all weights setting to 1 */
1 for each n-size mini-batch in $D$ do
2     $\theta = \theta - \alpha \times \frac{1}{n}\sum_{i=1}^{n}\frac{\partial L_i(\theta)}{\partial \theta}$ ;
/* B. Updating Process */
3 for each n-size mini-batch in $D$ do
      /* 1. Virtual-update */
4     Formulate gradients according to (14) ;
      /* 2. Gradient Sampling */
5     Gradient sampler updating and sampling (15) ;
      /* 3. Meta-update */
6     Updating $\psi$ with sampled hypergradients (18) ;
      /* 4. Actual-update */
7     Updating $\theta$ according to (19) ;
8 return $(\theta^*, \psi^*)$
```

6.4 Detailed Adaptive Designs in Section 4.4

Poisoning the Majority: This adaptive design simply poisons more than 50% of the samples following existing attack design. This attack design is effective as it breaks our basic assumption that the poisoned samples of each class account for less than 50% (Section 3.1). This attack design is less practical as manipulating the majority of the samples is hard.

Adversarial Noise using a Clean Model: Following the noise synthesis formulation in Eqn. (21), for each given poisoned sample in the original D, $z_{poi} = (x, y)$, we use SGD with the Adam optimizer and a learning rate of 0.01 for 100 rounds. We do not introduce additional norm constraints on the perturbation as we find that the converged perturbation does not introduce large visual artifacts. As we consider a white-box attack scenario, the model structure is the same as what we adopted for the feature extractor in Sifters.

Adversarial Noise using Sifters: Following the noise synthesis formulation in Eqn. (22), for each given poisoned sample, $z_{poi} = (x, y)$, we use stochastic gradient ascent with the Adam optimizer and a learning rate of 0.01 for 100 rounds. Note that we consider the original D to follow the same attack settings (poison ratio, trigger design, label manipulation, if applicable) as listed in Table 5, and the adaptive noise is adopted to perturb the original poisoned samples.

6.5 Additional Results (CIFAR-10)

Lower Poison Ratios. Table 4 in the main text uses the poison ratios reported in the original papers for these attacks. The attacks can achieve similar attack effects but with a smaller poison ratio. Primarily, many defense works have found attacks with lower poison ratios are harder to be identified [20, 50].

	Label-only		Feature-only			Label-Feature						
	Targeted Label-Flipping [3]	Random Label-Flipping [4]	Clean-Label [6]	Narcissus Backdoor [7]	Poison Frog [5]	BadNets One-Tar [8]	Smooth One-Tar [11]	IAB One-Tar [49]	Blended One-Tar [9]	BadNets All-to-all [8]	Smooth All-to-all [11]	Blended All-to-all [9]
Poison ratio	Tar: 2%	All: 5%	Tar: 1%	Tar: 1%	Tar: 1%	Tar: 9.1%	Tar: 9.1%	Tar: 9.1%	Tar: 9.1%	All: 5%	All: 5%	All: 5%
META-SIFT	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**
Poison ratio	Tar: 9%	All: 10%	Tar: 5%	Tar: 5%	Tar: 5%	Tar: 16.7%	Tar: 16.7%	Tar: 16.7%	Tar: 16.7%	All: 10%	All: 10%	All: 10%
META-SIFT	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**

Table 11: NCR results of our method under the CIFAR-10 settings with lower poison ratios.

	Label-only		Feature-only			Label-Feature						
	Targeted Label-Flipping [3]	Random Label-Flipping [4]	Clean-Label [6]	Narcissus Backdoor [7]	Poison Frog [5]	BadNets One-Tar [8]	Smooth One-Tar [11]	IAB One-Tar [49]	Blended One-Tar [9]	BadNets All-to-all [8]	Smooth All-to-all [11]	Blended All-to-all [9]
Poison ratio	Tar: 40%	All: 40%	Tar: 40%	Tar: 40%	Tar: 40%	Tar: 40.2%	Tar: 40.2%	Tar: 40.2%	Tar: 40.2%	All: 40%	All: 40%	All: 40%
DCM	**63.0**	94.3	37.8	**102.0**	**112.2**	98.3	**27.5**	**104.6**	**54.6**	78.2	**27.6**	63.8
MI-DCM*	71.2±10.5	**100.7±8.8**	93.5±0	**362.0±88.6**	**311.0±76.9**	**131.5±15.6**	**168.6±20.9**	**163.8±30.1**	**153.6±25.5**	**117.0±19.6**	**153.8±16.8**	**137.3±26.3**
SF-Least*	97.4±1.67	**31.2±4.63**	86.3±3.45	**43.6±0**	**106.3±9.88**	**32.4±0.0**	28.3±1.67	**44.6±3.53**	60.3±2.71	**56.7±0**	58.9±0	**63.2±6.72**
Loss-Scan*	**173.9±16.9**	**443.1±57.8**	**163.6±23.4**	**188.0±41.1**	**134.0±23.8**	83.6±16.3	**116.7±22.9**	**106.9±31.6**	**142.6±13.8**	96.7±19.6	85.8±9.8	91.6±16.6
Self-IF*	**135.9±23.6**	**154.6±33.7**	**0.0±0.0**	**336.7±85.7**	**56.8±14.6**	39.8±6.8	59.6±10.0	**117.6±22.4**	88.4±26.3	**132.4±33.6**	**154.9±8.7**	**146.5±25.6**
META-SIFT	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**

Table 12: NCR results under the CIFAR-10 settings with higher poison ratios.

	Label-only		Feature-only			Label-Feature						
	Targeted Label-Flipping [3]	Random Label-Flipping [4]	Clean-Label [6]	Narcissus Backdoor [7]	Poison Frog [5]	BadNets One-Tar [8]	Smooth One-Tar [11]	IAB One-Tar [49]	Blended One-Tar [9]	BadNets All-to-all [8]	Smooth All-to-all [11]	Blended All-to-all [9]
Poison ratio	Tar: 50%	All: 50%	Tar: 50%	Tar: 50%	Tar: 50%	Tar: 50%	Tar: 50%	Tar: 50%	Tar: 50%	All: 50%	All: 50%	All: 50%
META-SIFT	**35.2±21.4**	**23.6±11.6**	**46.1±15.4**	**58.9±20.1**	**38.6±16.2**	**53.6±14.3**	**54.2±15.3**	**63.1±19.2**	**46.4±10.8**	**56.3±13.7**	**53.6±13.3**	**43.7±10.3**

Table 13: NCR results of our method when poisons are of the majority under the CIFAR-10 settings.

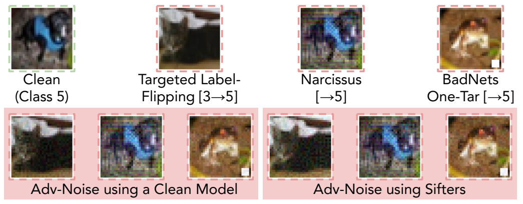


Figure 7: Visual results of the original poisoned samples and the adversarial noise disguised poisoned samples (CIFAR-10).

	Targeted Label-Flipping [3]	Narcissus Backdoor [7]	BadNets One-Tar [8]
Cln-Model-Based Adv-Noise (%)	ACC: 92.01 ASR: 6.35 Tar-ACC: 88.08	ACC: 93.31 ASR: 5.78	ACC: 85.36 ASR: 88.92
META-SIFT	**0±0**	**0±0**	**0±0**
Sifter-Based Adv-Noise (%)	ACC: 91.56 ASR: 45.71 Tar-ACC: 82.51	ACC: 93.19 ASR: 9.81	ACC: 84.33 ASR: 91.64
META-SIFT	**0±0**	**0±0**	**0±0**

Table 14: Adversarial-noise-based adaptive attacks' effects on the original attack results and the results of META-SIFT .

Thus, we present additional results of META-SIFT on CIFAR-10 with smaller poison ratios in Table 11. In particular, for each evaluated attack, we re-examine META-SIFT with two settings of smaller poison ratios (half of or quarter of the original poison ratio) listed in Table 11. We found META-SIFT can still reliably provide high-precision sifting results.

Higher Poison Ratios. In addition to lower poison ratios, we also evaluate whether META-SIFT 's effectiveness can extend to high poison ratios. We tested the competitors and our proposed solution on corrupted datasets with higher poison ratios (40%) in Table 12. A higher poison ratio results in a significant decrease in the performance of methods that utilize model output features, such as DCM, MI-DCM, and SF-Least. This is because as the poison ratio gets higher, the features of poisoned samples are getting more diverse. At the same time, the inclusion of a larger number of poisoned samples in the optimization process results in a heightened difficulty in reducing the average loss of poison samples. As a result, the efficacy of the Loss-Scan method is also impacted. However, we found META-SIFT generates a completely clean base set consistently under the evaluated higher poison ratios.

Poisoning the Majority. Elaborated in Section 3.1, we assume that the number of poison samples should be less than that of clean samples. This assumption is plausible under normal circumstances. Now we study how our method will perform in an extreme case where the attacker controls more than half of the samples. We evaluate the performance of our method through a series of experiments with each attack using a large poison ratio over 50% in Table 13. Noting this attack setting is a plausible but less practical adaptive attack as discussed in Section 4.4. The results show that our defense can no longer sift out a purely clean base set since the underlying assumption is not met. Nevertheless, poisoning the majority of the dataset in practice is hard, e.g., to poison the training set of CLIP [1] requires the attacker to at least have access and be able to manipulate 200 million image-text pairs.

Adaptive Attacks via Adversarial Noise. For the clean-model-based design, we first train θ_{cln} using the clean portion of D, achieving an average training loss of 0.001. Then, we use this model to synthesize the noise following Eqn. (21), for each z_{poi}. The average loss over these poisoned samples dropped from 5.135 to 0.0015 (with $\tilde{z}_{poi}$). For the Sifter-based design, we plug 5 trained Sifters (which can sift out a clean base set of size 1000 from D) into Eqn. (22) for noise synthesis. Then, with the updated dataset $\tilde{D}$, we find the old five Sifters trained using D will select only poisons for the target class to form the base set. Both results indicate that the synthesized noise aligns with our optimization goals, as we expected. The poisoned samples perturbed by the synthesized noises are visualized in Figure 7. The attack performance on models trained over $\tilde{D}$ and the sifting results with META-SIFT are shown in Table 14. Despite the noises being effective in achieving low loss on the clean model or being selected by the old Sifters, META-SIFT remains a 0% NCR when applied to $\tilde{D}$ that contains adaptively designed poisons.

6.6 Additional Results (GTSRB, PubFig)

We further study the sifting effectiveness of different methods over the popular traffic signs dataset GTSRB in Table 15 and the real-world face dataset PubFig in Table 16. These

	Label-only		Feature-only		Label-Feature							Overhead (s)
	Targeted Label-Flipping [3]	Random Label-Flipping [4]	Narcissus Backdoor [7]	Poison Frog [5]	BadNets One-Tar [8]	Smooth One-Tar [11]	IAB One-Tar [49]	Blended One-Tar [9]	BadNets All-to-all [8]	Smooth All-to-all [11]	Blended All-to-all [9]	
DCM	**103.1**	**107.5**	37.7	75.5	29.6	44.4	87.0	**44.4**	38.5	**121.0**	**108.0**	10
MI-DCM*	28.1±0	**112.5±0**	**1000±0**	**528.3±0**	51.9±0	**196.3±0**	**105.1**	**108.9±0**	**118.5±0**	30.5±0	41.2±0	480+300
SF-Least*	**0.0±0**	**18.5±1.73**	75.47±0	**113.21±0**	74.1±0	62.96±0	63.0±0	54.8±0	**1.0±0**	**3.33±0.58**	**3.33±0**	480+15
Loss-Scan*	**347.9±56.4**	**507.0±0.0**	**32.6±2.35**	**10±0.0**	**23.7±6.73**	**23.7±5.45**	**45.0±6.47**	63.7±2.67	**325.0±56.1**	**285.5±104.3**	**271.6±57.9**	110
Self-IF*	**128.1±5.41**	**107.5±0**	**125.8±10.9**	**245.3±0**	37.0±7.41	51.9±0	84.0±0	54.8±1.21	**110.5±0**	**105.5±0**	**102.2±0**	**480+17432**
META-SIFT	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	5×80

Table 15: NCR results under the GTSRB settings.

	Label-only		Feature-only		Label-Feature					Overhead (s)
	Targeted Label-Flipping [3]	Random Label-Flipping [4]	Narcissus Backdoor [7]	Poison Frog [5]	BadNets One-Tar [8]	IAB One-Tar [49]	Blended One-Tar [9]	BadNets All-to-all [8]	Blended All-to-all [9]	
DCM	**173.3**	**97.1**	**135.0**	90.0	74.1	**24.0**	88.9	**148.1**	**135.1**	10
MI-DCM*	**38.4±0**	**100.3±5.83**	**74.1±0**	86.4±21.4	**167.5±4.33**	81.1±0	**142.5±26.0**	55.5±9.6	**41.3±2.25**	2100+300
SF-Least*	**160.3±11.1**	**187.3±8.69**	**74.1±0**	**111.1±0**	87.5±4.33	78.0±4.33	**122.5±4.33**	**40.3±1.26**	92.8±4.37	2100+15
Loss-Scan*	**167.9±56.3**	**438.5±118.3**	90.0±0	**60.0±0**	**63.0±9.87**	54.1±6.78	**56.7±6.57**	**193.5±56.1**	**192.8±16.9**	140
Self-IF*	**179.5±61.8**	**108.7±6.83**	**74.1±0**	86.4±21.4	85.0±11.6	63.0±11.6	57.5±11.5	**111.2±12.8**	**121.2±8.39**	**2100+21045**
META-SIFT	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	**0±0**	5×80

Table 16: NCR results under the PubFig settings.

datasets are more representative of ML tasks encountered in real-world applications. Both GTSRB and PubFig are imbalanced, unlike CIFAR-10. Therefore, we maintain the same class distribution when sifting the samples from each class. Again, similar to the observation in the main text, none of the baselines can constantly produce a fully clean base set for these two datasets. Even with datasets more challenging than CIFAR-10, META-SIFT continues to deliver a completely clean base set with zero variance in each evaluated attack setting and also is faster than the other baselines.

6.7 Ablation Study

Throughout this study, we compare the empirical upper bound on the size of the clean subset sifted under varying parameter settings. Note that this upper bound value implies that for given parameters, META-SIFT can sift out a clean base set (of size equal to the upper bound) with 100% precision.

# Sifter	1	3	5	10
Clean subset size	155(×10)	274(×10)	517(×10)	533(×10)

Table 17: Sifting performance vs. number of Sifters.

First, we study the effect of adding additional Sifters to select the clean base set (Table 17). We consider the CIFAR-10 [42] for all ablation studies. The hyperparameter settings remain as declared in Table 3, except we use the ResNet-18 [55] architecture. We poison the dataset with the BadNets [8] in this study with a 5% poison ratio into target class 5. We observe that as the number of Sifters increases, the upper bound on the base set size also increases. The subset size grows sharply from one to five Sifters, i.e., from 155(×10) to 517(×10). Although the subset size keeps growing after five Sifters, the pace is comparatively gradual.

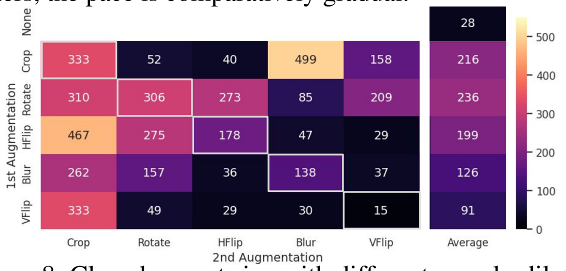


Figure 8: Clean base set size with different sample-dilution settings. The last column reflects the average over the row.

Secondly, we discuss the impact of different random augmentations used in sample-dilution towards the size of achievable clean base set (Figure 8). We adopt the widely used

	Without dilution	With dilution	Change
DCM	7(×10)	3(×10)	**-4**(×10)
MI-DCM*	12(×10)	9(×10)	**-3**(×10)
SF-Least*	11(×10)	15(×10)	**+4**(×10)
Loss-Scan*	6(×10)	2(×10)	**-4**(×10)
Self-IF*	8(×10)	14(×10)	**+6**(×10)

Table 18: Baseline results with sample-dilution.

augmentation settings for all experiments throughout this paper. Specifically, for Random Crop, the image size is 32px, and the padding size is 4px; for Gaussian Blur, the kernel size is 3px; for Random Horizontal & Vertical Flip, the probability is 0.5. From Figure 8, we observe that different augmentations impact the size of the clean base set greatly. For example, when only applying Random Horizontal Flip, we can acquire a clean set of 178(×10) samples. However, the sole application of Random Vertical Flip results in a size of only 15(×10) samples. We also found combinations of the augmentations lead to better performance, e.g., combining Random Crop with Gaussian blur increases the size from 333(×10) to 499(×10). In our evaluation, we use no more than four random augmentations in the sample-dilution since too many augmentations may lead to large information loss. Note that, without any augmentation being adopted, our method can still sift out a clean base set of 28(×10). In comparison, other baselines' best results with or without sample dilution is only 15(×10), Table 18. It also goes without saying that these baseline approaches have inconsistent performance against different types of attacks, and they might even have worse performance against other settings of attacks (Table 1,5,6,15,16).

6.8 Visual Examples of the Selected Points.

Figure 9 show examples of data selected by the META-SIFT or random-sampled. The META-SIFT -selected samples exhibit slightly better semantic information for the label In comparison. However, it is hard to draw conclusions about sample consistency based solely on visual observations

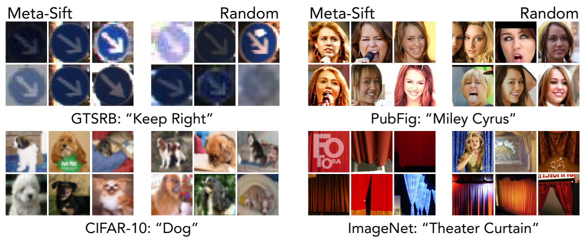


Figure 9: Selected sample comparison.