Exploring Training Provenance for Clues of Data Poisoning in Machine Learning

by

Jon-Nicklaus Z. Jackson

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
(Cybersecurity and Information Assurance)
in The University of Michigan-Dearborn
2023

Master's Thesis Committee:

Assistant Professor Birhanu Eshete, Chair
Assistant Professor Jin Lu
Assistant Professor Probir Roy

Dedication

I would like to dedicate my thesis work to my friends and family that have been incredibly supportive throughout this process. I am extremely grateful to those that helped make this possible given the hardships my family has experienced the past few years. I would not have been able to pursue my Master's degree without their support and encouragement. I would also like to recognize all teaching staff of the department of Cybersecurity and Information Assurance for their efforts in ensuring I had the best learning experience throughout these two years. I also dedicate this thesis to my manager and mentor Geoff for giving me the opportunity to intern with Robert Bosch LLC, and for taking a personal interest in seeing how I have been doing in my coursework and research. A big thank you to all my friends, family, professors, and mentors for making this quite the learning experience and journey.

Acknowledgements

First and foremost, I would like to acknowledge and thank my advisor Professor Birhanu Eshete for his counsel, contributions, and uncompromising efforts toward making this thesis a success. I would also like to thank my longtime girlfriend Claudia, who has supported me in my pursuit of my Master's degree. In addition to my longtime girlfriend, I would like to acknowledge my cousin Giselle and her husband Ruben for their encouragement and advise on pursuing a Master's degree with Thesis.

Table of Contents

List of Figures

Figure

List of Tables

Table

Abstract

Machine Learning today plays a vital role in a wide range of critical applications. To ensure ML models are consistently able to produce correct output, such models must be retrained as new input samples become available to avoid performance degradation as a result of data drift. During retraining, such models are left vulnerable to possible injection of poisonous samples via *data poisoning attacks*, where adversaries manipulate the training data to have the ML model misbehave to serve adversarial goals. Inspired by previous works that leverage *data provenance* for detecting and filtering out poisonous data from the training set, we adapt the definition of provenance for what we define as *training provenance*, the history of training metrics captured over training time-frame. We then build a framework that allows us to capture both key performance metrics for the overall dataset and per-class metrics for each label at every epoch as training provenance. Through exploratory analysis of captured training provenance we aim to find clues that standout to serve as strong signals for making a call on training data poisoning. We evaluate our proposed framework on two benchmark image classification datasets: MNIST and CIFAR-10. For MNIST, we observed promising signal(s) for establishing a per-epoch poisoning detection threshold based on captured metrics at the overall dataset level. For CIFAR-10, captured overall dataset metrics as training provenance for clean and poisoned training data were not as effective as compared to our observations for MNIST experiments. As for captured per-class metrics, we discovered that these metrics provided little insight as a result of the nondeterministic nature of machine learning. Overall, we observe that this is a promising direction that invites further exploration with more poisoning attacks and diverse datasets.

Keywords: Training Data Poisoning, Data Provenance, Machine Learning.

Chapter 1: Introduction

Machine Learning (ML) today plays a vital role in a wide range of critical applications from malware detection to autonomous vehicles, facial recognition to financial predictive models. In an effort to ensure ML models within these domains are consistently able to produce correct output, such models must be retrained as new input samples become available in order to avoid performance degradation as a result of data drift. This leads ML model owners to retrain their model with data collected through various sources. However, this opens them up to possible *data poisoning attacks* [1, 4, 8, 14, 19, 21, 25, 26], where adversaries manipulate the training data in order to have the ML model misbehave in favor of the adversarial goals.

Prior work: To combat against these attacks, previous work has proposed both *provenance-based* [2, 3, 23] and *non-provenance-based* [5, 6, 18, 20, 24] defenses. Inspired by [2, 3] which leverage *data provenance*, "the lineage or meta-data associated with a data point that shows the operations that led to its creation, origin, and derivation," for detecting and filtering out poisonous data from the training set, we adapt their definition of provenance for what we define as *training provenance*, the history of common training metrics (e.g., loss, accuracy) captured over training time-frame. In positioning our work with respect to these related works, it is important to note that our work is not a proposed defense rather we set the groundwork for establishing threshold detection by exploring what we characterize as training provenance for clues of data poisoning. We also note that our exploratory approach of training provenance capture relies on metrics that are freely available in modern-day ML frameworks and the main focus is to go beyond metadata / data source trustworthiness, and capture the actual training progression in pursuit of clues for poisoning.

Approach Overview: With the main objective of exploring training provenance for establishing detection threshold against data poisoning attacks, we build a framework that allows us to capture both key performance metrics (e.g., loss, accuracy) for the overall dataset and per-class metrics (e.g., precision, recall) for each label at every epoch as training provenance. Having the capability to determine whether or not the data used in the retraining of a model is poisonous,

a model owner can then decide whether it is safe to deploy the current model instance. Within our framework we first initially train our ML model on original clean dataset to serve as our baseline. We then retrain our model on the same dataset with additional clean samples generated via *data augmentation* [22], and again on the same dataset, but with triggered samples generated via simulated data poisoning attack. To discover potential clues of poisoning, we compare the training provenance captured for each of these datasets in our training provenance-based exploratory analysis where we analyze via pairwise, our per-epoch comparison of (original, clean-augmented) and (original, triggered) at the overall dataset level as well as the per-class level. In addition to our per-epoch comparison, we calculate distance at epoch for (original, clean-augmented) and (original, triggered) with the goal of finding clues (consistent patterns) that standout to serve as strong signals for making a call on training data poisoning.

Evaluation Overview: We evaluate our framework on two benchmark image classification datasets: MNIST and CIFAR-10. For MNIST, we simulate an "All-to-All" attack as described in [8] where our ML model is retrained on a portion of poisonous samples for each label leading to our model misclassifying these samples into the target label of the adversary's choosing instead of their correct label. For CIFAR-10, we simulate a "Single Target" attack where the only difference is that one particular label is targeted. For each dataset, we consider injecting different percentages of input samples to see how this impacts the captured training trajectory of ML model. We base our evaluation on the following criteria: (*i*) the effectiveness of captured metrics characterized as training provenance at the overall dataset and per-class level, (*ii*) whether training provenance can be leveraged to establish a threshold for detecting poisoning, (*iii*) whether per-class metrics can provide us with further insight such that we can attribute the ML model's mistakes (e.g., incorrect predictions) to the portion of poisonous data injected in the training set during the retraining process, and (*iv*) whether the results we obtain are stable across multiple runs of our framework.

For our MNIST experiments, we observe consistently larger distance between triggered and original MNIST when compared with the distance between clean-augmented and original MNIST; a promising signal for establishing a per-epoch poisoning detection threshold. As for captured per-class metrics, we were only able to attribute the model's mistakes to the smallest percentage of poisonous data injected to the training set upon retraining. For CIFAR-10 experiments, captured overall dataset metrics (e.g., loss, accuracy) as training provenance for clean and poisoned training data were not as effective as compared to our observations for MNIST experiments. There were

no clear distinctions between captured training provenance of CIFAR-10 dataset with clean-augmented input samples and training provenance of CIFAR-10 dataset with triggered input samples as the capture metric values tended to fluctuate throughout the training process and as more input samples were added. We observe similar fluctuations when evaluating the effectiveness of per-class metrics captured as training provenance. While not as promising as our results for MNIST, we believe our results for CIFAR-10 were heavily impacted by the augmentation and trigger disrupting the original data distribution significantly enough that our observations fluctuated over multiple (e.g., 30) runs.

Contributions: In the space of provenance-based defenses against data poisoning attacks, this work makes the following contributions:

- Building on and complementary to prior provenance-based poisoning defenses [2, 3, 23], we articulate the definition for training provenance focused on capturing training history as training unfolds.
- We propose a framework that captures and analyzes training provenance at the overall dataset level (e.g., loss, accuracy) and at the per-class level (e.g., precision, recall) for clues of training data poisoning.
- An evaluation of our proposed framework on two benchmark datasets, MNIST and CIFAR-10, and two variations of BadNets backdoor attack [8].

Thesis Organization: The rest of this thesis report is organized as follows. In Chapter 2, we cover background on ML preliminaries, data poisoning, and provenance. In Chapter 3, we present our threat model and problem statement. In Chapter 4, we survey related literature to our work. In Chapter 5, we describe the details of our Training Provenance-Based Exploration Framework. In Chapter 6, we evaluate the effectiveness of our framework focusing on overall dataset effectiveness, per-class effectiveness, and stability of results over multiple runs of the framework. Finally, we conclude with insights gained from our study and identify future work in Chapter 7.

Chapter 2: Background

To ease forthcoming discussions, this chapter first defines terminology and notations relating to machine learning, training data poisoning attacks, and data provenance.

2.1 Machine Learning Preliminaries

Supervised ML Training. We focus on supervised machine learning models. Given a set of labeled training samples $X_{train} = (X_i, y_i) : i \leq n$, where X_i is a training example and y_i is the corresponding label, the objective of training a ML model f_θ, parameterized by a weight vector θ, is to minimize the expected loss over all (X_i, y_i) computed as:

$$J(\theta) = \frac{1}{n}\sum_{1}^{n} l(\theta, X_i, y_i) \tag{2.1}$$

In ML models such as deep neural networks (DNNs), the loss minimization problem is typically solved using stochastic gradient descent (SGD) by iteratively updating θ as:

$$\theta = \theta - \epsilon \cdot \Delta_\theta \sum_{i=1}^{n} l(\theta, X_i, y_i) \tag{2.2}$$

where Δ_θ is the gradient of the loss with respect to the weights θ; X is a randomly selected set (e.g., a *mini-batch*) of training examples drawn from X_{train}; and ϵ is the *learning rate* which controls the magnitude of change on θ.

Typical ML Testing: Let X be a d-dimensional feature space and Y be a k-dimensional output space, with underlying probability distribution $Pr(X, Y)$, where X and Y are random variables for the feature vectors and the classes (labels) of data, respectively. The objective of testing a ML model is to perform the mapping $f_\theta : X \rightarrow Y$. The output of f_θ is a k-dimensional vector and each dimension represents the probability of input belonging to the corresponding class/label.

2.2 Training Data Poisoning Attacks

Machine learning is vulnerable to adversarial manipulation both at test time (e.g., via adversarial examples [7]) and during training time (via poisoning attacks [1, 4, 8, 14, 19, 21, 25, 26]). At the training stage, adversaries can manipulate a subset of training data to influence model behavior (e.g., misclassifying certain inputs targeted by the attacker). Previous research has shown that poisoning is realized in different ways:

- In *integrity poisoning attacks* [10], the goal of the poisoning is to have samples pertaining to a given class misclassified into any other class other than its own.
- In *availability poisoning attacks* [10], the goal of the poisoning to decrease the overall accuracy of the model so that the model's utility degrades as a result.
- *Targeted poisoning attacks* [10] aim at making a model misclassify a single datapoint or small set of data-points (e.g., misclassify "cat" as "dog").
- *Indiscriminate poisoning attacks* [10] aim at making a model misclassify general class of points.
- In our work, the poisoning attack that we simulate is a backdoor poisoning attack. In *backdoor poisoning attacks* [8], the adversary's goal is to plant a backdoor (or watermark) pattern called a *trigger* that forces the backdoored model to learn to recognize a trigger known only by the adversary. The planting of the trigger is typically done by injecting the trigger via manipulation of features (e.g., pixels in images) in a manner that influences the model's behavior (e.g., prediction result). The attack aims to have the model misclassify input samples with injected triggers into targeted labels while having little/no impact on the prediction results of normal (trigger-free) input samples; thereby, making it extremely difficult for one to detect.

2.3 Data Provenance vs. Training Provenance

As defined in [2, 3], Data Provenance is "the lineage or meta-data associated with a data point and shows the operations that led to its creation, origin, and derivation. This may include information about the device from which the data was gathered, its firmware version, user id, and timestamp among others". Provenance has been widely used in systems/network security to capture causal linkage and provenance of host/network entities (e.g., processes, files, network sockets) to detect and forensically analyze sophisticated and stealthy cyber-attacks [9, 16, 17].

In Chapter 4 we present previous proposed defenses against data poisoning attacks that have inspired our interest in investigating whether or not training provenance can be leveraged to establish threshold detection against such attacks. In particular, we draw inspiration from [2, 3] as these defenses leverage data provenance for detecting and filtering out poisonous data from the training set.

While we are inspired by the likes of [2, 3], the notion of training provenance in our work is centered around the key idea of *capturing training artifacts within the time window of training* so as to (a) characterize training progression/history as training provenance and (b) explore the use of *training provenance* for establishing threshold detection against data poisoning attacks. More precisely, we capture training provenance as the history of common training metrics (e.g., loss, accuracy) captured over training time-frame, detailing how these metrics change as new input samples are introduced in the retraining process of one's ML model.

Chapter 3: Threat Model and Problem Statement

In this chapter, we set the scene for the rest of the thesis by presenting (a) a motivational scenario inspired by real-world training data poisoning risks; (b) our threat model; and (c) problem statement.

Motivational Scenario. A model owner trains a model (e.g., spam filter *f*) on training data *D* at time *t*. They deploy *f* as a prediction API (e.g., integrated to an email service). Once in a while (e.g., every month), up on acquiring a new batch of data-points (i.e., email samples) *D′*, the model owner needs to retrain *f* on *D* + *D′*. Retraining is needed, for instance, to keep up with the latest adversarial tactics or to improve *f*'s accuracy. We note that data-points in *D′* could be acquired from potentially untrustworthy/compromised sources. In this scenario, the model owner is confronted with the question *"how do I verify that this new chunk of new training samples D′ is not going to poison my original training data D?"*

3.1 Threat Model

In the context of the motivational scenario, our goal is to explore whether training provenance serves as a basis to give clues of data poisoning attacks where a fraction (say 5%) of the training data is manipulated by an adversary to either result in an overall accuracy drop or a targeted poisoning that aims to skew prediction accuracy of one or more labels. We have two parties, a model owner and data poisoning adversary. In the following, we describe their goals, capabilities, and our assumption about each party.

Model owner. The goal of the model owner is, once *f* is deployed, to periodically retrain *f* by augmenting the original training data *D* with user-supplied test samples *D′*. The size and distribution of *D′* is determined with respect to the size and distribution of *D* and is configurable by the model owner. The frequency of retraining is also assumed to be decided by the model owner depending on whether there is a need to do so (e.g., enough user-supplied test inputs that could improve model accuracy on retraining). In addition, we assume that the model owner:

- has access to the original (non-poisoned) training data *D* and the model *f* trained on it;

- has access to a trusted computing base (e.g., GPUs, ML frameworks) that is not compromised and remains the same (in terms of specifications) between successive retrainings of the model;
- has access to secure storage of all training artifacts with their integrity intact.
- backups logs of captured training provenance regularly as a recovery mechanism in the unlikely event an adversary gains access to the location where these logs are kept and deletes such logs.

Data Poisoning Adversary. In line with common assumptions made by existing work on poisoning attacks, we assume that the adversary:

- has the capability to manipulate any portion of the newly acquired training points (i.e., samples in D') which will be merged with D to retrain f;
- can poison in the worst case 20% of the training dataset used to retrain the model;
- has no access to the model architecture and its parameters, including the logging of training provenance;
- may challenge an anti-poisoning defense deployed by the model owner by capturing the same type of training provenance metrics that the model owner captures. We note, however, that the adversary is limited in knowing the dataset distribution for which the model owner will be retraining their model on. The distribution of the dataset would change according to the percentages of samples being added to the original dataset D, which is typically under the control of the model owner;
- does not have knowledge of the exact poisoning attack or set of attacks used in the model owner's pipeline for establishing poisoning detection threshold based on the comparison of captured training provenance for clean training set and poisoned training set.

3.2 Problem Statement

Given a training data D from which model f is trained, let D' be a separate dataset that is disjoint with D and is a small fraction (e.g., 5%-10%) of D. Our goal is to leverage the training process/trajectory/history, which we call *training provenance*, to characterize and capture "what happened in the process of learning f from D'. More precisely, we aim to explore the utility of training provenance in pursuit of clues indicative of whether or not a model retrained on $D+D'$ is poisoned by D'. A natural challenge to tackle here stems from the innate nondeterminism of ML

training computations (after all the canonical training algorithm of modern day DNNs is *stochastic* gradient descent). In addition, other contributors to this randomness are factors such as mini-batch selection, random model weights initialization, and potential data distribution shift when D is augmented with D'.

Chapter 4: Related Work

In this chapter, we survey related literature in training data poisoning defenses and position our work with respect to closely related work of provenance-based data poisoning detection.

4.1 Provenance-Based Defenses

In [2], the defense proposed by Baracaldo et al. leverages data provenance to proactively detect and filter out poisonous data prior to training the ML model. The approach is tested on two different types of data: partially trusted data, and fully untrusted data. Each data point in the training dataset is linked with its "provenance record" or its meta-data. Based on feature values or *provenance signatures* shared, untrusted data points are divided into groups. For each of these groups, the ML model is trained with and without the group. The methodology applied to the second type of data differs slightly in its implementation in that the dataset is randomly split into training and validation sets, two ML models are trained, one with all the training data and the other without a given group, and the models are evaluated without the corresponding group. The performance of the model trained on the filtered (without) dataset is then compared to the performance of the model trained on the unfiltered (with) dataset. If the filtered model outperforms the unfiltered model, then the group of untrusted data is determined to be poisonous and it is removed from the untrusted dataset.

Evaluation of the approach suggests that it greatly improves the classification performance of models on both datasets used in the evaluation. Furthermore, the results suggest that their approach outperforms *Reject On Negative Impact* (RONI) [18], where RONI requires the ML model to be trained at least once for each data point, while [2] only requires that the ML model be retrained to a fraction of total number of untrusted points.

In [3], Baracaldo et al. extend their original work [2] by considering collusion attacks as in their original work, they assumed data sources to be independent of each other. Moreover, they present "a complete analysis of both partially and fully untrusted data defenses and evaluation" where in their original work they evaluate their proposed defense solely on partially trusted datasets.

In [23], Stokes et al. propose VAMP for defending against ML poisoning attacks via "cryptographically authenticated provenance" of machine learning objects. These objects are the training and validation datasets, the software used for training and evaluating the ML model, and the ML model itself. In their approach pipeline, the publisher of an object (e.g., training dataset) would generate corresponding metadata and embed it in a data structure, referred to in the paper as a *manifest*. The publisher would then bind the manifest to the object through cryptographic hashing and sign it with their private key prior to uploading the manifest to online database, or embedding it in the object. Optionally, the manifest can also be added into "an immutable, distributed ledger ... provid[ing] publicly available evidence" ensuring the manifest has not been tampered with.

By cryptographically binding the manifest to the object and signing it, one can then authenticate the source of the machine learning object and ensure the object's integrity has not been compromised prior to training the model. Furthermore, the metadata contained within the manifest would allow one to "trace back the machine learning object from the final prediction score or inference object back to all of its original components."

4.2 Non-Provenance-Based Defenses

In [18], Nelson et al. "demonstrate how attackers can exploit machine learning to subvert the SpamBayes statistical spam filer" and propose a potential defense to these attacks. The attacks presented in this paper include three *causative attacks* that influence the ML model's learning by controlling some portion of the training data. The goal of these attacks is to degrade the filter's performance to properly classify incoming messages. Two of these attacks, *dictionary attack* and *focused attack* impact the filter's availability by rendering the filter unusable. The *dictionary attack* aims to make the spam filter unusable by having legitimate emails indiscriminately misclassified as spam whereas *focused attack* aims to have targeted legitimate emails misclassified as spam. The third attack, *causative integrity attack - pseudospam*, aims to have spam messages misclassified as legitimate by directly manipulating the filter's training data.

As a potential defense to these attacks, Nelson et al. propose *Reject On Negative Impact* (RONI). RONI assesses the impact each sample has on training and doesn't train on those that have a large negative impact. This is done by testing the performance difference of the filter with and without the sample in the training set. The defense is successful in defending against *dictionary*

attacks as described in the paragraph above; however, it fails against *focused attack* emails since this attack targets future messages. Furthermore, one limitation of their proposed defense is that it can only be applied in environments where trusted data is available.

In [5], Chakarov et al. propose "an approach based in Pearl's theory of causation, and" the use of Pearl's Probability of Sufficiency (PS) for scoring and ranking training points that are most likely the root cause of a ML model incorrectly classifying samples. The primary intuition behind their work is motivated by debugging ML tasks where potential errors in training data are present. Similar to RONI, PS can only be applied in environments where trusted data is available.

In [24], Wang et al. propose a "robust and generalizable detection and mitigation system for DNN backdoor attacks." Given a trained DNN model, the authors aim "to identify if there is an input *trigger* that would produce misclassified results when added to an input, what that trigger looks like, and how to mitigate, i.e., remove it from the model." In order to identify if a trigger is present in an input sample, the authors consider "'minimal' trigger" needed to have a DNN misclassify samples from other labels into a given label. After determining all potential triggers for each label, the authors gauge the size of these triggers in order to find outliers, smallest potential triggers, as these would closely represent the real trigger. The authors consider this "*reverse engineered trigger*." In turn, this helps them to determine the target label of the backdoor attack. With the reverse engineered trigger identified, the authors propose three mitigation techniques: "*filter for detecting adversarial inputs*, *patching DNN via Neuron Pruning*, and *patching DNN via Unlearning*."

In [6], Chen et al. propose "*Activation Clustering* method for detecting poisonous training samples crafted to insert backdoors into DNNs." The primary intuition behind this approach is in the way the model makes decisions for classifying legitimate samples vs. backdoor samples, which can be observed via neuron activations of the "last hidden layer." These activations are "segmented according to their label" with each segment "clustered separately." The legitimate and poisonous activations are then divided into two clusters via k-means. To determine which of these clusters "corresponds to poisonous data," Chen et al. propose the following three cluster analysis methods: *Exclusionary Reclassification*, *Relative Size Comparison*, and *Silhouette Score*, details of each method is in the paper [6].

Once a cluster is determined to be poisonous, or not, Chen et al. further propose verification techniques to confirm the corresponding data is in fact poisonous. For image datasets, this consist

of "constructing image sprites for each cluster and averaging the images for the activations in the cluster." For text datasets, this consists of "using Latent Dirichlet allocation (LDA) to identify the primary topics in each cluster." After identifying and verifying that the data is poisonous, the model can be repaired by removing the data and retraining the model, or by relabeling the data "with its source class and continu[ing] to train the model on these samples until it converges again."

Compared to other methods, the method of clustering activations proposed by Chen et al. does not require verified, trusted data and has been evaluated on both text and image datasets.

In [20], Shan et al. propose a forensic traceback tool for identifying poisoned input samples in the training set that are responsible for an observed "misclassification event." The approach consist of "iteratively clustering and pruning ... 'innocent' training samples until all that remains is the set of poisoned data responsible for the attack." Data samples are separated into two clusters via Mini-Batch K-means "based on their impact on model parameters" with one cluster solely consisting of benign samples and the other being a mixture of poisoned and benign samples. The cluster is determined to be benign, or poisonous via comparison of the cross-entropy loss obtained when the DNN is trained on the full dataset versus the cross-entropy loss obtained after "functional unlearning" of one of the clusters. If the attack confidence level does not increase when removing the cluster, the cluster is determined to be benign and the samples corresponding to the cluster are pruned.

One limitation of the authors' forensic traceback tool is that it can be "vulnerable to 'false flag' attack where an attacker carefully crafts a misclassification event that triggers [the] ... system to blame innocent data provider." Note that to be able to assign "blame to any parties involved" one would have to pair the authors' proposed traceback tool with "metadata or logs that track the provenance of training data." Moreover, the provenance data should be accurate and tamper-free.

While these related works consist of provenance-based and non-provenance-based data poisoning detection defenses, it is important to note that our work is not a proposed defense, rather we set the groundwork for establishing threshold detection by exploring what we characterize as training provenance for clues of data poisoning. Our exploratory approach of training provenance capture relies on metrics that are freely available in modern-day ML frameworks and the main focus is to go beyond metadata and/or data source trustworthiness, and capture the actual training progression in pursuit of clues for poisoning. We also note that, in line with our motivational scenario (Section 3.1), instead of replacing samples with new clean/poisonous samples, we rather

add clean/poisonous samples as a simulation of how a model owner would retrain models in real-life.

Chapter 5: Approach

This chapter first presents a high-level overview of training provenance-based exploration of clues of training data poisoning in Section 5.1. Subsequent sections (5.2 – 5.3) describe details of training provenance artifacts that we capture, technical details of the attack(s) used for our experiments, and how training provenance clues could be used to pinpoint data poisoning.

5.1 Training Provenance-Based Exploration Overview

The main objective of training provenance-based exploration is to see if training provenance, as defined in Section 2.3 can be used for establishing detection threshold against data poisoning attacks. Having the capability to determine whether or not the data used in the retraining of a model is poisonous, a model owner can then decide whether it is safe to deploy the current model instance. As motivated in Chapter 1, the key intuition behind our approach is the comparison of training provenance from previous training cycles to training provenance captured for the current training cycle to see if any observations can be made that would significantly distinguish one training cycle from the other. From these observations, we ask ourselves if there is sufficient evidence for establishing a detection threshold for detecting if new input samples are poisonous, or not. Towards this goal, our approach is shown in Figure 5.1 and is composed of the following four phases:

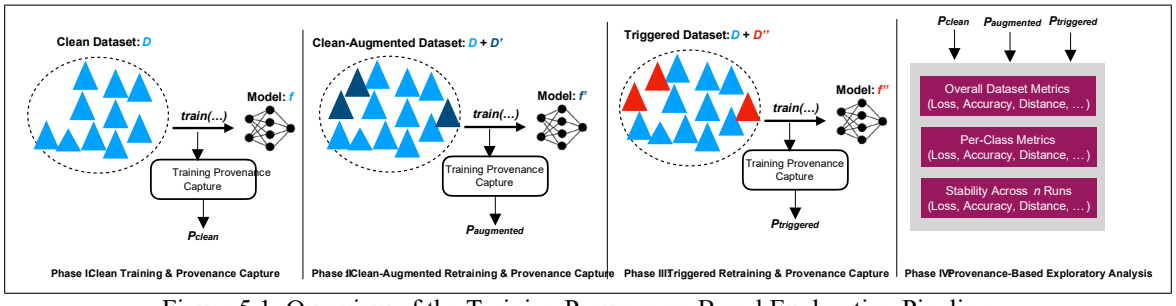


Figure 5.1: Overview of the Training Provenance-Based Exploration Pipeline.

- **Phase I: Clean Training and Provenance Capture**: Given an original (clean) training data D, we first train a model f on it. During training, we capture key performance metrics (e.g. loss, accuracy, etc.) for the overall dataset as well as per-class metrics (e.g. precision, recall, etc.) at every epoch for each label. Since we capture training artifacts at each iteration/epoch, the captured metrics carry the training provenance as to what happened during training and how training evolved when observed through the lens of multiple metrics. Let us collectively call the captured training provenance P_{clean}.
- **Phase II: Clean-Augmented Retraining and Provenance Capture**: We first generate additional clean samples D' through *data augmentation* to simulate a real-life scenario where a model owner would collect samples out in the real-world to retrain their ML model in an effort to improve its performance and generalization capability for unseen data samples. We then train our model f' on the original dataset with these augmented samples included in training set while maintaining the data distribution for each label, i.e., $D + D'$. During training, we capture the same key performance metrics and per-class metrics at every epoch as we did in Phase I. We call the captured provenance $P_{augmented}$.
- **Phase III: Triggered Retraining and Provenance Capture**: We first generate our triggered (poisonous) samples D'' to be added to our original dataset D through the simulation of a data poisoning attack as defined in Section 2.2. We then train our model f'' on $D + D''$. During training, we capture the same key performance metrics and per-class metrics at every epoch as we did in Phase I or II. Similarly, let the captured provenance be $P_{triggered}$.
- **Phase IV: Training Provenance-Based Exploratory Analysis**: Given, P_{clean}, $P_{augmented}$, and $P_{triggered}$, we perform our training provenance-based exploratory analysis via pairwise, per-epoch comparison of (P_{clean}, $P_{augmented}$) and (P_{clean}, $P_{triggered}$). Our analysis is aimed at finding clues (consistent patterns) that standout to serve as strong signals for making a call on training data poisoning. We do the analysis both at the overall dataset level and also zoom-in on each label so as to examine the effect of poisoning on individual labels. In addition to per-epoch training progression capture, we also compute the distance at each epoch by taking the difference in value for captured metric between (P_{clean}, $P_{augmented}$) and (P_{clean}, $P_{triggered}$). We compare these distances side by side to see how they change over time during training. With these distance measures for each of the captured metrics, we

observe if there are any anomalies, or strong indicators that would distinguish the performance of our ML model when retrained on clean input samples compared to triggered input samples. If there is sufficient evidence of this, we describe how one can establish threshold rules that can then be used for detecting if a dataset used for retraining ML model is poisonous, or not.

Given the nondeterministic nature of machine learning, we also take the Euclidean distance between (P_{clean}, $P_{augmented}$) and (P_{clean}, $P_{triggered}$) for the metric being captured. We calculate distance using the following formula:

$$\sqrt{\sum_{i}^{n}(v_1[i] - v_2[i])^2} \tag{5.1}$$

where $v_1[i]$ represents a value point out of all value points that make up one vector and $v_2[i]$ represents a value point out of all value points that make up the other vector. In our case, these value points represent one of the captured metrics across every epoch for the original dataset D, clean-augmented dataset $D+D'$ and triggered dataset $D + D''$. The distance measures summarize the relative differences between a given captured metric for the original dataset and that of the clean-augmented, or triggered dataset. Through comparison of the Euclidean distance captured for (P_{clean}, $P_{augmented}$) and (P_{clean}, $P_{triggered}$), we see if our observations are stable, and hold true for multiple runs.

In the rest of this chapter, we expand our discussion on the different components of our Training Provenance-Based Exploration Pipeline.

5.2 Captured Training Provenance

In this section, we discuss the metrics we consider as captured training provenance. Furthermore, we provide a technical description of the attack used for generating triggered samples for the given dataset that we have trained our ML model on.

5.2.1 Metrics Considered as Training Provenance

As we have defined in Section 2.3, training provenance is the history of captured training metrics, which details how these metrics change over the course of time as new input samples are introduced in the retraining process of one's ML model. In our case, we consider the following

metrics captured for the overall dataset and at the per-class label trained on our ML model. While we only take into account the metrics described below, one can certainly expand and modify our pipeline to incorporate other metrics (e.g., False Positive Rate (FPR), False Negative Rate (FNR), True Positive Rate (TPR), F1-Score).

Overall Dataset Metrics. For the overall dataset, the metrics that we focused on are Categorical Cross-Entropy (CE) Loss and Categorical Accuracy. Categorical Cross-Entropy Loss, also referred to as SoftMax logarithmic loss, is the penalty calculated during training that measures how far off the predicted probability of an input sample is from its actual value. This metric is used when adjusting the ML model's weights with the goal being to minimize this as training progresses and optimize the ML model. Categorical Cross-Entropy Loss can be calculated as follows:

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \tag{5.2}$$

$$CE = \sum_i^C t_i log(f(s)_i) \tag{5.3}$$

where the first equation represents the SoftMax activation function, which converts a vector of raw outputs, represented as *s*, from our ML model to a vector of probability scores. In this equation, the value of *e* is approximately 2.718. The second equation represents cross-entropy loss function, which sums up the product of ground-truth vector, represented as t_i, and *log* of vector probability scores. In both equations, *C* represents the number of classes in the given dataset. Categorical Accuracy is the percentage of predicted values from our ML model that match with actual values, ground-truth labels. It can be calculated as follows:

$$\frac{T}{n} \tag{5.4}$$

where *T* represents the total number of input samples for which the predicted value and actual value are the same, and *n* represents the total number of input samples in the given dataset.

Per-Class Metrics. For per-class labels, we focus on Precision, Recall, and Support. Precision is the proportion of predicted positives that are truly positive. It can be calculated as follows:

$$Precision = \frac{TP}{TP + FP} \tag{5.5}$$

Recall is the proportion of actual positives that are correctly classified, and can be calculated as follows:

$$Recall = \frac{TP}{TP + FN} \tag{5.6}$$

In both equations, true positive (*TP*) is the sum of input samples that have been predicted correctly for given class. False positive (*FP*) is the sum of input samples falsely predicted as a given class when they actually belong to another class. False negative (*FN*), is the sum of input samples that belong to the given class, but they are predicted as belonging to a different class. It is important to note that recall in multiclassification is equal to what we define as per-class accuracy. Finally, we capture *Support* as the number of input samples actually belonging to given class. This particular metric is helpful for tracking how the distribution changes when adding new input samples for each class.

It is important to note that for triggered samples, the actual label taken into consideration for computing these metrics is not the ground-truth label, but the train label, or target label for which the adversary wants ML model to learn for input samples where triggers are present. In the following section, we will discuss how these triggered samples are generated via simulated data poisoning attack.

5.2.2 Technical Description of Data Poisoning Attack

To realize Phase III of Figure 5.1, any data poisoning attack among the ones we discussed earlier is applicable. To streamline the discussion here, we use a poison backdoor generation framework as an illustration. In particular, to generate our datasets used in our set of experiments described in Chapter 6, we utilize scripts from TrojAI [11], an open-source software framework that consists of a "set of Python modules that enable one to quickly and reproducibly generate trojaned deep learning classification and reinforcement learning models". The framework is "extensible to enable generation of datasets ... with a wide variety of triggers." The framework

consists of two submodules: datagen and modelgen. For the purpose of generating our triggered samples to be used in our experiments, we only utilize the *datagen submodule*. The framework in its entirety is described in [11]. Here, we specifically focus on providing a technical description of the data poisoning attacks for which the scripts we use are based on.

In the first set of experiments we conduct, we train our ML model on the MNIST dataset. The TrojAI submodule, datagen, contains "mnist_badnets.py" script that replicates the data poisoning attack on MNIST as described in the BadNets paper [8] by embedding a "*pattern* backdoor, a pattern of bright pixels, ... in the bottom right corner of the image". The BadNets attack first verifies that this corner "is always dark in non-backdoored images" to guarantee that no false positives are produced. The *pattern* backdoor used in this attack is a reverse lambda as seen in Figure 5.2. While the BadNets paper, implements two types of attacks using this backdoor trigger: the attack carried out in our experiment is defined as an "*All-to-all attack*" in [8] where the label of a digit *i* is incremented by 1 for input samples of digit *i* where this trigger is present. This label can be set per the adversary's goals. Our ML model is trained on this label instead of the actual ground-truth label; thereby, mislearning the relationship between it and its corresponding input sample. For input samples where the trigger is present, the ML model would mis-classify to the specified label set by the adversary whereas the model would behave normally for input samples where the trigger is not present.

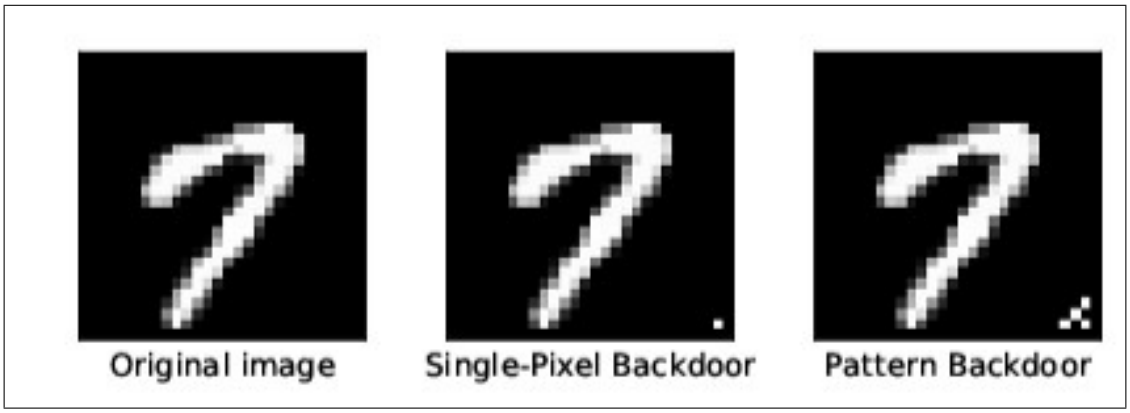


Figure 5.2: Original MNIST, Single-Pixel backdoor, and Pattern Backdoor [8].

In the second set of experiments, we train our ML model on CIFAR-10. The TrojAI submodule, modelgen, contains "gen_and_train_cifar10.py" script that "generates CIFAR-10 dataset with one class triggered using Gotham Instagram filter, and trains a model on various poisoning percentages." [11] The use of Instagram filters as triggers was first presented in the ABS

paper [15]. This type of attack is referred to as a *feature space attack* where "the pixel space mutation (to trigger mis-classification) is no longer fixed, but rather independent." Figure 5.3 shows how the Gotham Instagram filter can be used as a backdoor trigger by transforming the original RGB colored image "into black & white, with high contrast and bluish undertones ... the pixel level mutations induced by these filters vary from one image to another" [15].

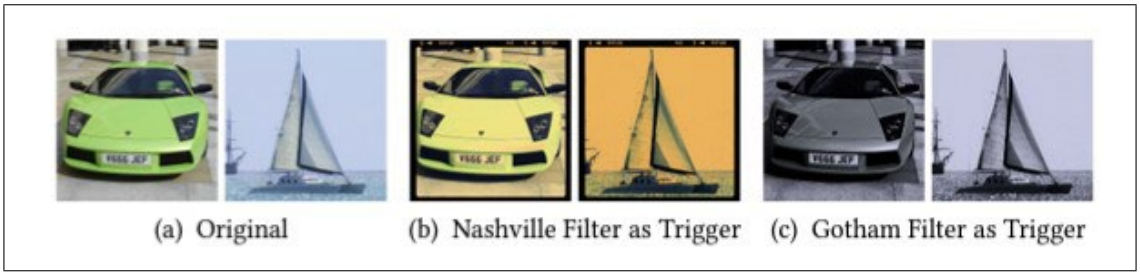


Figure 5.3: Data Poisoning Attack: Instragram Filter Triggers [15].

Similar to the BadNets "*Single target attack*", this attack "labels backdoored versions of digit *i* as digit *j*" where digit corresponds to each category class in CIFAR-10 [8]. This attack differs from the BadNets attack; however, as Liu et al. in [15] define the BadNets attack as a *pixel space attack* that embeds a *patch-based trigger* "... on the original input image and the patch covers part of the image." In our second set of experiments, this type of attack changes label *i* (actual ground-truth label that the adversary wants misclassified) to label *j* (train label set by the adversary) by shifting the label up by 1 for input samples where the Instragram Gotham Filter is applied. Our ML model is trained on this single target label instead of the actual ground-truth label; thereby, mislearning the relationship between it and its corresponding input sample. For input samples where the trigger is present, the ML model would misclassify to the specified target label set by the adversary whereas the model would behave normally for input samples where the trigger is not present.

Both TrojAI scripts generate a clean dataset and produce triggered samples by randomly selecting input samples from the clean dataset based on a given percentages for which one wants to poison the training set and test set. Along with the clean and triggered images, the scripts also produce three CSV files: (*i*) one that describes the training set, (*ii*) one that describes test data with no triggers, and (*iii*) one that describes the test data with triggers. Each CSV file is structured in the same manner as shown in Table 5.1, where the "file" column contains the file location of the input sample, the "true label" column contains the input sample's corresponding true label (actual

ground-truth label), the “train label” column contains the target label set by the adversary, and the “triggered” column contains either “True” or ”False” based on whether, or not the backdoor trigger is present in the input sample. We discuss the details of percentages selected for producing clean-augmented samples and triggered samples in our Experimental Setup (Section 6.1) where we describe our datasets and how we partition data for clean and triggered experiments.

file	true label	train label	Triggered
f1	1	1	False
f2	1	2	True
...	...	...	...

Table 5.1: CSV File Structure for Training and Test Data [11].

5.3 How Threshold can be Decided and Used

As described in our approach overview (Section 5.1), we calculate the distance at every epoch between the clean original dataset metrics captured as training provenance versus the metrics captured as training provenance for one of the other two datasets: the augmented dataset, or the poisoned dataset. The distance at epoch measurements for a given metric are then plotted on the same bar chart graph for a given percentage against the number of epochs so that one can visually compare how this calculated distance changes over the course of training for both the original vs. augmented and the original vs. triggered. Ideally, the smaller the distance for the given metric, the closer the metric results of the modified dataset (augmented or triggered) are to that of the original dataset without added input samples.

If there is sufficient evidence that distinguishes the two calculated distances from each other, we can then determine the expected threshold range for the overall dataset. For example, we can draw a line across the top of the bars plotted for the original vs. augmented distance at epoch. We can then say that for any calculated distance measurements that fall outside the threshold range that we have set, there are indicators for data poisoning. This is to say that retraining of triggered samples should produce higher distance measures compared to the retraining of clean-augmented samples. If we determine that we can establish threshold detection using overall dataset metrics as training provenance, we can then proceed to look at the per-class level to see if we can attribute the ML model’s incorrect predictions to portion (%) of poisoned samples injected during the retraining process.

To ensure that we are getting consistent results as we conduct our experiments for multiple runs, we repeat the entire process described above for the calculated Euclidean distance of a given metric plotted against number of runs. It is important to note, that deciding the threshold range, if sufficient evidence exist to support this, is dependent on the dataset used, as well as the percentage of samples injected during the retraining process, and the set of attacks simulated by the ML model owner. We will examine the prospect of determining a poisoning detection threshold based on the training provenance metrics discussed.

Chapter 6: Evaluation and Results

This chapter presents the dataset, experimental setup, and evaluation results of our exploratory analysis of training provenance for clues of data poisoning. The evaluation is guided by the following research questions:

- **RQ1**: How effective are the metrics used (collectively and individually) to establish training provenance for clean and poisoned training data?
- **RQ2**: How effective is our approach in establishing a threshold to detect poisoning?
- **RQ3**: When poisoning happens, can we attribute model's mistakes (e.g., incorrect predictions) to the portion (%) of poisonous data injected to the training set?

6.1 Datasets and Experimental Setup

We evaluate our approach on two benchmark image classification datasets: MNIST [13] and CIFAR-10 [12].

6.1.1 Datasets

MNIST. The MNIST [13] dataset consists of 28 × 28 grayscale images of handwritten digits (0 to 9). The dataset distribution is shown in Table 6.1 with 60K train set and 10K test set.

Digit	Train	Test
0	5923	980
1	6742	1135
2	5958	1032
3	6131	1010
4	5842	982
5	5421	892
6	5918	958
7	6265	1028
8	5851	974
9	5949	1009
Total	60K	10K

Table 6.1: Original MNIST Dataset Distribution.

To evaluate our approach against the simulated "All-to-All" BadNets attack (as described in Section 5.2), for the first set of experiments, we train our model on three variations of the MNIST dataset: the original MNIST dataset, the original MNIST dataset plus clean-augmented MNIST samples, and the original MNIST dataset plus triggered MNIST samples. Given the original dataset, we utilize the TrojAI datagen script (from [11]) to generate triggered samples. For generating new clean input samples through data augmentation [22], we perform height and width shift operations by a factor of 0.1 on the original clean dataset.

The number of triggered samples that are added to the original dataset to make up the triggered dataset is based on percentages of samples that fall under the ground-truth label. The percentages we selected for the purpose of evaluation are 5%, 10%, 15%, and 20%. This number of samples is then added under the target label in the triggered dataset. For example, we take 5% of samples with Label 0 in the original dataset for both the training set and test set, i.e., 296 training samples and 49 test samples, respectively. We then add these samples under each respective set (training set, test set) to Label 1, the target label which our model is trained on for these images where a trigger is present. We repeat this for each label for each of the percentages we have chosen. To keep the data distribution nearly identical to the triggered dataset, we add the same number of samples, but of clean-augmented images of the target label under the target label (e.g., images of "1" under Label "1"). The data distribution for both of these MNIST datasets, clean-augmented and triggered, is shown in Table 6.2.

Digit	Train (5%)	Test (5%)	Train (10%)	Test (10%)	Train (15%)	Test (15%)	Train (20%)	Test (20%)
0	6220	1030	6518	1081	6815	1131	7113	1182
1	7038	1184	7334	1233	7630	1282	7927	1331
2	6295	1089	6632	1146	6969	1202	7306	1259
3	6429	1062	6727	1113	7025	1165	7323	1216
4	6149	1033	6455	1083	6762	1134	7068	1184
5	5713	941	6005	990	6297	1039	6589	1088
6	6189	1003	6460	1047	6731	1092	7002	1136
7	6561	1076	6857	1124	7153	1172	7449	1220
8	6164	1025	6478	1077	6791	1128	7104	1180
9	6242	1057	6534	1106	6827	1155	7119	1204
Total	63K	10.5K	66K	11K	69K	11.5K	72K	12K

Table 6.2: Clean-augmented and Triggered MNIST Dataset Distribution.

CIFAR-10. The CIFAR-10 [12] dataset consists of 32 × 32 colored images that fall under the following 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset is equally distributed consisting of 50K images for training and 10K for testing/validation.

Following the same scheme that we described for MNIST, we utilize the TrojAI modelgen script, with slight modifications of our own, for generating triggered samples. For generating new clean input samples through data augmentation, we use the same augmentation parameters that we used for MNIST. To progressively evaluate our approach, we also use the 5%, 10%, 15%, and 20% incremental percentages of original CIFAR-10 data. To generate the triggered dataset, Label "deer" is embedded with backdoor trigger so that it is misclassified as Label "dog." The same data generation and distribution preservation approach used for MNIST applies for CIFAR-10 as well. The data distribution for the target label (Label 5 / dog) is shown in Table 6.3.

Digit	Train (5%)	Test (5%)	Train (10%)	Test (10%)	Train (15%)	Test (15%)	Train (20%)	Test (20%)
5	5250	1050	5500	1100	5750	1150	6000	1200
Total	50250	10050	50500	10100	50750	10150	51K	10200

Table 6.3: Clean-Augmented & Triggered CIFAR-10 Label 5 ("Dog") Data Distribution.

6.1.2 Experimental Setup

Model Architecture and Hyper-parameters. The ML algorithm that we use for our experiments is a Convolutional Neural Network (CNN). The architecture and hyper parameters for our model on which MNIST is trained on is shown in Table 6.4, the same model architecture described in the BadNets paper [8] with the addition of two Dropout layers. The Dropout layers help to prevent our model from overfitting. The model architecture for which CIFAR-10 is trained on is shown in Table 6.5. For both models, we use the *Adam* optimizer with learning rate set to 0.001. The batch size that we used for training MNIST is 32. For CIFAR-10, we used a batch size of 250. Both MNIST and CIFAR-10 datasets are trained for a total of 10 epochs. To observe the stability of training provenance metrics we capture across multiple runs, we train each model for a total of 30 runs.

Layer	Input	Filter	Stride	Rate	Activation
Conv2D	28x28x1	16x1x5x5	1	/	ReLU
Avg Pooling2D	24x24x16	2x2	2	/	/
Conv2D	12x12x16	32x16x5x5	1	/	ReLU
Dropout	8x8x32	/	/	0.25	/
Avg Pooling2D	8x8x32	2x2	2	/	/
Flatten	4x4x32	/	/	/	/
Dense	512	/	/	/	ReLU
Dropout	512	/	/	0.5	/
Dense	512	512x10	/	/	SoftMax

Table 6.4: Architecture MNIST Model with added Dropout Layers.

Layer	Input	Filter	Stride	Rate	Activation
Conv2D	32x32x3	64x3x4x4	1	/	ReLU
Conv2D	29x29x64	64x64x4x4	1	/	ReLU
Max Pooling2D	26x26x64	2x2	/	/	/
Dropout	13x13x64	/	/	0.4	/
Conv2D	13x13x64	128x64x4x4	1	/	ReLU
Conv2D	10x10x128	128x128x4x4	1	/	ReLU
Max Pooling2D	7x7x128	2x2	/	/	/
Dropout	3x3x128	/	/	0.4	/
Flatten	3x3x128	/	/	/	/
Dense	1152	/	/	/	ReLU
Dense	1024	/	/	/	ReLU
Dense	1024	1024x10	/	/	SoftMax

Table 6.5: Architecture of CIFAR-10 Model.

6.2 Results on MNIST Experiments

To answer research questions **RQ1** – **RQ3**, we examine the results we obtain for our MNIST Experiments with respect to the "All-to-All" BadNets attack described in Section 5.2. In particular,

we examine the use of captured overall metrics and per-class metrics as training provenance and determine if there are clues that may lead to establishing a poisoning detection threshold based on the comparison of training artifacts on clean-augmented and triggered datasets. We further examine if our observations hold true for multiple runs of our MNIST Experiments.

6.2.1 Effectiveness of Overall Dataset Metrics as Training Provenance

As described in Chapter 5, we capture Categorical Cross-Entropy (CE) Loss and Sparse Categorical Accuracy during training of our model as training provenance. These common metrics describe how well our model performs over the whole dataset throughout the training process. With the goal of establishing threshold detection based on calculated distance at epoch between the original MNIST dataset and one of the two MNIST dataset variants: clean-augmented and triggered MNIST, we train our model on these three variations of MNIST dataset. Our original MNIST dataset serves as the baseline for the model's performance, meaning that captured metrics for the other two MNIST dataset variants should produce similar measurement values for one to consider redeploying the retrained model on clean-augmented or triggered samples.

From the adversary's perspective, data poisoning attacks on training set tend to achieve the same overall accuracy so as to evade detection. However, we as defenders anticipate seeing some anomalies or strong signals that data poisoning has occurred through comparison of how a model behaves when adding clean samples to the original dataset versus adding triggered samples to the original dataset. One can attempt to observe such differences by comparing Figure 6.1 and Figure 6.2. During Epochs 0 and 1, triggered MNIST datasets display higher Cross-Entropy Loss values and lower Categorical Accuracy values when compared to the original MNIST and clean-augmented datasets. In fact, clean-augmented datasets perform closer to expected values obtained for the original MNIST dataset. As training progresses, however, the differences in captured metric values for these MNIST datasets begin to shrink. While the training history plots in Figure 6.1 and Figure 6.2 seem to offer a high level narrative of how training unfolds, for subtle poisoning attacks that are unlikely to penalize model accuracy, one has to zoom-in on the epoch-by-epoch comparative analysis of training progression for clean-augmented and triggered with respect to original training data.

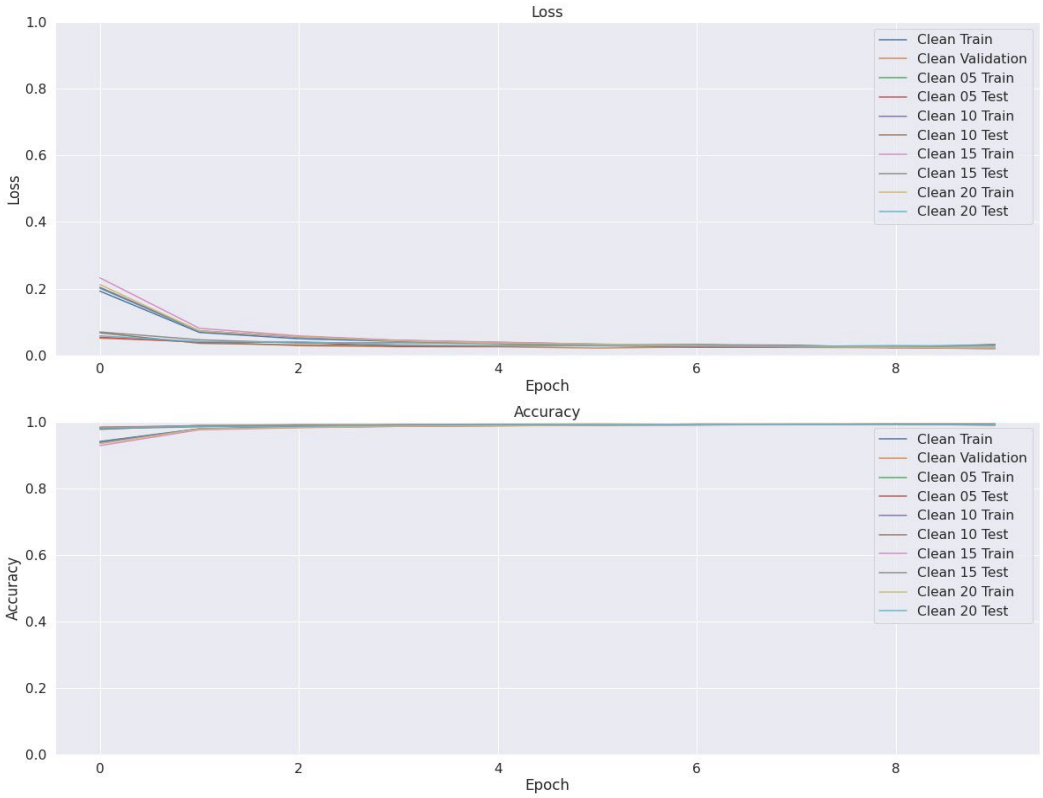


Figure 6.1: History Plot for Original MNIST vs. Clean-Augmented MNIST.

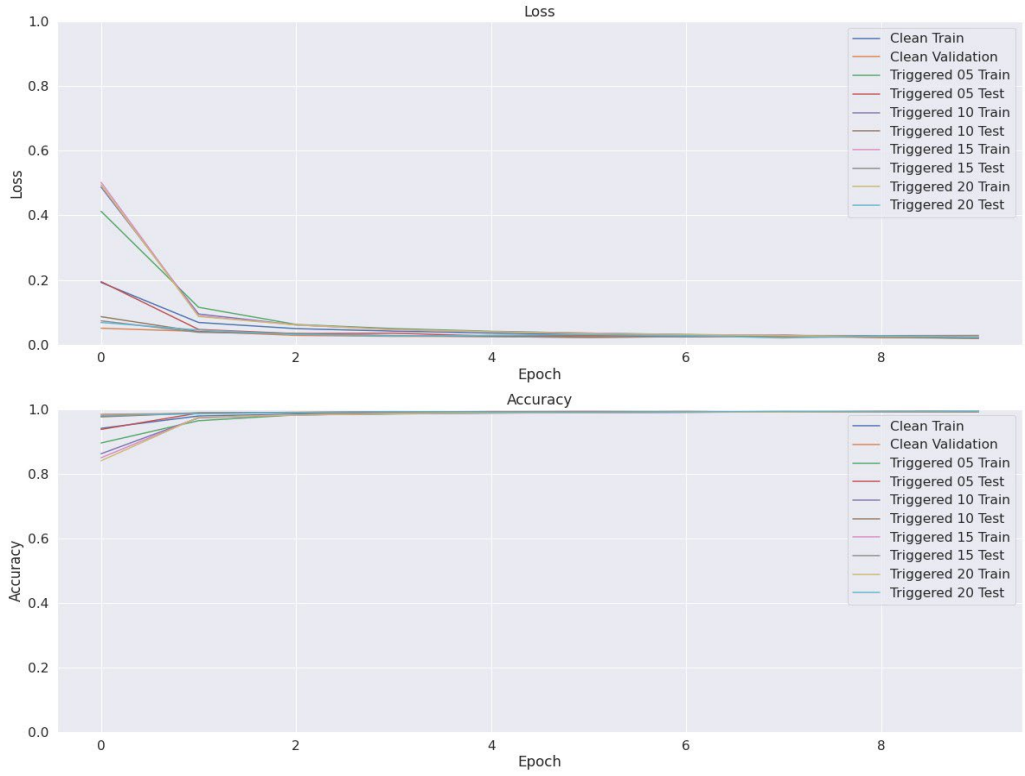
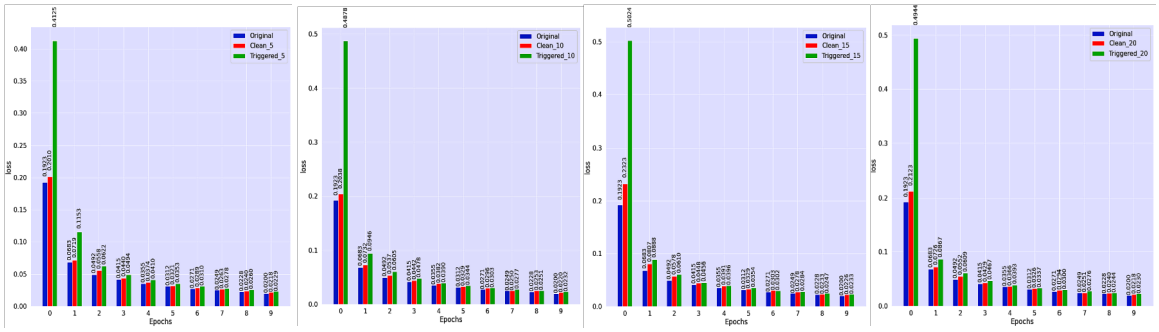


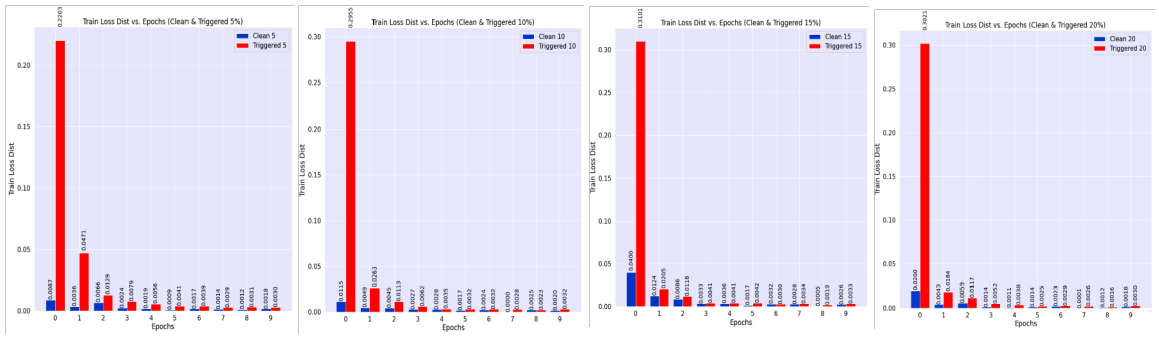
Figure 6.2: History Plot for Original MNIST vs. Triggered MNIST.



(a) 5% (b) 10% (c) 15% (d) 20%

Figure 6.3: Training Trajectory: Epoch vs. Loss ("Original" = MNIST-Original, "Clean " = MNIST-Clean-Augmented, "Triggered " = MNIST-Triggered).

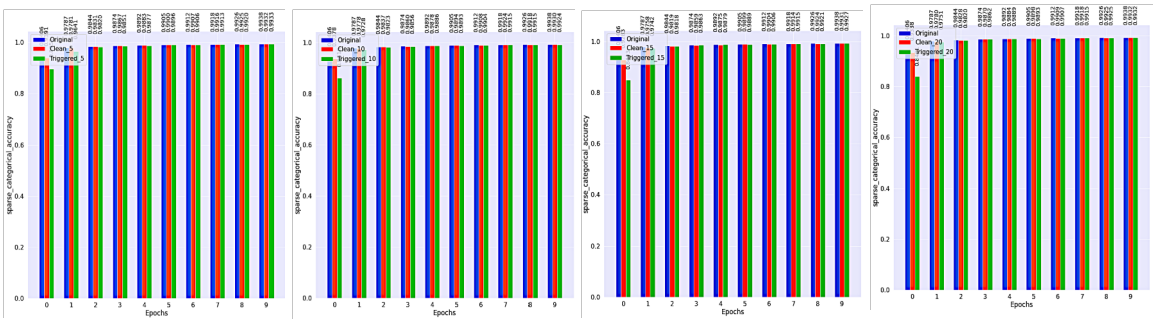
To help better visualize and compare the Cross-Entropy Loss values obtained for original MNIST, clean-augmented MNIST, and triggered MNIST datasets, the captured training trajectory of Cross-Entropy Loss is shown in Figure 6.3 for 5%, 10%, 15%, and 20% of clean-augmented and triggered samples. As one can observe, Cross-Entropy Loss captured when the model is trained on triggered MNIST datasets is higher throughout the entire training process when compared to the original MNIST and clean-augmented MNIST datasets. This is especially evident during the first epoch. As training progresses from the first epoch to the second, the captured Cross-Entropy Loss drastically drops. From the third epoch to the last, it continues to gradually descend until it closely resembles both the original MNIST and clean-augmented MNIST datasets. In comparison, the captured Cross-Entropy Loss values for clean-augmented MNIST datasets closely resemble that of the original MNIST dataset from beginning of the training cycle to the end. For clean-augmented MNIST and triggered MNIST datasets, one can also observe that as more input samples are added onto the original MNIST dataset, the Cross- Entropy Loss captured at Epoch 0 slightly increases whereas at the later epochs, it appears to decrease slightly.



(a) 5% (b) 10% (c) 15% (d) 20%

Figure 6.4: Loss-Distance (Clean, MNIST-Clean-Augmented) vs. Loss-Distance (Clean, MNIST-Triggered).

From the captured training trajectory of Cross-Entropy Loss, we can take a closer look at the differences of how the model behaves when trained on a specific MNIST dataset by calculating the distance between Cross-Entropy Loss value captured for original MNIST and clean-augmented MNIST datasets at each epoch as well as the distance at each epoch between original MNIST and triggered MNIST datasets. These calculated distances are displayed side by side in Figure 6.4 for different percentages of clean-augmented and triggered samples added to the original dataset. As one can observe, the Cross-Entropy Loss distance between original MNIST and triggered MNIST at Epoch 0 is exponentially higher, decreases drastically from Epoch 0 to Epoch 1, and then maintains a gradual descent until the final epoch. As for the Cross-Entropy Loss distance between original MNIST and clean-augmented MNIST, the distance is relatively small throughout the entire training process. For both clean-augmented and triggered MNIST, the calculated Cross-Entropy Loss Distance increases slightly at Epoch 0 when more clean-augmented, or triggered input samples are introduced during the retraining process. For triggered MNIST, the distance is lower at Epoch 1 when more triggered input samples are introduced (10%-20%). In general, the differences observed in distance for clean-augmented and triggered MNIST datasets becomes less apparent during the last epochs making it difficult to determine a universal distance threshold that can be used for detection.

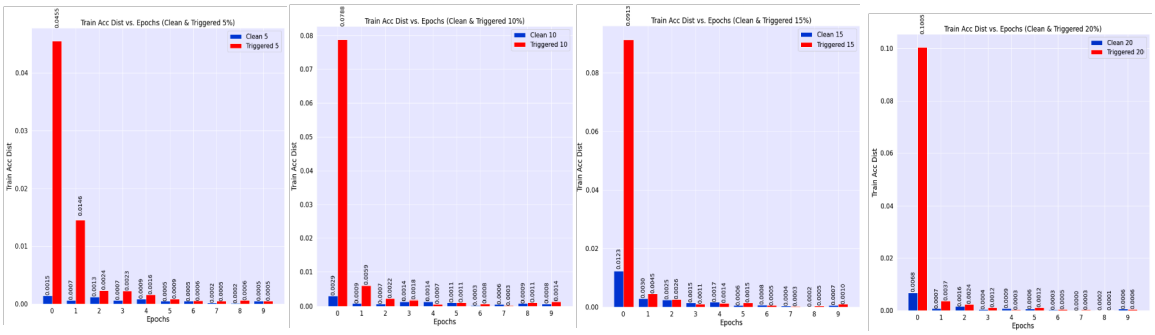


(a) 5% (b) 10% (c) 15% (d) 20%

Figure 6.5: Training Trajectory: Epoch vs. Train Accuracy ("Original" = MNIST Original, "Clean _" = MNIST-Clean-Augmented, "Triggered _" = MNIST-Triggered).

To help better visualize and compare the Accuracy values obtained for original MNIST, clean-augmented MNIST, and triggered MNIST datasets, the captured training trajectory of Accuracy is shown in Figure 6.5 for different percentages of clean-augmented and triggered samples added to the original dataset. As one can observe, Accuracy captured when the model is trained on triggered MNIST datasets is lower by few percent at Epoch 0, after which it steadily climbs up to closely resemble the values captured during the model training on the original and clean-augmented

MNIST datasets. In comparison, the captured Accuracy values for clean-augmented MNIST datasets closely resemble that of the original MNIST dataset from beginning of the training cycle to the end. For both clean-augmented MNIST and triggered MNIST datasets, one can also observe that when more input samples are added onto the original MNIST dataset, the Accuracy captured at the first epoch slightly decreases.



(a) 5% (b) 10% (c) 15% (d) 20%

Figure 6.6: Train Accuracy-Distance (Clean, MNIST-Clean-Augmented) vs. Train Accuracy-Distance (Clean, MNIST-Triggered).

From the captured training trajectory of Accuracy, we can take a closer look at the differences of how the model behaves when trained on a specific MNIST dataset by calculating the distance between Accuracy value captured for original MNIST and clean-augmented MNIST datasets at each epoch as well as the distance at each epoch between original MNIST and triggered MNIST datasets. These calculated distances are displayed side by side in Figure 6.6 for different percentages of clean-augmented and triggered samples added to the original dataset. As one can observe, the Accuracy distance between original MNIST and triggered MNIST at Epoch 0 is exponentially higher, decreases drastically from Epoch 0 to Epoch 1, and then maintains a gradual descent until the final epoch. For Figure 6.6 (a), it is important to note that captured Accuracy distance between original and triggered MNIST, does drastically decrease once more from Epoch 1 to Epoch 2. As for the Accuracy distance between original MNIST and clean-augmented MNIST, the distance is relatively small throughout the entire training process. For both clean-augmented and triggered MNIST, the calculated Accuracy Distance increases slightly at Epoch 0 when more clean-augmented, or triggered input samples are introduced during the retraining process. Similar to Cross-Entropy Loss distance, the differences observed in Accuracy distance for clean-augmented and triggered MNIST datasets becomes less apparent during the last epochs making it difficult to determine a universal distance threshold that can be used for detection.

Given that these calculated distances at epoch are taken at absolute value, it is important that these distance plots be observed alongside the training trajectory plots in order to tell whether the captured Cross-Entropy Loss and captured Accuracy values have improved when retraining the model on new input samples. In the case of MNIST, both the clean-augmented datasets and triggered MNIST dataset do not improve the overall model's performance when looking at captured Cross-Entropy Loss and captured Accuracy; however, we expect that for clean-augmented datasets this is due to the data augmentation operations used to produce such samples. Given clean samples that are closer in representation, we expect that our model would perform better with the addition of clean input samples to our training set.

Take-away: With respect to **RQ1** and **RQ2**, based on our observations, the metrics (Cross-Entropy Loss and Accuracy) used, individually/collectively, show consistently larger distance between triggered and clean when compared with the distance between clean-augmented and clean. While the magnitude of the distance values shrinks in the later epochs, the consistency is a promising signal to establish a per-epoch poisoning detection threshold. We note, however, that in order to build a reliable poisoning detection approach, this promising pattern for detection threshold needs to be stable across multiple runs of our training (which we will examine in Section 6.2.3).

6.2.2 Effectiveness of Per-Class Metrics as Training Provenance

Taking into consideration the key observations and takeaways from our evaluation of overall dataset metrics, Cross-Entropy Loss and Accuracy, as training provenance, we look at per-class metrics for further insight. We specifically look into the first two epochs to see if these metrics provide strong indicators for data poisoning and see if we can attribute the model's mistakes to the percentage of poisonous data injected into our MNIST training set. As mentioned in Chapter 5, the per-class metrics that we capture as training provenance are: Precision, Recall, and Support. While the first two metrics tell us how our model performs against each label, the Support metric allows us to keep track of the data distribution as new input samples are added during the retraining process.

When evaluating the use of per-class metrics as training provenance, we must consider how each label within our MNIST dataset is affected. As described in Section 5.2, the attack simulated to generate our MNIST trigger samples is an "All-to-All" attack. Each label in this case contains a

percentage of trigger samples that are misclassified into the label above it (e.g., Label 0 samples with trigger present are misclassified into Label 1 - triggered images have their numerical label shift by 1). Referring back to Section 6.1, the number of input samples that are added to each target label for triggered MNIST dataset is based on percentage of samples that fall under the ground-truth label. To keep the data distribution the same, the same number of clean-augmented samples are added to each label during retraining.

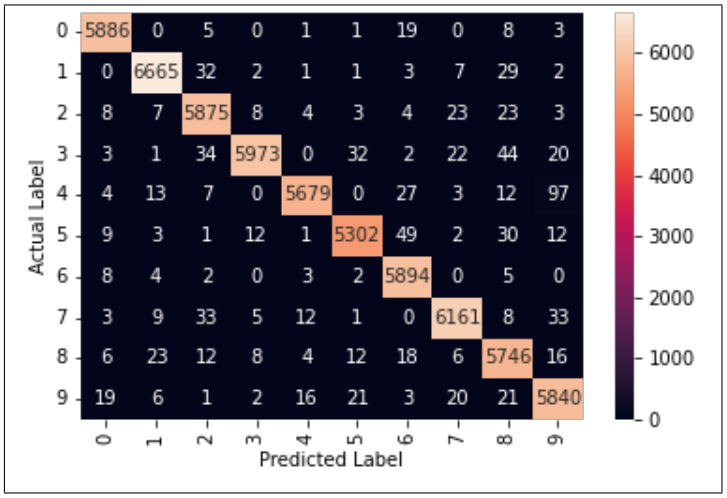
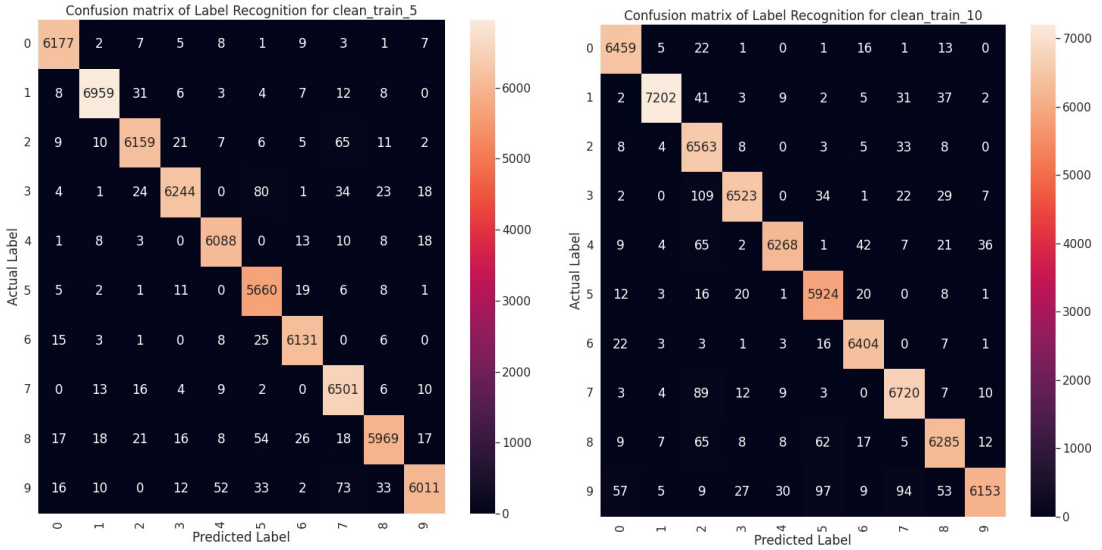


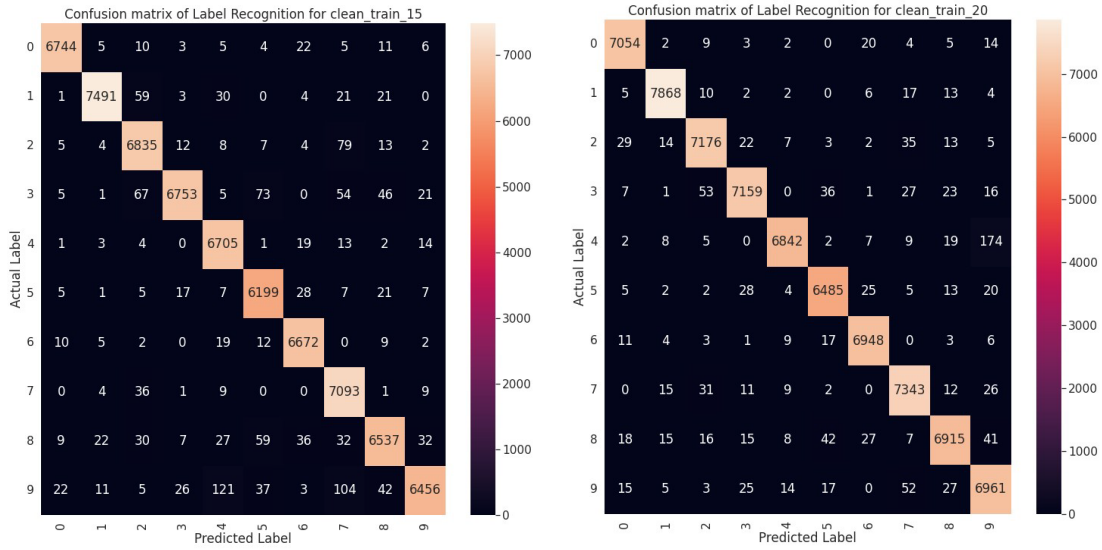
Figure 6.7: Confusion Matrix for Original MNIST at Epoch 0.

To see how the model is performing in terms of correctly classifying each sample with its corresponding label, we generated confusion matrices at each epoch for both original MNIST dataset, the clean-augmented MNIST dataset, and the triggered MNIST dataset. As mentioned before, our focus is on Epoch 0 and 1 as these early epochs showed distinctions when evaluating overall dataset metrics as training provenance. Figure 6.7, Figures 6.8 – 6.9, and Figures 6.10 – 6.11 show the confusion matrices generated at Epoch 0 for original MNIST, clean-augmented MNIST, and triggered MNIST datasets, respectively.



(a) 5% (b) 10%

Figure 6.8: Confusion Matrix for Clean-Augmented MNIST (5% & 10%) at Epoch 0.

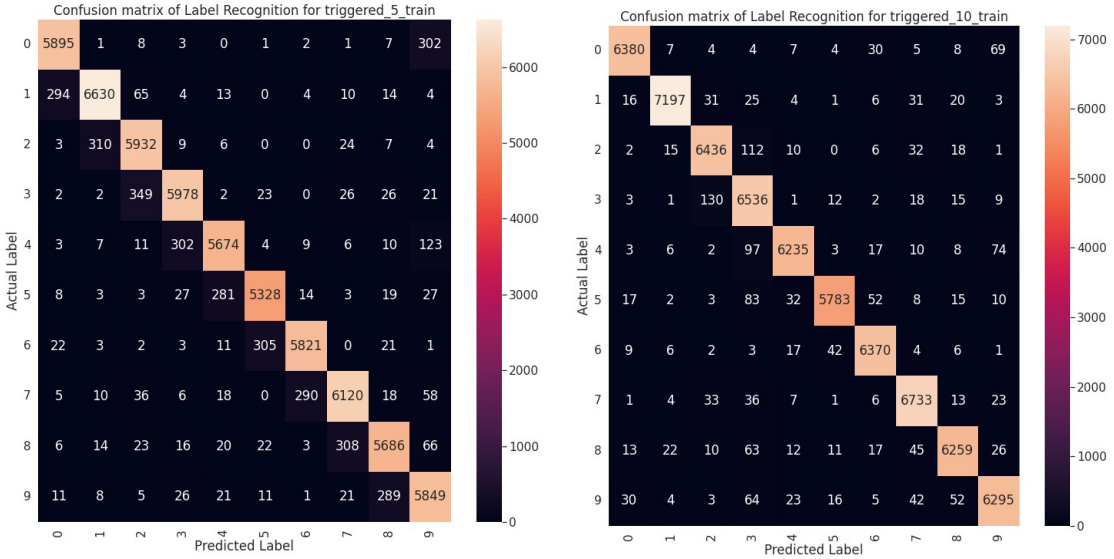


(a) 15% (b) 20%

Figure 6.9: Confusion Matrix for Clean-Augmented MNIST (15% & 20%) at Epoch 0.

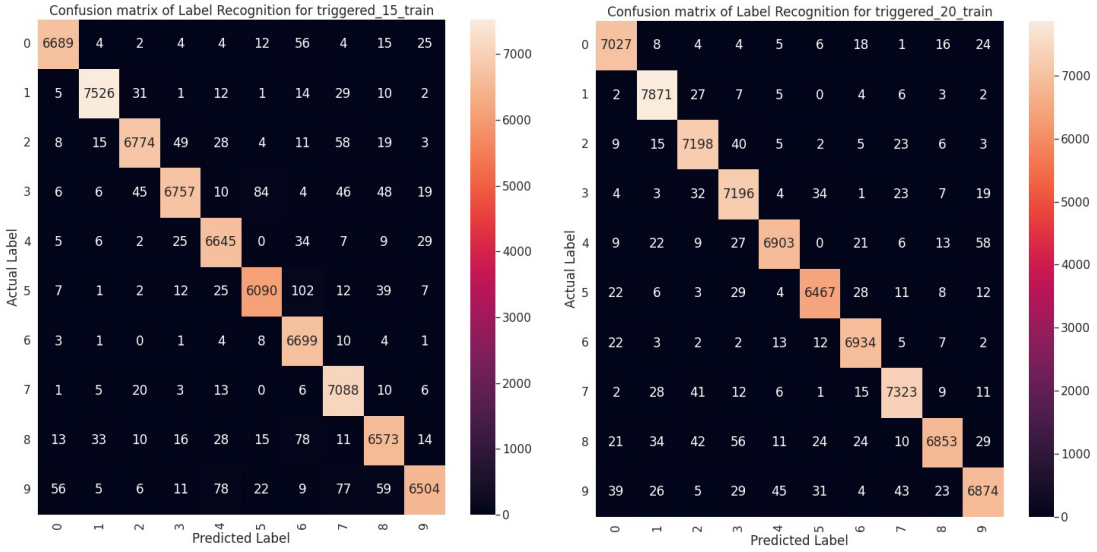
We base our initial observations on the comparison of the clean-augmented MNIST generated confusion matrix and the triggered MNIST generated confusion matrix at Epoch 0. One can see an extremely high number of incorrect predictions, approximately 300 samples incorrectly classified, under predicted label for each actual label when looking at confusion matrix of the triggered MNIST dataset 5% (Figure 6.10a) in comparison with confusion matrix of the clean-augmented MNIST dataset 5% (Figure 6.8a). Given that the triggered samples are trained on the target label and not the ground-truth label and we know the attack shifts the label by 1, these incorrect predictions are mostly likely the triggered samples being classified as their ground-truth label

instead of their target label. In other words, the trigger does not seem to be active during Epoch 0 as the input samples are not being misclassified to the target label. In this particular case, the incorrect predictions can be attributed to the % of poisonous data injected to the training set as approximately 300 incorrect predictions for 10 labels adds up to 3,000 samples which is 5% of 60K, the number of samples in the original MNIST training set. The calculated per-class metrics from these two confusion matrices also shows that the model is able to precisely and accurately predict the label of the original MNIST plus clean-augmented samples at Epoch 0 better than it is able to do for original MNIST plus triggered MNIST samples.



(a) 5% (b) 10%

Figure 6.10: Confusion Matrix for Triggered MNIST (5% & 10%) at Epoch 0.



(a) 15% (b) 20%

Figure 6.11: Confusion Matrix for Triggered MNIST (15% & 20%) at Epoch 0.

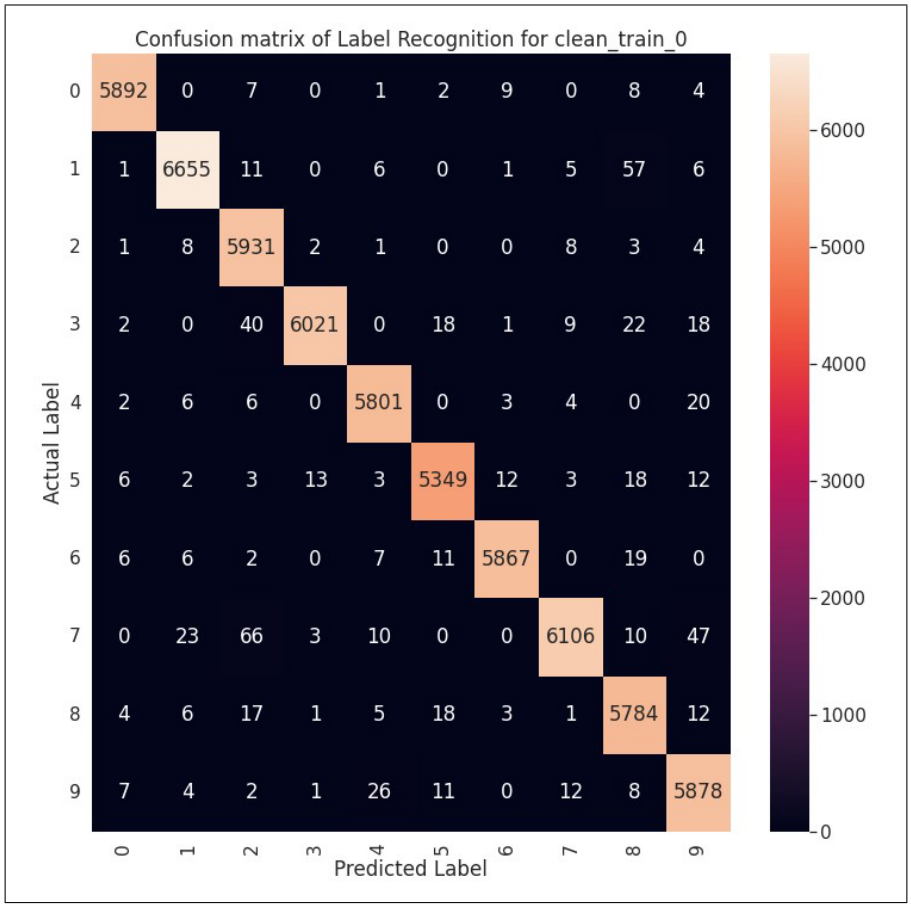
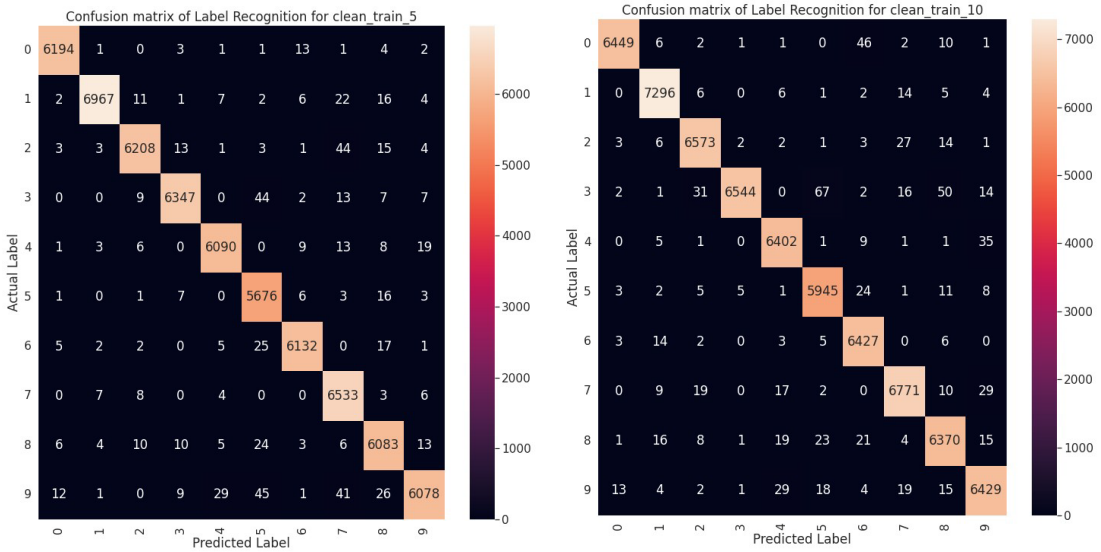


Figure 6.12: Confusion Matrix for Original MNIST at Epoch 1.

However, this observation does not hold when comparing confusion matrices of clean-augmented and triggered MNIST datasets with higher percentage (10%-20%) of input samples injected into the training set. There is still a high number of incorrect predictions, but these incorrect predictions seem to be distributed across predicted labels instead of falling under one predicted label. The calculated per-class metrics further support this as distinctions between the two datasets are less evident. When comparing values for Precision and Recall captured for one dataset side by side with the other dataset, the model may be able to predict more precisely, or accurately depending on the label for the given datasets. For instance, a label under clean-augmented MNIST dataset may have higher precision, or recall over the triggered MNIST dataset, or vice versa.

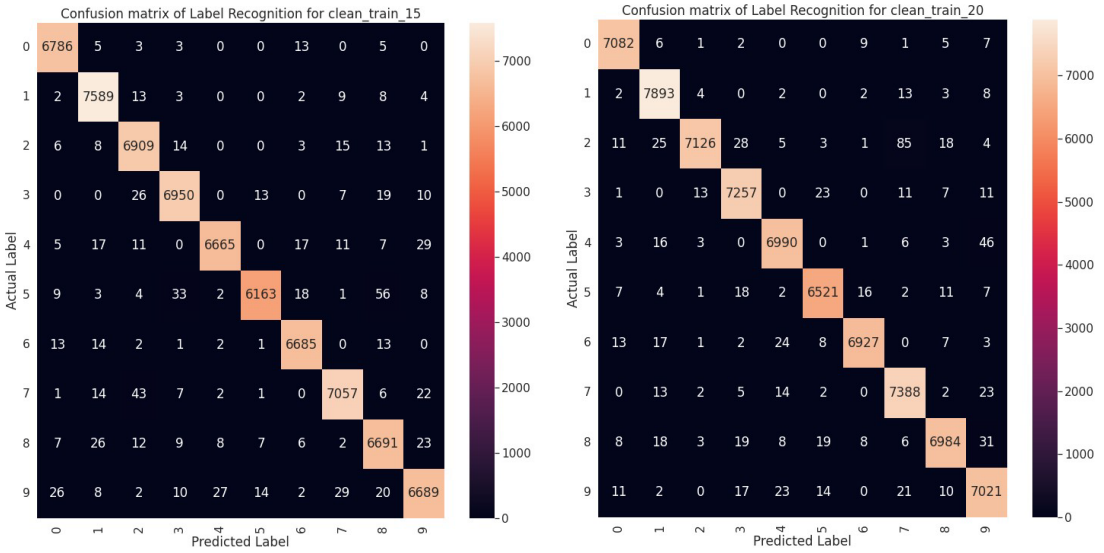
When looking at how the model performs against each label as training progress, we make comparisons of the clean-augmented MNIST generated confusion matrix and the triggered MNIST generated confusion matrix at Epoch 1 where we still observed some distinctions in our evaluation of overall dataset metrics. Figure 6.12, Figures 6.13 – 6.14, and Figures 6.15 – 6.16 show the

confusion matrices generated at Epoch 1 for original MNIST, clean-augmented MNIST, and triggered MNIST datasets, respectively.



(a) 5% (b) 10%

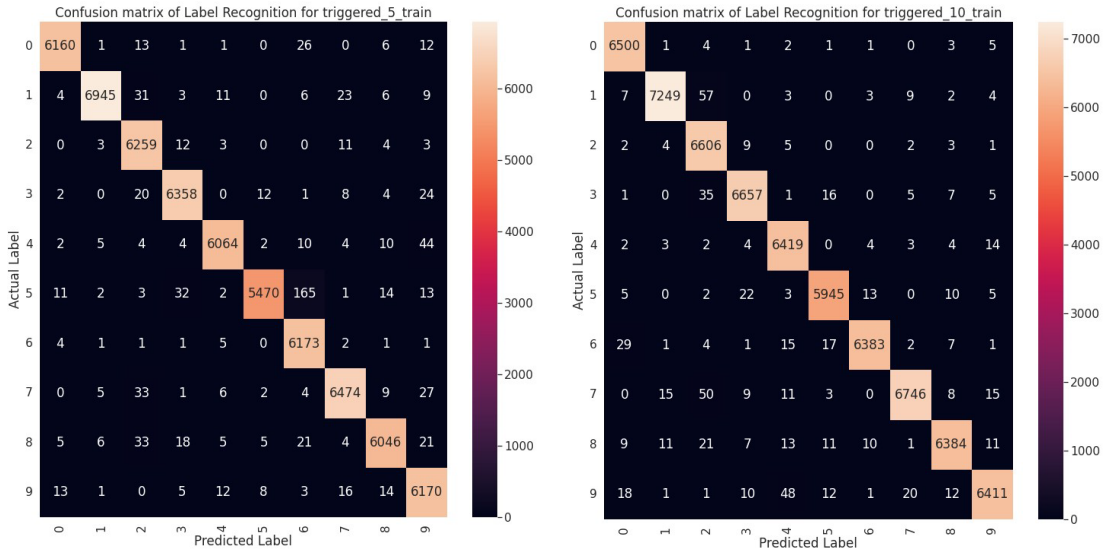
Figure 6.13: Confusion Matrix for Clean-Augmented MNIST (5% & 10%) at Epoch 1.



(a) 15% (b) 20%

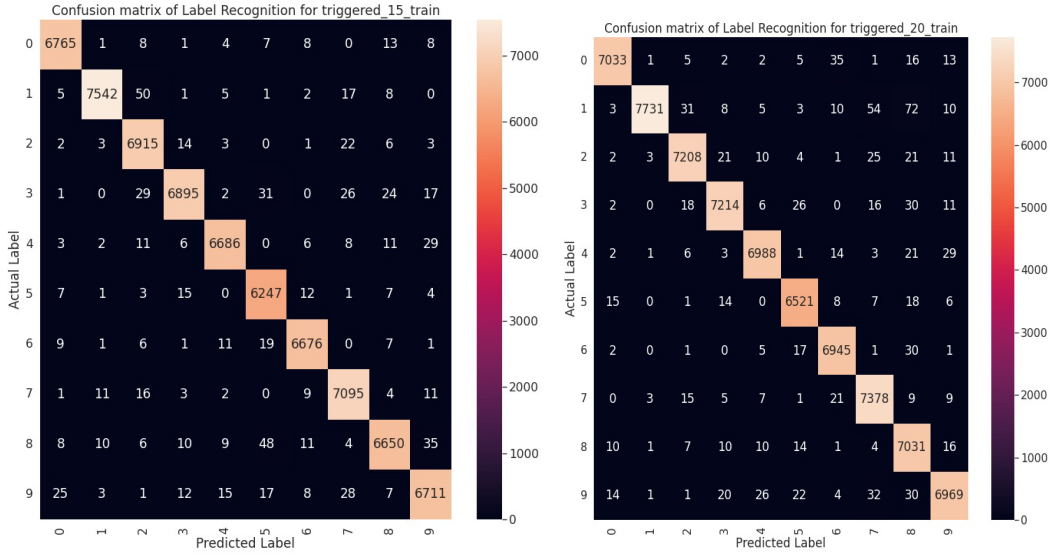
Figure 6.14: Confusion Matrix for Clean-Augmented MNIST (15% & 20%) at Epoch 1.

As one can observe, the number of incorrect predictions made across all labels by our model decreases from Epoch 0 to Epoch 1. Furthermore, the differences between clean-augmented MNIST and triggered MNIST confusion matrices and per-class metrics captured at Epoch 1 becomes less visible. This is consistent with what we have seen in our evaluation of overall dataset metrics where the results for each MNIST dataset resemble each other.



(a) 5% (b) 10%

Figure 6.15: Confusion Matrix for Triggered MNIST (5% & 10%) at Epoch 1.



(a) 15% (b) 20%

Figure 6.16: Confusion Matrix for Triggered MNIST (15% & 20%) at Epoch 1.

Take-away: With respect to **RQ3**, we cannot attribute the model's mistakes (e.g., incorrect predictions) to the % of poisonous data injected to the training set with certainty. While we might be able to do this for 5% of poisonous data injected to the training set when looking at Figure 6.10a, this observation does not hold when we increase the percentage of samples injected as observed in our comparison of clean-augmented and triggered MNIST at percentages 10 and higher at Epoch 0. The incorrect predictions are rather distributed among predicted labels at higher

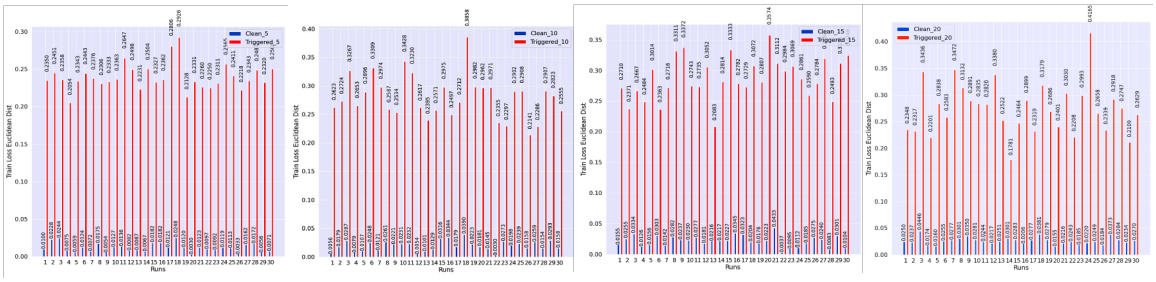
percentages. Moreover, the number of incorrect predictions decreases as training progresses for all MNIST datasets, which may indicate that for a poisoning attack to succeed, the bigger the triggered set percentage the better.

6.2.3 Stability of Observations Over Multiple Runs

We now turn our focus to the stability of observations made in our evaluation of overall dataset metrics as training provenance. Since we observed a consistent pattern when retraining our model on different percentages of input samples being injected into our training set, we expect to see consistent results for multiple runs as well.

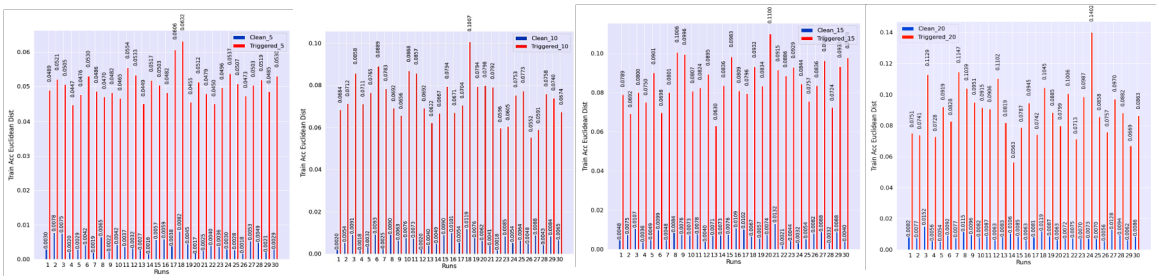
For the overall dataset metrics captured for MNIST, we observe the distance at Epoch 0 for triggered MNIST vs. original MNIST being significantly higher than that of clean-augmented MNIST vs. original MNIST. We believe this key observation demonstrates a potential indicator for data poisoning. To ensure we obtain similar results across runs that would further support our claim, we calculated the Euclidean Distance for Cross-Entropy and Accuracy. Euclidean Distance is essentially the sum of distances at every epoch for the given capture metric. We then generate a bar chart with the Euclidean Distance plotted against the number of runs that we carry out our experiments for. These bar charts are illustrated in Figures 6.17 and 6.18. Figure 6.17 represents the Euclidean Distance calculated for captured Cross-Entropy Loss. Figure 6.18 represents the Euclidean Distance calculated for captured Accuracy. As one can observe, the Euclidean Distance between the original and triggered MNIST for these captured metrics is always significantly higher compared to the Euclidean Distance between the original and clean-augmented MNIST when plotted against the 30 runs. This is attributed to the distance at Epoch 0 between original and triggered MNIST being consistently higher. Note that we do not analyze stability of per-class metrics as our evaluation of these per-class metrics is restricted to Epoch 0 and did not produce consistent observations across subsequent epochs.

Take-away: With respect to the "nondeterminism" challenge we highlighted in our problem statement, we conclude that our observations hold true across runs and do not change in the face of the innate stochastic nature of ML training computations.



(a) 5% (b) 10% (c) 15% (d) 20%

Figure 6.17: Loss-Distance (MNIST-Clean, MNIST-Clean-Augmented) vs. Loss-Distance (MNIST-Clean, MNIST-Triggered) Over 30 Runs.



(a) 5% (b) 10% (c) 15% (d) 20%

Figure 6.18: Train Accuracy-Distance (MNIST-Clean, MNIST-Clean-Augmented) vs. Train Accuracy-Distance (MNIST-Clean, MNIST-Triggered) Over 30 Runs.

6.3 Results on CIFAR-10 Experiments

We now examine the results for CIFAR-10 experiments when evaluating our approach against "Single Target Attack". In particular, we examine the use of captured overall metrics and per-class metrics as training provenance and determine if a poisoning threshold can be established based on the comparison of clean-augmented and triggered CIFAR-10 datasets.

6.3.1 Effectiveness of Overall Dataset Metrics as Training Provenance

Similar to MNIST experiments, we captured Categorical Cross-Entropy (CE) Loss and Sparse Categorical Accuracy during training of our model on CIFAR-10 as training provenance. From Figures 6.19 and 6.20, one can observe the captured Cross-Entropy Loss and Categorical Accuracy across epochs for each CIFAR-10 dataset. Figure 6.19 illustrates all clean-augmented CIFAR-10 datasets as well as the original CIFAR-10 dataset for comparison whereas Figure 6.20, on the other hand, illustrates all triggered CIFAR-10 datasets along with the original CIFAR-10 dataset for comparison. Both plots appear to be identical to each other, making it difficult to distinguish one from the other. Given the lack of detailed per-epoch insights, as we did for MNIST, we look at the training trajectory plotted against epochs for each of these datasets to see if we can gain further insight to any visible differences.

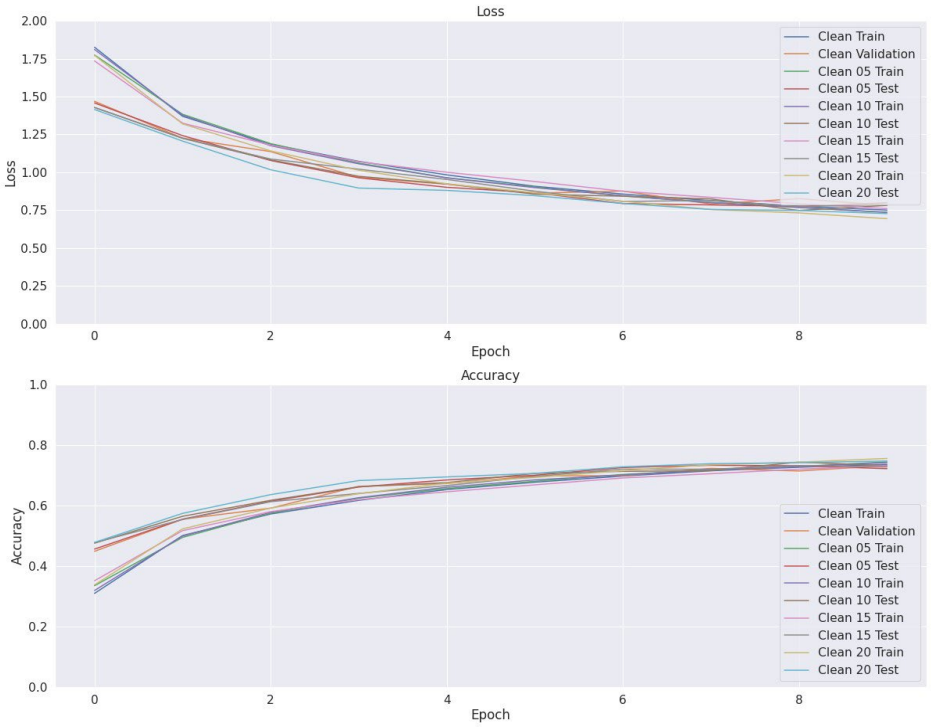


Figure 6.19: History Plot for Original vs. Clean-Augmented CIFAR-10.

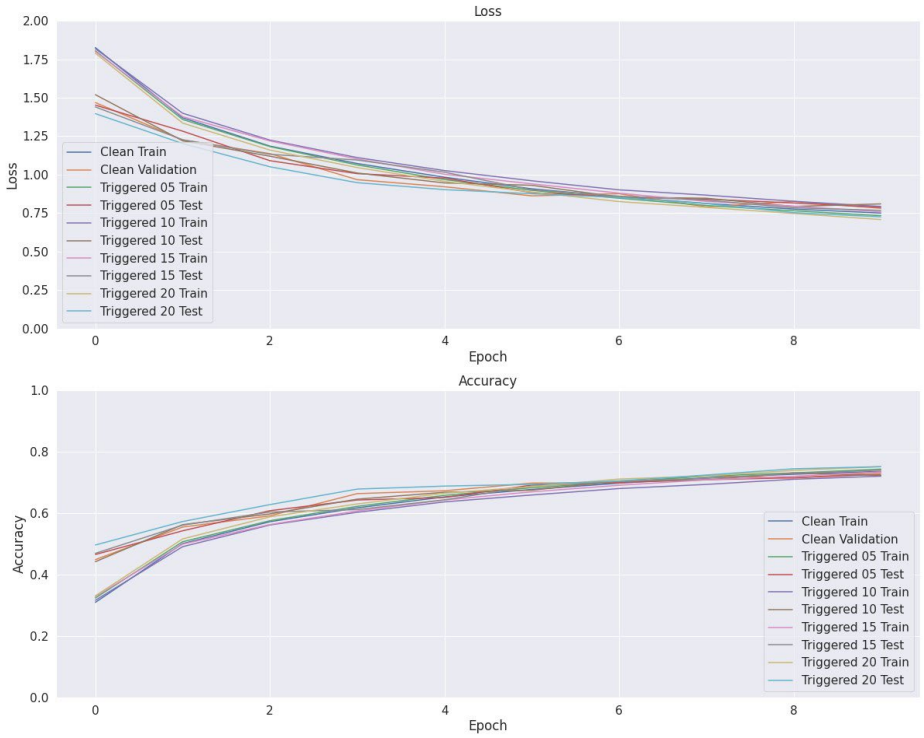
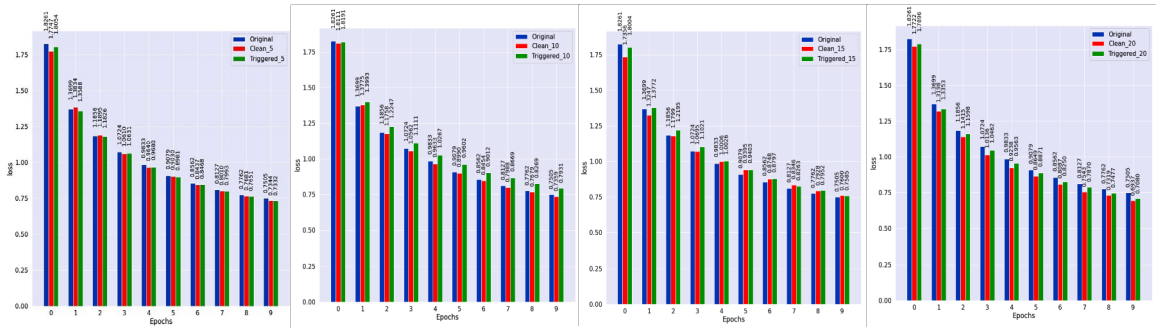


Figure 6.20: History Plot for Original vs. Triggered CIFAR-10.

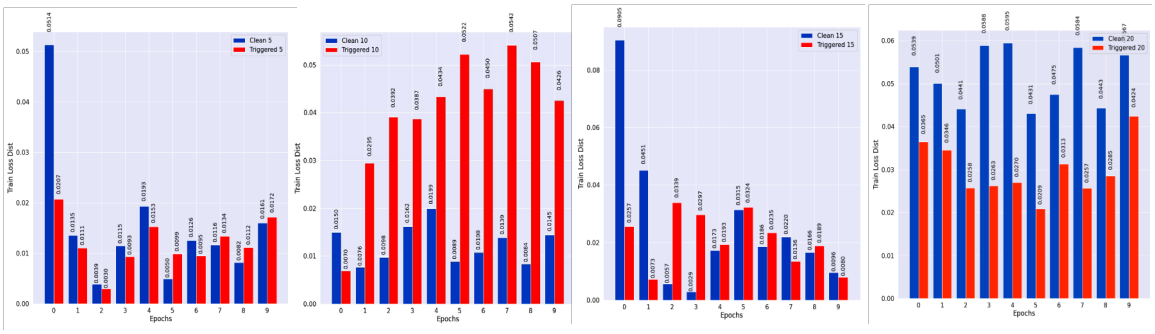


(a) 5% (b) 10% (c) 15% (d) 20%

Figure 6.21: Training Trajectory: Epoch vs. Loss ("Original" = CIFAR-10-Original, "Clean " = CIFAR-10-Clean-Augmented, "Triggered " = CIFAR-10-Triggered).

The training trajectory of Cross-Entropy Loss, the values obtained for the original, clean-augmented, and triggered CIFAR-10 datasets, is shown in Figure 6.21 for different percentages of clean-augmented and triggered samples added to the original dataset. Cross-Entropy Loss of the model trained on original CIFAR-10 dataset is slightly higher for most of the training process when compared to the clean-augmented and triggered CIFAR-10 datasets for 5% of input samples injected under "Label 5" during the retraining process. For 10% of input samples injected under "Label 5", Cross-Entropy Loss captured for the original and clean-augmented CIFAR-10 datasets are very close in value whereas the triggered CIFAR-10 dataset maintains a higher Cross-Entropy Loss through the training process. For 15% of input samples injected under "Label 5", Cross-Entropy Loss captured for the clean-augmented and triggered CIFAR-10 datasets are very close in value when compared to the original CIFAR-10 dataset. That said there appears to be fluctuation in the values as training progresses. For 20% of input samples injected under "Label 5", Cross-Entropy Loss captured for the clean-augmented maintains a lower value in comparison to the original and triggered CIFAR-10 datasets while original CIFAR-10 dataset maintains a higher value throughout the training process.

Given the fluctuations in captured Cross-Entropy Loss values for these datasets, we calculate the distance between Cross-Entropy Loss value captured for original and clean-augmented CIFAR-10 datasets at each epoch as well as the distance at each epoch between original and triggered CIFAR-10 datasets in hopes that any differences in the model's behavior that can be observed become more apparent. These calculated distances are displayed side by side in Figure 6.22 for different percentages of clean-augmented and triggered samples added to the original dataset. For 5% of injected samples, the Cross-Entropy Loss distance between original and clean-augmented

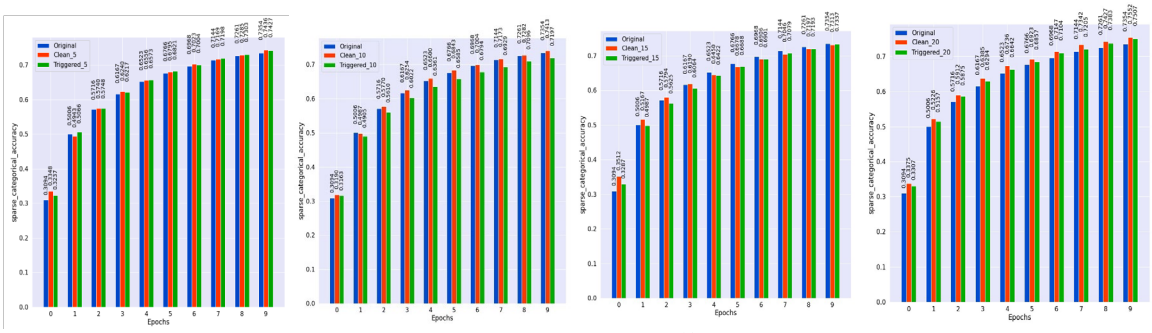


(a) 5% (b) 10% (c) 15% (d) 20%

Figure 6.22: Loss-Distance (Clean, CIFAR-10-Clean-Augmented) vs. Loss-Distance (Clean, CIFAR-10-Triggered).

CIFAR-10 at early epochs is higher than that of triggered CIFAR-10. As training progresses, however, the distance between original and triggered CIFAR-10 is slightly takeover for some epochs. The fluctuations in distance measurements seen here are attributed to fluctuations seen in Figure 6.21 (a) where for some epochs the Cross-Entropy Loss captured for clean-augmented CIFAR-10 is lower than the other two datasets and for others it is higher. For 10% of injected samples, the distance between original and triggered CIFAR-10 is substantially higher throughout the training process in comparison to the distance between original and clean-augmented CIFAR-10 datasets. This is in align with what we see in Figure 6.21 (b). For 15% of injected samples, the two distances differ at Epoch 0, but get closer in value with some fluctuation between epochs. This is attributed to fluctuations seen in Figure 6.21 (c) where the Cross-Entropy Loss value captured for clean-augmented CIFAR-10 is a lot lower than the original, but starts climbing up as training progresses to closely resemble the values captured for triggered CIFAR-10. For 20% of injected samples, the distance between original and clean-augmented CIFAR-10 is substantially higher throughout the training process in comparison to the distance between original and triggered CIFAR-10 datasets. This is attributed to the fact that Cross-Entropy Loss captured for clean-augmented CIFAR-10 is lower in value when compared to original and triggered CIFAR-10 as training progresses. This can be seen in Figure 6.21 (d). Given the fluctuations in distance measurements for Cross-Entropy Loss as input samples are injected, there isn't a clear pattern that supports using Cross-Entropy Loss as training provenance for establishing a detection threshold.

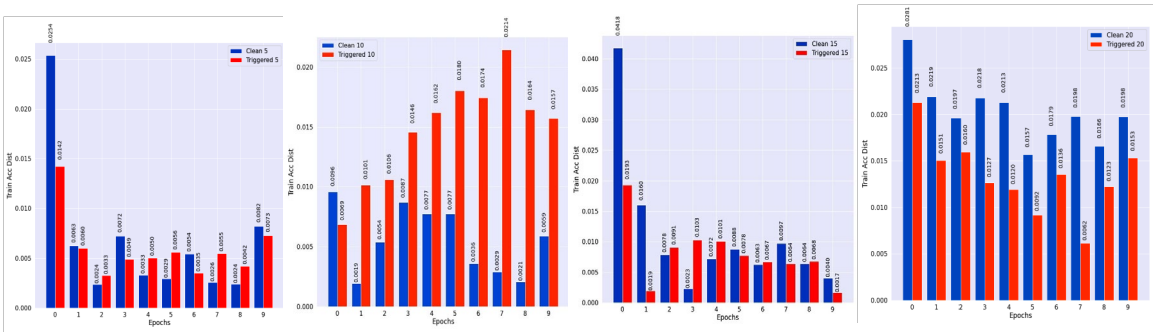
To help better visualize and compare the Accuracy values obtained for original, clean-augmented, and triggered CIFAR-10 datasets, the captured training trajectory of Accuracy is shown in Figure 6.23 for different percentages of clean-augmented and triggered samples added



(a) 5% (b) 10% (c) 15% (d) 20%

Figure 6.23: Training Trajectory: Epoch vs. Accuracy ("Original" = CIFAR-10-Original, "Clean " = CIFAR-10-Clean-Augmented, "Triggered " = CIFAR-10-Triggered).

to the original dataset. As one can observe for 5% of input samples injected into the training process, Figure 6.23 (a), Accuracy captured for both clean-augmented and triggered CIFAR-10 datasets is slightly higher than Accuracy captured for original CIFAR-10 dataset, and as training progresses, they start to resemble each other in value. For 10%, Figure 6.23 (b), Accuracy captured for clean-augmented CIFAR-10 maintains a slightly higher trajectory than original CIFAR-10 with triggered CIFAR-10 having a lower Accuracy throughout the training process. For 15%, Figure 6.23 (c), original CIFAR-10 maintains a higher Accuracy than both clean-augmented and triggered CIFAR-10 datasets. Clean-augmented CIFAR-10 does have a higher accuracy than triggered CIFAR-10 at first, but accuracy lowers to where they start to resemble each other close in value as training progresses. For 20%, Figure 6.23 (d), both clean-augmented and triggered CIFAR-10 maintain a higher accuracy than original CIFAR-10 with clean-augmented CIFAR-10 being the highest of the three.



(a) 5% (b) 10% (c) 15% (d) 20%

Figure 6.24: Train Accuracy-Distance (Clean, CIFAR-10-CleanAugmented) vs. Train Accuracy-Distance (Clean, CIFAR-10-Triggered).

From the captured training trajectory of Accuracy, we can take a closer look at the differences of how the model behaves when trained on a specific CIFAR-10 dataset by calculating the distance between Accuracy value captured for original and clean-augmented CIFAR-10 datasets at each

epoch as well as the distance at each epoch between original and triggered CIFAR-10 datasets. These calculated distances are displayed side by side in Figure 6.24 for different percentages of clean-augmented and triggered samples added to the original dataset. In alignment with what we observed in the training trajectory, some of the distance measurements are higher as a result of improved Accuracy captured for both clean-augmented and triggered CIFAR-10 where original CIFAR-10 achieved a lower Accuracy value throughout the training process. This contributes to the fluctuation observed in Figure 6.24 (a) where 5% of input samples injected into training set. For 20%, this is also the case with clean-augmented CIFAR-10 maintaining a higher Accuracy over the other two datasets; hence, why the calculated distance is higher than distance between original and triggered CIFAR-10 in Figure 6.24 (d). In some cases, the original and clean-augmented CIFAR-10 datasets achieve similar Accuracy values in comparison to triggered CIFAR-10 that achieves lower Accuracy. This is evident in Figure 6.24 (b) where calculated distance is higher for triggered CIFAR-10 at 10% of input samples being injected. For 15%, fluctuations are caused as a result of original CIFAR-10 achieving a higher Accuracy in comparison to the other two datasets as both clean-augmented and triggered CIFAR-10 start to resemble each other as training progresses. Given the fluctuations in distance measurements for Accuracy as input samples are injected, there isn't a clear pattern that supports using Accuracy as training provenance for establishing a detection threshold.

In general, it is difficult to make anything out of these results when looking at the calculated distances at epoch side by side with corresponding trajectory plots as the fluctuation in distance measurement values is a result of distance being taken at absolute value and does not indicate if the model's performance has improved, or not when injecting input samples. Given that clean-augmented CIFAR-10 and triggered CIFAR-10 achieve similar results in some instances, distance cannot be used for threshold detection, especially where both datasets improve the model's overall performance. We expect that data augmentation operations used to produce clean-augmented samples influence the model's performance in a way that the results obtained are closer to model's performance when trained on triggered CIFAR-10. Given clean samples that are closer in representation, we expect that our model should perform closer to that of the original CIFAR-10 and that significant distance between the captured metrics for these two datasets in comparison of the triggered CIFAR-10 could indicate data poisoning. As a model owner, we should welcome

improved metrics while maintaining that these captured metrics are reasonable for the model architecture and hyper-parameters that the dataset is trained on.

Take-away: With respect to **RQ1** and **RQ2**, the metrics (Cross-Entropy Loss and Accuracy) used to establish training provenance for clean and poisoned training data are not as effective as compared to our observations for MNIST experiments. There are no clear distinctions between captured training provenance of CIFAR-10 dataset with clean-augmented input samples and training provenance of CIFAR-10 dataset with triggered input samples as the captured metric values tend to fluctuate throughout the training process and as more input samples are added. Based on this, establishing a per-epoch threshold that can detect poisoning seems unattainable. The constant fluctuation in capture metric values suggests that the nondeterministic nature of machine learning may heavily influence any differences we observe when comparing how the model performs as it is trained on these datasets.

6.3.2 Effectiveness of Per-Class Metrics as Training Provenance

Like we did for MNIST experiments, we now look at per-class metrics for further insight. We specifically look at the injection of input samples under "Label 5" for CIFAR-10 via our evaluation of per-class metrics (precision, recall, and support) to see if any observation can be made that might explain why our model is performing the way it is, given that the data distribution is modified for a single label and not across the board for all labels. Unlike our evaluation of per-class metrics for MNIST Experiments where we relied on analyzing the confusion matrix at a given epoch, we will instead focus on analyzing history plots and distance at epoch for the per-class metrics to see if we can discover any visible patterns or differences between per-class metrics captured for Label 5 for original, clean-augmented, and triggered CIFAR-10 datasets. We also observe how these per-class metrics change over the course of training and as more input samples are injected under this label.

We first look at the captured per-class history plots for clean-augmented CIFAR-10 in comparison with triggered CIFAR-10. These plots are illustrated in Figures 6.25 and 6.26. The plots for Precision and Recall for both original vs. clean-augmented CIFAR-10 datasets and original vs. triggered CIFAR-10 datasets do not present a clear picture; however, one observation that we can make is that per-class metric values captured for clean-augmented CIFAR-10 datasets appear to rise and fall more often than the metric values captured for triggered CIFAR-10 datasets.

This may be attributed to the data augmentation operations used to generate the clean-augmented samples that are injected during the retraining process.

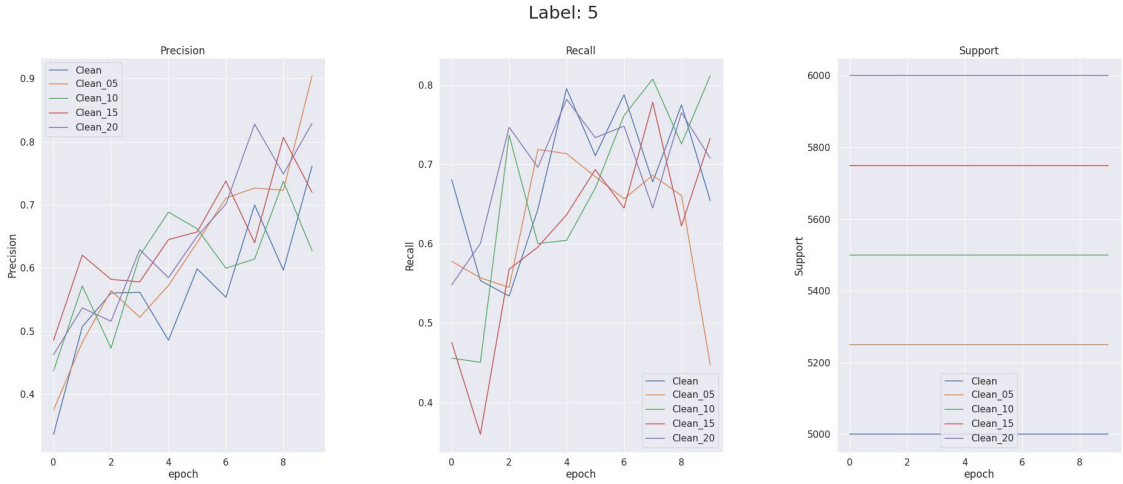


Figure 6.25: Per-Class History Plots for Label 5 Original vs. clean-augmented CIFAR-10.

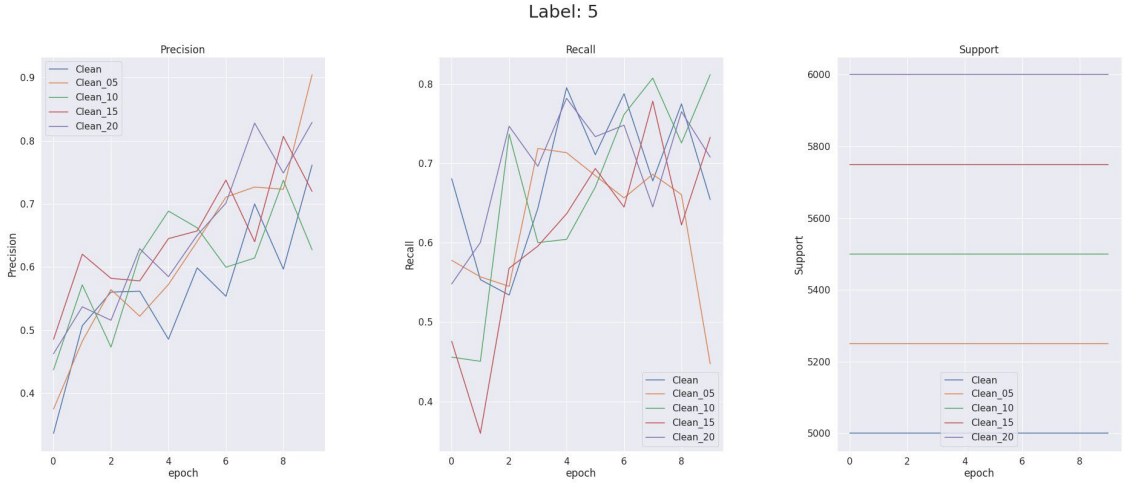
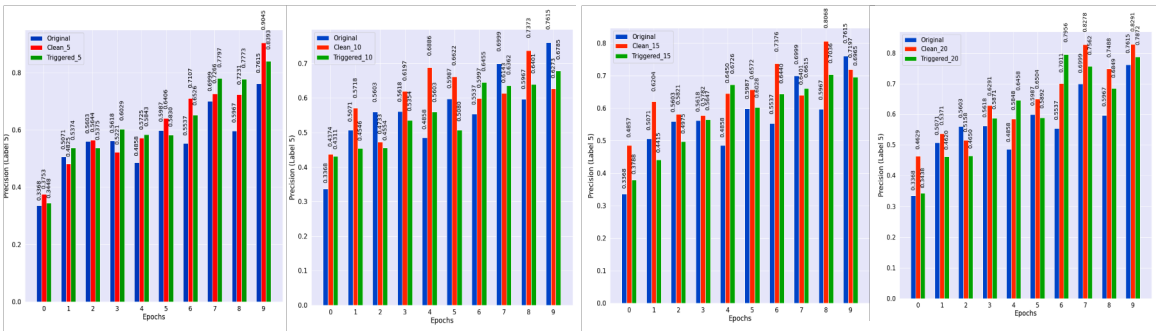


Figure 6.26: Per-Class History Plots for Label 5 Original vs. Triggered CIFAR-10.

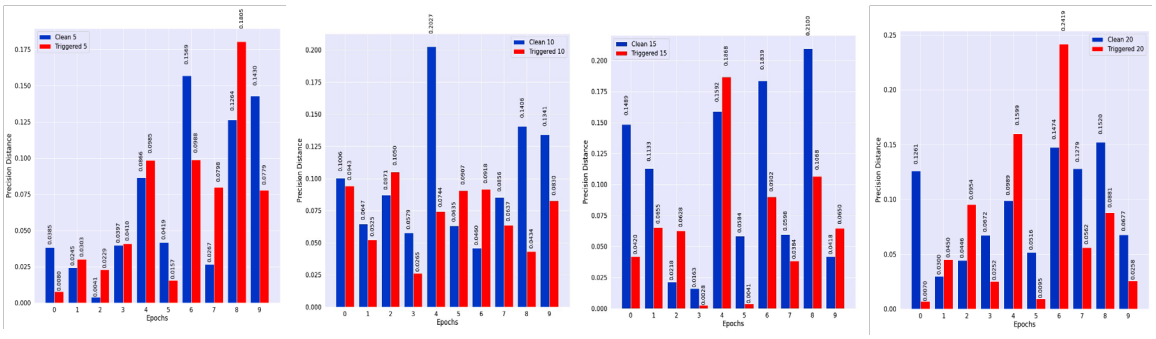
Next, we look at Precision training trajectory side by side with calculated distance at epoch between original vs. clean-augmented and original vs. triggered CIFAR-10 datasets to see if we can discover any clues that would indicate data poisoning and allow us to attribute the model's incorrect predictions to the percentage of poisonous samples injected under Label 5. Precision training trajectory is illustrated in Figure 6.27. Precision distance at epoch is illustrated in Figure 6.28. The Precision training trajectory supported by the calculated distance at epoch bar chart graphs show fluctuation in captured Precision values as training progresses. There isn't consistency, or a clear trajectory that shows the model performing better on one dataset over the

other. Moreover, there isn't any clear distinction between clean-augmented CIFAR-10 and triggered CIFAR-10. These observations are applicable for all percentages of input samples injected in the retraining process.



(a) 5% (b) 10% (c) 15% (d) 20%

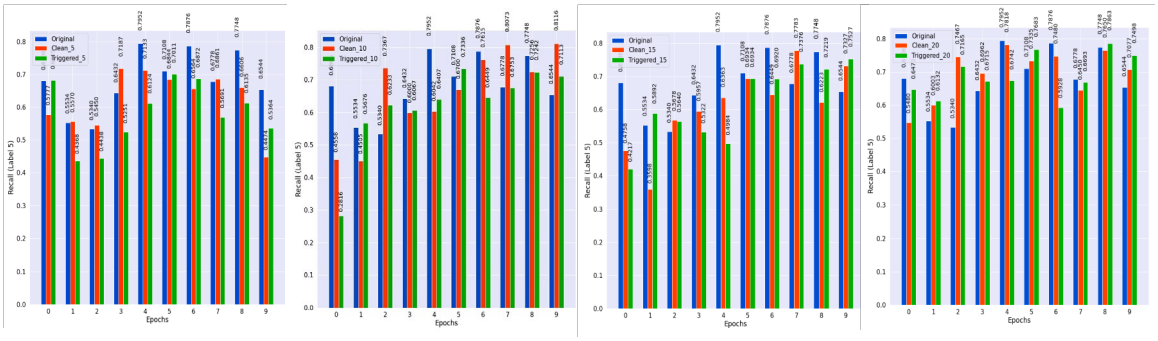
Figure 6.27: Precision Training Trajectory CIFAR-10 for Label 5.



(a) 5% (b) 10% (c) 15% (d) 20%

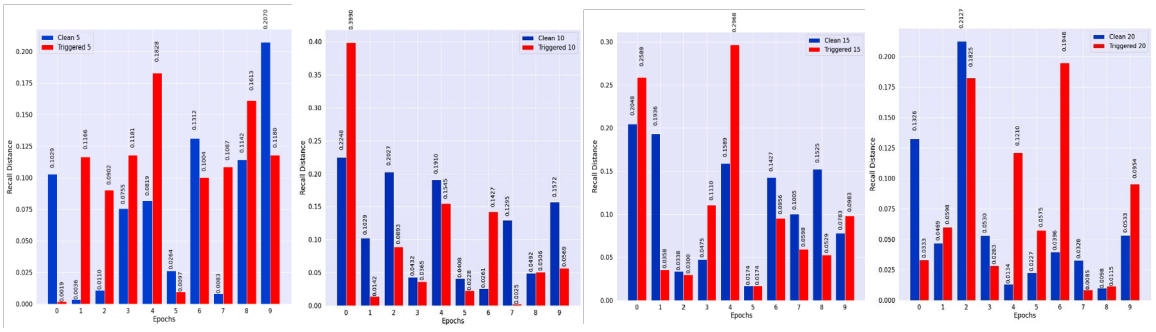
Figure 6.28: Precision Distance at Epoch CIFAR-10 for Label 5.

We then look at Recall training trajectory side by side with calculated distance at epoch between original vs. clean-augmented and original vs. triggered CIFAR-10 datasets to see if we can discover any clues that would indicate data poisoning and allow us to attribute the model's incorrect predictions to the percentage of poisonous samples injected under Label 5. Recall training trajectory is illustrated in Figure 6.29. Recall distance at epoch is illustrated in Figure 6.30. Similar in observations to Precision, the Recall training trajectory supported by the calculated distance at epoch bar chart graphs shows fluctuation in captured Recall values as training progresses. There isn't consistency, or a clear trajectory that shows the model performing better on one dataset over the other. Moreover, there isn't any clear distinction between clean-augmented CIFAR-10 and triggered CIFAR-10. These observations are applicable for all percentages of input samples injected in the retraining process.



(a) 5% (b) 10% (c) 15% (d) 20%

Figure 6.29: Recall Training Trajectory CIFAR-10 for Label 5.



(a) 5% (b) 10% (c) 15% (d) 20%

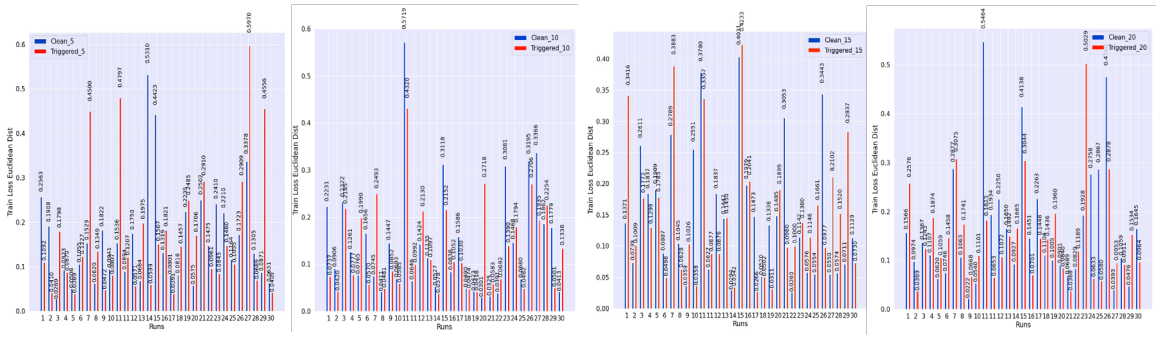
Figure 6.30: Recall Distance at Epoch CIFAR-10 for Label 5.

Take-away: With respect to **RQ1** and **RQ2**, per-class metrics do not seem to be effective at establishing training provenance for clean and poisoned training data; nor, are they effective in our approach for establishing threshold detection of poisoning. These per-class metrics tend to fluctuate as training progresses and do not hold a stable pattern that would allow one to establish any type of threshold detection. Similar to our evaluation of overall dataset metrics, there are no clear differences, or patterns observed when capturing these per-class metrics for clean-augmented and triggered CIFAR-10, rather the captured per-class metrics tend to fluctuate as training progresses. This is observed for all percentages of input samples injected under Label 5 of CIFAR-10. With respect to **RQ3**, based on our observations made during our evaluation of per-class metrics as training provenance, we are not able to attribute the model's mistakes (e.g., incorrect predictions) to the % of poisonous data injected to the training set as there are no clues that would distinguish poisonous data from clean data based on our comparative analysis.

6.3.3 Stability of Observations Over Multiple Runs

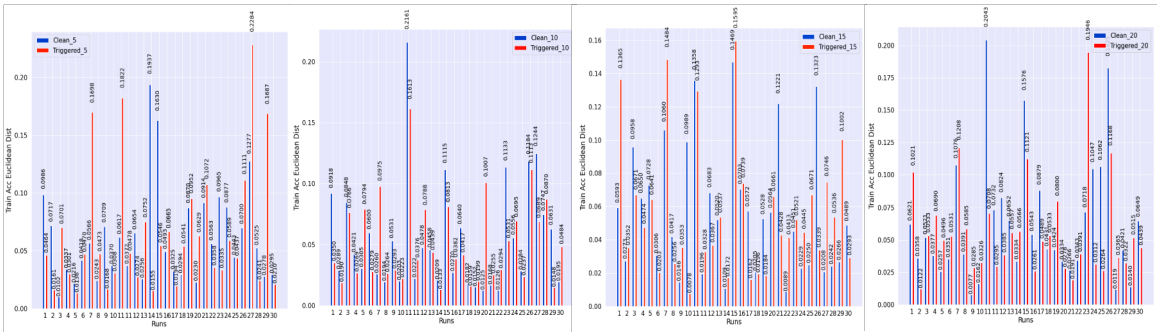
Finally, we provide further evidence that the results we obtain for our CIFAR-10 Experiments are shown to fluctuate as training progresses and that no meaningful clues are discovered in our comparative analysis of clean-augmented CIFAR-10 and triggered CIFAR-10 datasets.

For the overall dataset metrics captured for CIFAR-10, we observe fluctuations between the distance at epoch for triggered vs. original CIFAR-10 and the distance at epoch for clean-augmented vs. original CIFAR-10. Hence, we are not able to see any detectable differences between the two distance measurements. In comparing the Euclidean Distance for Cross-Entropy and Accuracy that we calculated for these captured metrics across the 30 runs we ran our experiments for, we see fluctuations from run to run in Figures 6.31 (Euclidean Distance calculated for captured Cross-Entropy Loss) and Figure 6.32 (Euclidean Distance calculated for captured Accuracy).



(a) 5% (b) 10% (c) 15% (d) 20%

Figure 6.31: Loss-Distance (CIFAR-10-Clean, CIFAR-10-Clean-Augmented) vs. Loss-Distance(CIFAR-10-Clean, CIFAR-10-Triggered) Over 30 Runs.



(a) 5% (b) 10% (c) 15% (d) 20%

Figure 6.32: Train Accuracy-Distance (CIFAR-10-Clean, CIFAR-10-Clean-Augmented) vs. Train Accuracy-Distance (CIFAR-10-Clean, CIFAR-10-Triggered) Over 30 Runs.

Take-away: In contrast to the consistently stable observations for MNIST experiments, for CIFAR-10, the augmentation and trigger seem to disrupt the original data distribution significantly enough that our observations fluctuate across 30 runs.

Chapter 7: Conclusion and Future Work

Conclusion: This work explores what we characterize as training provenance, the history of training metrics captured over training timeframe, for clues of data poisoning. We present a framework for capturing and analyzing training provenance to see if it can be leveraged for detection against poisoning attacks. By evaluating our proposed framework on MNIST, we found promising signs that training provenance can be leveraged to establish per-epoch detection threshold against data poisoning attack based on overall dataset metrics (e.g., loss, accuracy). While our results for CIFAR-10 did not pan out as anticipated, we attribute this to the disruption of the data distribution as result of the augmentation operations and simulated backdoor attack used to generate clean and poisoned CIFAR-10 input samples that we injected into the original training set during retraining of our ML model. As for per-class metrics, we discovered that these metrics offered little insight, and proved less effective as training provenance.

Future Work: Given the promising results we obtained for MNIST, we believe there is room for future work. For one, there is the possibility of evaluating the effectiveness of training provenance in pursuit of clues of data poisoning against other datasets as well as other data poisoning attacks. In this work, we only consider MNIST, CIFAR-10, and the backdoor attack described in [8, 15] for our evaluation. Another area to explore is the tracking of poisonous data throughout the training process for each label, especially during batch selection by tagging samples as “suspected” if we suspect that the sample might be poisonous. Our current framework does not track the new input samples that are injected during the retraining process. While a challenging aspect, we believe our framework can be expanded to track samples as we captured per-class metrics via generation of confusion matrix. This would prove more fruitful in attributing the ML model’s incorrect predictions to the percentage of newly injected input samples.

References

[1] Hojjat Aghakhani, Dongyu Meng, Yu-Xiang Wang, Christopher Kruegel, and Giovanni Vigna. Bullseye polytope: A scalable clean-label poisoning attack with improved transferability. In *IEEE European Symposium on Security and Privacy, EuroS&P 2021,* Vienna, Austria, September 6-10, 2021, pages 159–178. IEEE, 2021.

[2] Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, and Jaehoon Amir Safavi. Mitigating poisoning attacks on machine learning models: A data provenance based approach. In Bhavani Thuraisingham, Battista Biggio, David Mandell Freeman, Brad Miller, and Arunesh Sinha, editors, *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017,* Dallas, TX, USA, November 3, 2017, pages 103–110. ACM, 2017. doi: 10.1145/3128572.3140450. URL https://doi.org/10.1145/3128572.3140450.

[3] Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Jaehoon Amir Safavi, and Rui Zhang. Detecting poisoning attacks on machine learning in iot environments. In *2018 IEEE International Congress on Internet of Things, ICIOT 2018,* San Francisco, CA, USA, July 2-7, 2018, pages 57–64. IEEE Computer Society, 2018. doi: 10.1109/ICIOT.2018.00015. URL https://doi.org/10. 1109/ICIOT.2018.00015.

[4] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012,* Edinburgh, Scotland, UK, June 26 - July 1, 2012. icml.cc / Omnipress, 2012.

[5] Aleksandar Chakarov, Aditya V. Nori, Sriram K. Rajamani, Shayak Sen, and Deepak Vijaykeerthy. Debugging machine learning tasks. *CoRR*, abs/1603.07292, 2016. URL http://arxiv.org/abs/1603.07292.

[6] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian M. Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *CoRR*, abs/1811.03728, 2018. URL http://arxiv.org/abs/1811.03728.

[7] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR*, 2015

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

[9] Md Nahid Hossain, Sadegh M. Milajerdi, Junao Wang, Birhanu Eshete, Rigel Gjomemo, R. Sekar, Scott D. Stoller, and V. N. Venkatakrishnan. SLEUTH: real-time attack scenario reconstruction from COTS audit data. In *26th USENIX Security Symposium, USENIX Security 2017,* Vancouver, BC, Canada, August 16-18, 2017., pages 487–504, 2017.

[10] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, AISec '11, page 43–58, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450310031. URL https://doi.org/10.1145/2046684.2046692.

[11] Kiran Karra, Chace Ashcraft, and Neil Fendley. The trojai software framework: An opensource tool for embedding trojans into deep learning models. *CoRR*, abs/2003.07233, 2020. URL https://arxiv.org/abs/2003.07233.

[12] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). 2009. URL http://www.cs.toronto.edu/~kriz/ cifar.html.

[13] Yan LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/, 2020.

[14] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018,* San Diego, California, USA, February 18-21, 2018. The Internet Society, 2018.

[15] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 1265–1282, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367479. doi: 10. 1145/3319535.3363216. URL https://doi.org/10.1145/3319535.3363216.

[16] Sadegh M. Milajerdi, Birhanu Eshete, Rigel Gjomemo, and V. N. Venkatakrishnan. POIROT: aligning attack behavior with kernel audit records for cyber threat hunting. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019,* London, UK, November 11-15, 2019, pages 1813–1830. ACM, 2019.

[17] Sadegh Momeni Milajerdi, Rigel Gjomemo, Birhanu Eshete, R. Sekar, and V. N. Venkatakrishnan. HOLMES: real-time APT detection through correlation of suspicious information flows. In *2019 IEEE Symposium on Security and Privacy, SP 2019,* San Francisco, CA, USA, May 19-23, 2019, pages 1137–1152, 2019.

[18] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. Doug Tygar, and Kai Xia. Exploiting machine learning to subvert your spam filter. In Fabian Monrose, editor, *First USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET '08,* San Francisco, CA, USA, April 15, 2008, *Proceedings*. USENIX Association, 2008. URL http://www.usenix.org/events/leet08/tech/full_ papers/nelson/nelson.pdf.

[19] Giorgio Severi, Jim Meyer, Scott E. Coull, and Alina Oprea. Explanation-guided backdoor poisoning attacks against malware classifiers. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021,* August 11-13, 2021, pages 1487–1504. USENIX Association, 2021.

[20] Shawn Shan, Arjun Nitin Bhagoji, Haitao Zheng, and Ben Y. Zhao. Poison forensics: Traceback of data poisoning attacks in neural networks. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022,* Boston, MA, USA, August 10-12, 2022, pages 3575– 3592. USENIX Association, 2022. URL https://www.usenix.org/conference/ usenixsecurity22/presentation/shan.

[21] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security,* Vienna, Austria, October 24-28, 2016, pages 1528–1540. ACM, 2016.

[22] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *J. Big Data*, 6:60, 2019. doi: 10.1186/s40537-019-0197-0. URL https://doi.org/10.1186/s40537-019-0197-0.

[23] Jack W. Stokes, Paul England, and Kevin Kane. Preventing machine learning poisoning attacks using authentication and provenance. In *2021 IEEE Military Communications Conference, MILCOM 2021,* San Diego, CA, USA, November 29 - Dec. 2, 2021, pages 181–188. IEEE, 2021. doi: 10.1109/MILCOM52596.2021.9653139. URL https://doi.org/10.1109/MILCOM52596.2021.9653139.

[24] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy, SP 2019,* San Francisco, CA, USA, May 19-23, 2019, pages 707–723. IEEE, 2019. doi: 10.1109/SP.2019.00031. URL https://doi.org/10.1109/SP.2019.00031.

[25] Emily Wenger, Josephine Passananti, Arjun Nitin Bhagoji, Yuanshun Yao, Haitao Zheng, and Ben Y. Zhao. Backdoor attacks against deep learning systems in the physical world. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021,* virtual, June 19-25, 2021, pages 6206–6215. Computer Vision Foundation / IEEE, 2021.

[26] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y. Zhao. Latent backdoor attacks on deep neural networks. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019,* London, UK, November 11-15, 2019, pages 2041–2055. ACM, 2019.