

Secure Transfer Learning: Training Clean Model Against Backdoor in Pre-trained Encoder and Downstream Dataset

Yechao Zhang¹, Yuxuan Zhou¹, Tianyu Li¹, Minghui Li¹, Shengshan Hu¹, Wei Luo², Leo Yu Zhang³

¹Huazhong University of Science and Technology ²Deakin University ³Griffith University

{ycz, yuxuanchou, tianyu1i, minghuili, hushengshan}@hust.edu.cn
wei.luo@deakin.edu.au leo.zhang@griffith.edu.au

Abstract—Transfer learning from pre-trained encoders has become essential in modern machine learning, enabling efficient model adaptation across diverse tasks. However, this combination of pre-training and downstream adaptation creates an expanded attack surface, exposing models to sophisticated backdoor embedding at both the encoder and dataset levels—an area often overlooked in prior research. Additionally, the limited computational resources typically available to users of pre-trained encoders constrain the effectiveness of generic backdoor defenses compared to end-to-end training from scratch. In this work, we investigate how to mitigate potential backdoor risks in resource-constrained transfer learning scenarios. Specifically, we first conduct an exhaustive analysis of existing defense strategies, revealing that many follow a reactive workflow based on assumptions that do not scale to unknown threats, novel attack types, or different training paradigms. In response, we introduce a proactive mindset focused on identifying clean elements and propose the Trusted Core (T-Core) Bootstrapping framework, which emphasizes the importance of pinpointing trustworthy data and neurons to enhance model security. Our empirical evaluations demonstrate the effectiveness and superiority of T-Core, specifically assessing 5 encoder poisoning attacks, 7 dataset poisoning attacks, and 14 baseline defenses across 5 benchmark datasets, addressing 4 scenarios of 3 potential backdoor threats.

1. Introduction

Transfer learning (TL) has become an essential tool in machine learning applications, allowing developers to create sophisticated models by modifying existing ones to suit their specific tasks. This training paradigm is especially advantageous for users with limited computational and training resources, as it only requires collecting a small amount of training data and minor adaptation of a pre-trained encoder from third-party or open-source repositories. However, relying on pre-trained encoders or collecting data from external sources opens up significant security risks, exposing transfer learning models to malicious actors with access to the pre-trained model or the data to launch backdoor attacks. In backdoor attacks, an attacker may manipulate a few training samples of the victim by embedding a backdoor trigger and (mis)labeling them as a target class [1, 2] or directly

train a backdoor model [3–5] and deliver it to the victim. Either way, during inference time, they allow an attacker to stealthily control the victim model’s behavior to produce target outcomes given specific conditions.

A Challenging Defense Context. Transfer learning poses potential backdoor risks from both sources, creating a complex situation where the pre-trained encoder, the dataset, or both may be compromised, leading to three types of backdoor threats. However, prior research has primarily focused on only one vector of poisoning. Moreover, the transfer learning setup poses underappreciated challenges for general edge users with computational constraints when addressing backdoor threats. For these users, their limited computational resources typically can only support them in fine-tuning only part of the model parameters in transfer learning, *e.g.*, the last few layers. Nevertheless, almost all existing defenses are designed for scenarios where the entire model can be trained, usually requiring the support to train a model from scratch in an end-to-end learning manner.

A Exhaustive Defense Analysis. To further understand the combat against backdoor threats in this general yet challenging resource-limited transfer learning scenario, we exhaustively discuss and analyze existing defense strategies that presumably can help to mitigate the backdoor threats. However, we found that defenses effective in eliminating backdoors during end-to-end training from scratch fail to produce a clean model with high accuracy when adapted to the new defense context. Many existing defenses primarily rely on a reactive workflow to identify and eliminate poison elements associated with specific known threats, heavily depending on assumptions about these elements. Unfortunately, these assumptions do not scale to unknown threats, novel attack types, or different training paradigms, as summarized in Table 1.

A Proactive Mindset. To overcome the limitations of reactive workflows, we advocate for a proactive mindset. Instead of trying to identify all poisoned elements in a mixture of clean and poisoned samples from the post-attack model, we emphasize the importance of identifying high-credibility clean elements and proactively training with these trusted samples to address *unknown* backdoor threats. The rationale is that pinpointing a limited number of clean elements is significantly easier and more accurate than identifying all poisoned elements, particularly in complex scenarios where

poison can originate from different sources and backdoor triggers can take various forms. Defenders can then gradually expand this foundation by evaluating additional trustworthy elements while keeping untrusted elements separate.

A Bootstrapping Defense Framework. We propose a Trusted Core (T-Core) Bootstrapping framework that utilizes a proactive approach to learning a clean model through gradually bootstrapping verified data and model neurons. Our process begins by identifying a limited number of high-credibility samples from each class. This selection is based on assessing topological invariance across different layers of a well-trained model using the entire dataset. Next, we create a clean subset by treating these samples as seed data. We apply a method of unlearning the seed data while simultaneously learning from the remaining data and selecting the sample with maximal prediction loss to expand the dataset. Subsequently, we introduce a selectively imbalanced unlearn-recover process to filter clean channels within the pre-trained encoder, utilizing the clean subset we’ve established. Finally, we leverage both the trusted dataset and the trusted model neurons to bootstrap further the learning of a clean model, which involves gradually expanding the pool of clean samples and refining the model.

A Comprehensive Empirical Evaluation. We extensively evaluate a diverse set of 5 encoder poisoning attacks [4, 6–9], 7 dataset poisoning attacks [1, 2, 10–12, 12, 13], and 5 benchmark datasets of image classification (CIFAR-10, STL-10, GTSRB, SVHN, and ImageNet), covering 4 possible scenarios of all 3 types of backdoor threats in transfer learning. Throughout the paper, we evaluate 14 baseline defenses [14–27] within the defense context of transfer learning, and none of them match the effectiveness of our Trusted Core Bootstrapping framework in any module or in its entirety. This illustrates the potential of the proactive mindset we advocate in defending against unknown backdoor threats.

Finally, we summarize our contributions as follows:

- We identify a complex and challenging yet general backdoor threat model within the transfer learning scenario that previous research has overlooked.
- We conduct an exhaustive analysis of the existing backdoor defense in the defense context and reveal their limitations under the transfer learning scenario.
- We propose a proactive mindset as an alternative and introduce a Trusted Core Bootstrapping defense framework as an instantiation, providing concrete designs that are more robust and generalizable.
- We conduct extensive experiments to demonstrate the effectiveness of our Trusted Core Bootstrapping framework, as well as its superiority over existing designs in both module-by-module and end-to-end evaluation.

2. Preliminaries

In this section, we define our setup (Sec. 2.1), deliver the threat model of backdoor attacks under transfer learning, and the defense context for a general edge user (Sec. 2.2).

2.1. Training Procedure, Models and Data

We consider a popular transfer learning (TL) paradigm in image classification, *i.e.*, “*self-supervised pretraining* followed by *supervised fine-tuning*”. The self-supervised stage involves using a large set of unlabeled data, referred to as the *pre-training dataset* \mathcal{D}_{pre} , to develop an image encoder serving as a representation function $g : \mathcal{X} \mapsto \mathcal{E}$, where $\mathcal{X} = \mathbb{R}^d$ represents the input space, and \mathcal{E} is the embedding space. In the fine-tuning stage, a task-specific classification head $f(\cdot; \phi)$ is concatenated on top of $g(\cdot; \theta)$, creating a combined network $h(\cdot; \theta, \phi) = f(\phi) \circ g(\cdot; \theta) : \mathcal{X} \mapsto \Delta^C$, which is tailored for the *downstream task* of C -classes image classification and Δ^C representing the probability output space over the C classes. Overall, h comprises multiple layers, $\{h^{(l)} : l \in [1, N]\}$. Given an input x , the output of the neural network h is computed as $h(x) = f(g(x)) = (h^{(N)} \circ \dots \circ h^{(1)})(x)$, where f only represents the last few layers. Taking advantage of the feature extraction capability of the encoder, the fine-tuning process primarily updates ϕ on the C -class labeled *downstream training dataset* \mathcal{D} with training loss $\sum_{(x_i, y_i) \in \mathcal{D}} \ell(h(x_i), y_i)$ between the probability of output $h(x)$ and y , while generally freezing or making minor adjustments to θ . During inference, for any input feature $x \in \mathcal{X}$, the resulting classifier $F(\cdot) := \operatorname{argmax}_c h_c(\cdot)$ takes $F(x)$ as the predicted label. We use \mathcal{P} to denote the clean data distribution of the downstream classification task and $\mathbb{E}_{(x, y) \sim \mathcal{P}} \mathbb{I}(F(x) = y)$ as clean accuracy (ACC).

2.2. Threat Model and Defense Context

Our work considers the scenario in which an edge user aims to securely develop an image classifier based on an untrusted pre-trained encoder g and an untrusted training dataset \mathcal{D}_{pre} . This is relevant for a typical user who cannot guarantee the authenticity of the data it collects and does not have the resources (*e.g.*, computational power, and memory) to train a model from scratch, thus requiring a pre-trained model to facilitate its training. Given the growing trend of users accessing open-source datasets and pre-trained models from AI platforms such as HuggingFace, where anyone can freely upload resources, we believe this considered scenario is increasingly practical. In particular, our work focuses on the backdoor threats within this scenario.

Attack Goals. In general, regardless of how and where the trigger is injected, a desired backdoor attack should satisfy:

$$\mathbb{E}_{(x, y) \sim \mathcal{P}} \mathbb{I}(F(x) = y) \geq \tau_{ACC}, \quad (1)$$

$$\mathbb{E}_{(x, y) \sim \mathcal{P} | y \neq t} \mathbb{I}(F(\mathcal{T}(x)) = t) \geq \tau_{ASR}, \quad (2)$$

where $\mathcal{T} : \mathcal{X} \mapsto \mathcal{X}$ represents the adversary’s trigger function that transforms a benign input into a trigger-injected one, and t denotes the *target class* chosen by the adversary. Eq. (1) specifies that the attack shall not affect the standard functionality, that is, a high clean accuracy (above some τ_{ACC}) for the classification model F . Eq. (2), on the other hand, requires the backdoored classifier to classify any trigger-injected input $\mathcal{T}(x)$ as the target class t with a high probability, thus a high attack success rate (ASR).

Attack Vectors. In the context of transfer learning, we identify three potential backdoor threats to downstream tasks based on the adversary’s capabilities and attack approaches. These threats are illustrated as follows:

- **(Threat-1) Encoder Poisoning:** The attacker of this type aims to inject the backdoor by producing a poisoned image encoder g so that the downstream classifier F built based on g satisfies Eqs. (1) and (2). In this context, an attacker may operate as follows: 1) As a dishonest model provider, the attacker injects a backdoor into a pre-trained encoder and distributes it to users. 2) As a malicious third party, the attacker fine-tunes a clean encoder to include a backdoor and uploads it to platforms like HuggingFace. 3) As a disingenuous data provider, the attacker creates poisoned data for the pre-training dataset \mathcal{D}_{pre} of others, poisoning their encoder. Either way, the attacker targets a specific concept associated with a downstream task.
- **(Threat-2) Dataset Poisoning:** Another type of attacker can directly or indirectly poison the training dataset \mathcal{D} of the downstream user with a limited amount of trigger-injected data. This can occur when the downstream user collects data in an untrusted environment or from a disingenuous data provider. We assume the attacker of this type has no knowledge of the pre-trained encoder. Even if the attacker knows the user is resource-limited and likely to use transfer learning, the architecture and parameters of the pre-trained encoder remain unknown.
- **(Threat-3) Adaptive Poisoning:** In an extreme and yet feasible case, an adaptive attacker could potentially compromise both the pre-trained encoder g and the downstream dataset \mathcal{D} using the same backdoor trigger. This could happen if the user obtains both the model and training data from the same dishonest service provider. The attacker maximizes effectiveness by applying the same trigger to both g and \mathcal{D} . Consequently, even if the model or data is purified, the backdoor may remain effective unless completely eliminated.

We assume that in all these types of threats, the attacker does not know the training details of the downstream task. Once the poisoned data or encoder is delivered to the downstream user, how the user proceeds with the training or any potential defense is unknown to the attackers. It is important to note that multiple independent attackers can operate simultaneously. For instance, **Threat-1** and **Threat-2** may both occur: one attacker compromises the encoder g with trigger \mathcal{T}_1 targeting class t_1 , while another contaminates the downstream training data \mathcal{D} with a different trigger \mathcal{T}_2 targeting class t_2 . Each attacker remains unaware of the other’s presence.

Defender’s Goals. The defender we considered can be downstream users themselves or a cybersecurity service company like Darktrace¹ that provides automatic *on-premise* defense for users since the users may not be willing to adopt a cloud service and update their collected data, sometimes personal data. The defense framework takes the untrusted encoder g , and the untrusted training set \mathcal{D} as inputs and intends to build an accurate and safe classifier

at the endpoint. The goals of the defense are three-fold as follows: **1) Utility:** the resulting classifier F has a high ACC on the downstream task; **2) Security:** for any injected backdoor in g and \mathcal{D} , its effect will be eliminated, the resulting classifier itself F can correctly classify according to the actual semantic of the input, *i.e.*, exhibiting a low ASR without additional effect at the inference time; **3) Generalizability:** The defense should effectively counter all backdoor threats using various trigger embedding methods \mathcal{T} . Regardless of how the backdoor is injected, the defense methodology must maintain the effectiveness and security of the resulting F across different datasets, encoders, attack vectors, and hyperparameters (e.g., poison rate).

Defender’s Capabilities and Constraints. We illustrate the defender’s knowledge and constraints below:

- 1) **Access limited to g and \mathcal{D} :** We consider the defender has no access to additional data other than \mathcal{D} , such as the pre-training dataset \mathcal{D}_{pre} or a hold-out dataset containing a sufficient amount of clean samples. This is because model providers typically safeguard their pre-training dataset for proprietary or privacy reasons. Likewise, obtaining a completely clean hold-out dataset may not always be feasible. Nevertheless, we assume the defender possesses full autonomy over \mathcal{D} and g , which includes the ability to access, analyze, and modify their elements as needed.
- 2) **Ignorance of threat model:** The defender is unaware of the specific kind of backdoor threat it is dealing with. In other words, whether the encoder g or the dataset \mathcal{D} has been poisoned remains uncertain. To the defender, it is possible that neither, either, or both the encoder g and the dataset \mathcal{D} have been poisoned. As such, the defender has to treat both g and \mathcal{D} as untrustworthy.
- 3) **Computational constraints:** We assume the defense’s computational power is limited, reflecting the reality of defense deployments on edge users’ devices. Thus, computing gradients for the entire network is often infeasible due to the large size of pre-trained encoders and limited memory. This also agrees with the user’s intention of adopting the transfer learning pipeline, where the fine-tuning process usually mainly focuses on a few layers that are appended to the encoder. Thus, designing a memory-efficient defense is in the defender’s interest. Nevertheless, we assume the defense process can span a relatively long period to ensure the successful removal of potential backdoors from the fine-tuned model.

3. Methodological Analysis

3.1. Assessing Existing Defense Methodology

In this section, we evaluate the current defense methodologies within the defense context of TL to assess their effectiveness. We aim to identify their limitations and uncover the necessities required for a successful defense in the given context. Note that we do not elaborate the inference-time defenses, such as preprocessing that disrupt trigger-injected

1. <https://darktrace.com>

TABLE 1: An overview of existing backdoor defense methods, their assumptions or requirements, and how they fail or are not well-suited under the defense context.

| Defense Type | Approach | Assumption or Requirement | Limitation under the Defense Context |
|---------------------------|--|---|---|
| Poison Detection | Latent separation-based detection | Poisoned samples can be distinguished as outliers in the latent space. | Poisoned samples blend into the clean samples in the latent space when only tuning the classification head. |
| | Confusion training-based detection | Only backdoor correlations are well-preserved after confusion training. | A huge proportion of clean samples still have minimal losses after confusion training when only tuning the classification head. |
| Poison Suppression | Training restricted suppression | Demanding the computational resource to sufficiently train an entire encoder or DNN. | Edge users’ devices typically do not support computing the entire network’s gradients. |
| | Spurious correlation-based suppression | Backdoor features are easier to learn compared to clean features. | The learning pace advantage between backdoor and clean data differs across various backdoor attacks and threat types. |
| Poison Removal | Fine-tuning-based removal or Trigger synthesis-based removal | Requiring a hold-out clean dataset to fine-tune the model or reverse the trigger. | An additional completely hold-out clean dataset may not be feasible. |
| | Lipschitz-based removal | The channels with larger upper bounds of Lipschitz constant are more likely to be backdoor-related. | There is no absolute correlation between the Triggered-Activation Change and the Lipschitz constant’s upper bound. |

input before forwarding into the model [28–30] or input-based detection that assesses and rejects malicious inputs [23, 24, 31, 32]. This exclusion stems from our commitment to the model’s inherent security and creating a backdoor-free classifier for reliable deployment. Also, inference-time defenses can drastically raise processing costs by up to two orders of magnitude [31], and they often fail to provide a satisfactory security-utility trade-off (see Table 19).

Thus, we exhaustively explore the other types of defenses that can presumably mitigate the backdoor threats and can help to fulfill the defender’s goals in the defense context: (1) poison detection to identify backdoor poison data, (2) poison suppression to obstruct backdoor learning during training, and (3) poison removal to eliminate backdoor within a model after the attack. It is important to note that most of these defenses are proposed to address a specific backdoor threat, which only applies to part of the cases in the defense context. Therefore, we evaluate each defense fairly, focusing on its intended purpose or presumably solvable backdoor threat. From a defender’s perspective in the defense context, we assess whether existing methods can achieve their intended goals and discuss their limitations.

In this section, we primarily utilize two standard backdoor dataset poisoning methods, BadNet, and Blended attacks for **Threat-2**. For **Threat-1** and **Threat-3**, we leverage two encoder poisoning methods, BadEncoder [4], and DRUPE [7]. In **Threat-3**, we use the trigger of backdoor encoders for downstream poisoning. For both **Threat-2** and **Threat-3**, 20% of the samples from the target class are poisoned in the downstream dataset. More configuration details are included in Appendix-B.2 and Appendix-B.3.

3.2. Defense Type I: Poison Detection

Poison detection involves identifying and removing abnormal samples from a backdoor dataset to ensure the creation of a purified dataset, which is then used to train a clean model [11, 18–22]. Thus, poison detection techniques are meant to address dataset poisoning of **Threat-2**. However,

TABLE 2: Poison detection and subsequent evaluation of fine-tuned model under **Threat-2** with CIFAR-10 as the downstream dataset and STL-10 as pre-training dataset: 20% samples of the target class are poisoned ones. Both CT and ASSET are granted 1000 clean samples.

| Methods | BadNets | | | | Blended | | | |
|-------------------|---------|-------|-------|-------|---------|-------|-------|-------|
| | TPR↑ | FPR↓ | ACC↑ | ASR↓ | TPR↑ | FPR↓ | ACC↑ | ASR↓ |
| No Defense | - | - | 85.04 | 92.21 | - | - | 84.84 | 89.12 |
| STRIP | 6.08 | 6.18 | 82.80 | 87.16 | 6.00 | 5.15 | 83.24 | 83.01 |
| AC | 71.92 | 5.61 | 82.88 | 76.02 | 0.00 | 6.86 | 83.74 | 82.48 |
| Spectral | 49.76 | 37.19 | 78.33 | 59.24 | 64.48 | 36.81 | 78.44 | 39.21 |
| SPECTRE | 22.16 | 37.89 | 76.54 | 74.80 | 29.44 | 37.71 | 76.40 | 68.60 |
| CT | 0.00 | - | 85.04 | 92.21 | 0.00 | - | 84.84 | 89.12 |
| ASSET | 32.16 | 1.23 | 84.28 | 64.68 | 22.88 | 1.46 | 83.76 | 87.10 |

one would face a dilemma when adopting prior techniques for this defense context.

First, almost all prior literature on poison detection relies on a preset end-to-end supervised learning (SL) approach, assuming the training of the entire network is feasible². These approaches initialize and train an end-to-end DNN from scratch on the entire poisoned dataset, identifying and removing backdoor samples based on their distinctive characteristics in the post-attacked model. Most focus on feature characteristics, claiming poisoned samples can be identified as outliers in the latent space, as demonstrated in the BadNets detection of STRIP [22], AC [18], Spectral [19], and SPECTRE [16]. However, this assumption is invalidated by *latent space adaptive attacks* [11, 12, 12]. *Second*, to effectively filter out poisoned samples, many methods [20–22, 32, 33] require a certain number of clean samples to build their outlier detectors, which can conflict with the defender’s constraints in our context. This includes state-of-the-art poison detection methods, CT [21] and ASSET [20], which do not rely on latent separability. Both adopt a confusion training approach, optimizing over a hold-out

2. ASSET, the only method claiming effectiveness on TL, still requires training an additional full DNN.

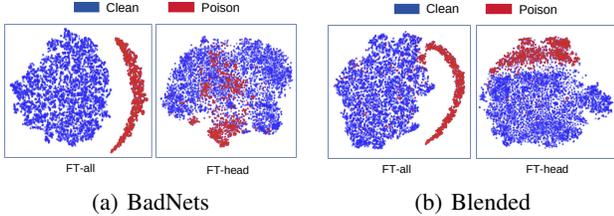


Figure 1: t-SNE comparison of feature space from a model trained on poisoned CIFAR-10: contrasting fine-tuning the entire network (FT-all) with fine-tuning only the 3-layer classification head (FT-head) under **Threat-2**.

clean set and the poisoned training dataset in the reverse direction. For instance, CT aims to reduce the model’s accuracy on clean samples by unlearning on clean data while learning from the training set, leading it to focus primarily on the backdoor data. Then, samples predicted correctly or with the smallest loss are considered poisoned [20, 21].

Despite assuming a clean hold-out set, we still evaluate the effectiveness of CT, ASSET, and others in detecting poisoned samples, measuring True Positive Rate (TPR) and False Positive Rate (FPR) in the context of **Threat-2** under the vanilla BadNets [1] and Blended [2] attacks. We further fine-tune models on each purified dataset for subsequent assessment of ASR and ACC, with model training adjusted for the classification head to fit the defense context. However, as shown in Table 2, none of these backdoor detection methods achieve a sufficient TPR or produce a clean classifier free from the backdoor of **Threat-2**.

We now examine why these detection methodologies are inadequate in this TL setting. In this context, the encoder is inherited with knowledge from pre-training, and fine-tuning is limited to the linear layers of the classification head. As a result, features from the penultimate layer, which the latent separation-based detector depends on, are merely a linear transformation of the fixed encoder’s output, leading to a condensed feature space that complicates the distinction between clean and poisoned samples. In Fig. 1, we show that if we fine-tune the entire network on the poisoned dataset, a clear boundary between poisoned and clean samples would still exist. However, when only the classification head is tuned, this distinction blurs or disappears. Furthermore, the narrow optimization space limits the ability to offset clean features in the poisoned set under CT and ASSET. As illustrated in Fig. 2, the low-loss region in the fine-tuned head scenario contains many clean samples, making them indistinguishable from poisoned samples. These results highlight the challenges of eliminating poisoned samples from a poisoned dataset in this defense context.

Remark 1 – The constraint on training the entire network also constrains the ability to detect backdoor-poisoned samples, as their supposedly distinguishable characteristics become indistinct. Detecting poisoned samples based on some assumption about them is unreliable because it may not scale across different training paradigms.

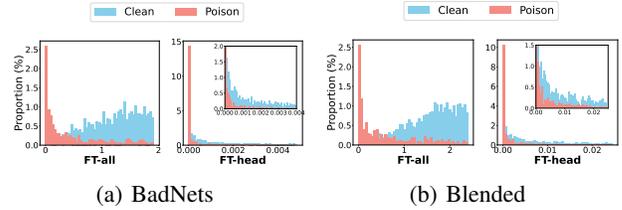


Figure 2: Distribution of poisoned and clean samples in the low-loss region (**lowest 40% loss** of the training set) after Confusion Training (CT), contrasting results from fine-tuning the entire network h (FT-all) and just the 3-layer classification head f (FT-head) under **Threat-2**.

3.3. Defense Type II: Poison Suppression

Poison suppression reduces the impact of poisoned samples during training, allowing for direct learning of a clean model under dataset poisoning as **Threat-2** without needing access to clean datasets. However, we found that many defenses of this type, by design, require training the entire network, limiting their applicability in the defense context. For instance, DBD [34] explicitly demands a self-supervised learning process over the entire encoder for semantic clustering. Other defenses [35, 36] observe that backdoors increase the linearity through activation functions, thus filtering out potentially affected neurons with more linearity during training. As a result, they depend on monitoring specific non-linear layers and activation functions during training, but fine-tuning only the classification head does not affect them.

The two poison suppression methods suitable for the defense context are *Anti-Backdoor Learning* (ABL) [15] and *Causality-inspired Backdoor Defense* (CBD) [14], both aim to suppress the model’s *spurious correlations* to mitigate backdoor attacks. Since DNNs often confuse causal relationships with statistical associations, favoring easier correlations, ABL and CBD target features that are easier to learn. ABL identifies samples with rapidly decreasing training losses and applies unlearning to these easy-to-learn samples. In contrast, CBD first trains an initial backdoor model on the poisoned dataset for a few epochs, allowing backdoor features to be learned while clean features remain underdeveloped. Then, it trains a clean model from the poisoned set while minimizing the mutual information between its representations and those of the initial backdoor model.

Since both ABL and CBD assume that backdoor features are learned faster, we infer that they may also be suitable for the case when both the encoder and downstream dataset contain the same trigger. Thus, we evaluate them under both **Threat-2** and **Threat-3**. We adjust their key hyperparameters to assess the security-utility trade-off. For ABL, the isolation ratio indicates the percentage of lowest-loss training samples selected for unlearning. For CBD, the training epoch number indicates how well backdoor’s spurious correlation is captured. According to the results in Tables 3 and 4, neither ABL nor CBD can achieve a high ACC and low ASR at any point. ABL shows much lower ACC compared to when the defense is inactive, with inad-

TABLE 3: ABL’s defense performance (ACC and ASR) under **Threat-2** and **Threat-3** using GTSRB as the downstream dataset and CIFAR-10 as the pre-training dataset while varying the isolation ratio. We also report the *number of isolation data* (NID) and the *number of poison data* (NPD) within the isolation data for each ratio.

| Threat Type | Threat-2 | | | | | | Threat-3 | | | | | | | | | |
|-------------|-----------------|-------|------|---------|-------|-------|-----------------|-----|-------|-------|------|-----|-------|-------|------|-----|
| | BadNets | | | Blended | | | BadEncoder | | | DRUPE | | | | | | |
| | ACC↑ | ASR↓ | NID | NPD | ACC↑ | ASR↓ | NID | NPD | ACC↑ | ASR↓ | NID | NPD | ACC↑ | ASR↓ | NID | NPD |
| No Defense | 77.51 | 91.12 | - | - | 77.57 | 83.14 | - | - | 78.10 | 99.97 | - | - | 74.42 | 98.58 | - | - |
| 0.01 | 46.70 | 64.92 | 392 | 0 | 46.63 | 49.83 | 392 | 0 | 44.74 | 98.29 | 392 | 0 | 20.44 | 96.20 | 392 | 1 |
| 0.05 | 38.30 | 61.78 | 1960 | 0 | 39.03 | 57.69 | 1960 | 0 | 42.05 | 97.96 | 1960 | 401 | 17.22 | 1.26 | 1960 | 458 |
| 0.10 | 27.90 | 72.45 | 3920 | 36 | 25.66 | 66.45 | 3920 | 0 | 30.14 | 95.86 | 3920 | 519 | 13.51 | 0.37 | 3920 | 464 |
| 0.15 | 17.50 | 83.70 | 5881 | 147 | 16.48 | 30.40 | 5881 | 0 | 17.51 | 0.00 | 5881 | 520 | 11.03 | 0.40 | 5881 | 480 |
| 0.20 | 14.26 | 68.12 | 7841 | 146 | 10.37 | 38.70 | 7841 | 122 | 13.62 | 0.03 | 7841 | 520 | 7.07 | 0.02 | 7841 | 482 |

equate ASR reduction, as it isolates clean samples instead of poisoned ones and fails to isolate poisoned samples at small isolation ratios (0.01 and 0.05) under **Threat-2**. CBD consistently decreases both ACC and ASR as the training epochs increase, indicating entanglement between poison and clean features in the representation, and minimizing mutual information could harm both.

To analyze their failure, we plot the loss trajectories under end-to-end supervised learning (SL), **Threat-2**, and **Threat-3**. Fig. 3 shows that the model learns backdoor data faster than clean data in the SL. Overall, in the TL, the model learns both types of data faster. Under **Threat-3**, poisoned data is learned faster than clean data only in the initial epochs. Under **Threat-2**, the model learns poisoned and clean data at similar paces for BadNets but learns clean data faster for Blended. Thus, there is no consistent evidence to support that backdoor features are easier to learn than clean features, indicating that both ABL and CBD target inaccurate objects for suppression, whether sample-wise or representation-wise.

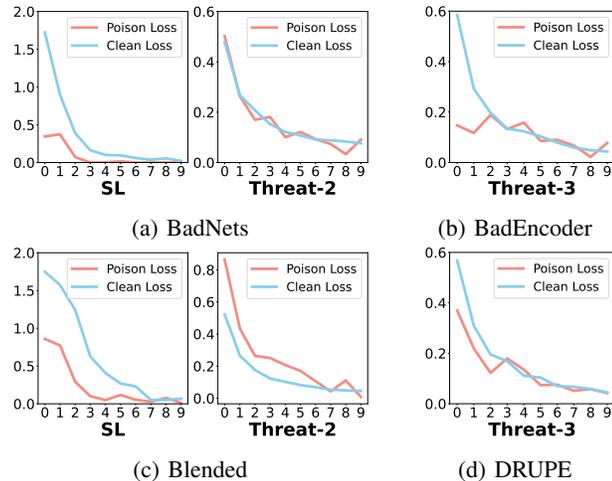


Figure 3: Comparison of average training losses for poisoned and clean samples in early epochs: **SL** trains from scratch on a poisoned dataset (BadNets or Blended). **Threat-2** fine-tunes a classifier head after a clean encoder on a poisoned dataset. **Threat-3** fine-tunes a classifier head after a poisoned encoder (BadEncoder or DRUPE) on a poisoned dataset with the same trigger.

TABLE 4: CBD’s defense performance (ACC and ASR) under **Threat-2** and **Threat-3** using GTSRB as the downstream dataset and CIFAR-10 as the pre-training dataset while varying the training epochs of the initial backdoor model.

| Threat Type | Threat-2 | | | | Threat-3 | | | |
|-------------|-----------------|-------|---------|-------|-----------------|-------|-------|-------|
| | BadNets | | Blended | | BadEncoder | | DRUPE | |
| | ACC↑ | ASR↓ | ACC↑ | ASR↓ | ACC↑ | ASR↓ | ACC↑ | ASR↓ |
| No Defense | 79.15 | 96.36 | 77.08 | 92.10 | 79.41 | 99.71 | 74.71 | 99.30 |
| 1 | 75.14 | 93.75 | 74.58 | 89.78 | 75.80 | 99.48 | 45.60 | 96.97 |
| 3 | 72.90 | 93.53 | 73.64 | 89.53 | 72.06 | 98.47 | 44.37 | 95.91 |
| 5 | 68.91 | 92.49 | 64.73 | 88.18 | 61.47 | 87.65 | 42.87 | 94.86 |
| 7 | 63.19 | 78.64 | 57.51 | 88.64 | 54.04 | 0.19 | 39.46 | 93.28 |
| 9 | 61.46 | 77.48 | 51.93 | 54.88 | 53.38 | 0.75 | 38.78 | 92.78 |

Remark 2 – The shift of the training paradigm also dramatically shifts the learning dynamic of both backdoor features and clean features. Locating the accurate “poison” for poison suppression based on some assumptions about how they are learned differently is also unreliable, especially in such a complex scenario where diverse triggers and backdoor threats are possible.

3.4. Defense Type III: Poison Removal

Poison removal works during the post-attack period, aiming to reconstruct a clean model by directly modifying the backdoor model, regardless of how the backdoor was injected, which makes it suitable to address **Threat-1**, **Threat-2**, and **Threat-3** all. However, most backdoor removal methods also rely on a clean dataset, including fine-tuning methods that use clean data to adjust the model [25, 26, 37] and trigger synthesis methods that generate potential backdoor patterns from clean data [38–41]. The only method that operates independently of clean data is Channel-wise Lipschitz Pruning (CLP) [17].

CLP is based on an observation that channels associated with backdoors exhibit greater changes in activation when presented with backdoor samples. Specifically, the *Trigger-Activated Change* (TAC) $TAC_k^{(l)}(x) = \|F_k^{(l)}(x) - F_k^{(l)}(\mathcal{T}(x))\|_2$ should be larger for backdoor-related channels than for normal ones, where $F_k^{(l)}(\cdot)$ is the output of the k -th channel at the l -th layer for input x . Additionally, the Lipschitz-continuous function $h_k^{(l)}(\cdot)$ has an *upper bound* of *Channel Lipschitz Constant* (UCLC) $\sigma(\theta_k^{(l)})$, which limits output differences. For any inputs z_1 and z_2 at layer l , $\|h_k^{(l)}(z_1) - h_k^{(l)}(z_2)\|_2 \leq \text{UCLC}$, and $\sigma(\theta_k^{(l)})$ is the spectral norm of model parameters $\theta_k^{(l)}$ at k -th channel of l -th layer. Since TAC is untraceable due to the unknown backdoor trigger \mathcal{T} , CLP identifies channels with high $\sigma(\theta_k^{(l)})$ values as backdoor-related. A larger $\sigma(\theta_k^{(l)})$ indicates more room for activation changes when a backdoor trigger is present, suggesting a higher likelihood of being backdoor-related.

For each layer, CLP sets the threshold for the l -th layer with K channels as $\mu^{(l)} + u \cdot s^{(l)}$, where $\mu^{(l)}$ and $s^{(l)}$ are the mean and standard deviation of the layer’s $\{\sigma(\theta_k^{(l)}) : k = 1, 2, \dots, K\}$. A smaller u results in more channels being pruned. We evaluate CLP’s effectiveness by varying

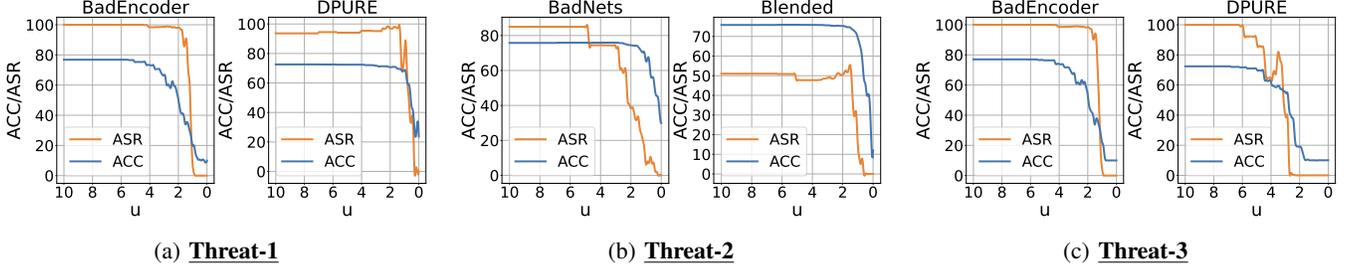


Figure 4: CLP performance on different types of threat from an **omniscient** defender’s perspective: (a) CLP is applied to the encoder g only because the pre-trained encoder is poisoned and the downstream dataset \mathcal{D} is clean under **Threat-1**; (b) CLP is applied to the linear layers of the classification head f because the encoder g is clean and only f is fine-tuned over a poisoned \mathcal{D} under **Threat-2**; (c) CLP is applied to f and g since both encoder and dataset are poisoned under **Threat-3**.

u from 0 to 10 in increments of 0.1 across all three backdoor threat types. Note that the original CLP is only employed on the convolution operators. In this experiment, we apply CLP to g , f , or both, depending on the specific threat model from an omniscient defense perspective. As shown in Fig. 4, CLP fails to achieve low ASR and high ACC for any u across all threat types. The ACC and ASR descend almost together with the decline of u , suggesting that wherever the backdoor is injected, the channels of the backdoor do not depend exclusively on the ones with the larger upper bound on activation changes (UCLC).

To understand the gap in CLP’s performance between the end-to-end SL and the FT-head setting, we visualize the correlation between UCLC and TAC in Fig. 5. Under SL, it does show a positive correlation ($corr$) between UCLC and TAC. However, under the transfer learning context with novel backdoor injection, there is no absolute correlation between UCLC and TAC, indicating a lack of strong evidence linking real backdoor-related channels (high TAC) with the presumptive ones (high UCLC). In other words, the backdoor trigger in SL inductively chooses the channels that can help induce TAC more easily. Instead, the backdoor trigger under TL is more covert as it depends on channels that are, in a way, more dispersed.

Remark 3 – The change in how the backdoor is injected also changes where the backdoor activation relies on. Blindly making assumptions on what kind of neurons are more likely to be responsible for backdoor, only based on the model itself, is also unreliable, as there is no guarantee of the distribution of where the backdoor activates.

3.5. Towards A Proactive Mindset

To mitigate backdoor risks and achieve a clean classifier with high ACC and low ASR, poison detection, poison suppression, and poison removal are the typical solutions for a defender. At a high level, each of them requires identifying what constitutes poisoned features or characteristics—whether on a sample-wise, representation-wise, or neuron-wise basis—followed by eliminating these poison elements. In a sense, these solutions can be seen as *reactive*, as they primarily focus on defending against somewhat *known*

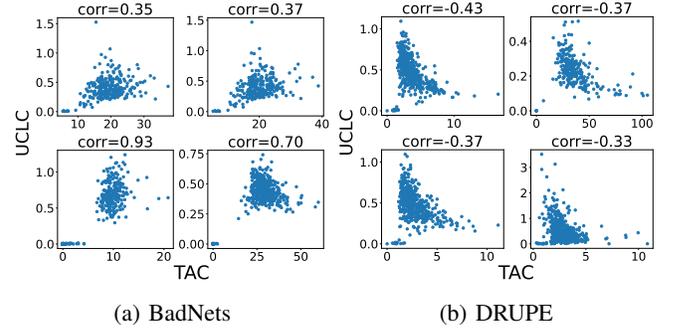


Figure 5: The scatter plot of the upper bound of activation changes (UCLC) versus actual triggered-activation changes (TAC) for all channels in the last four convolution layers. $corr$ presents the Pearson Correlation Coefficient. (a) depicts an end-to-end SL-trained ResNet18 classifier with BadNets dataset poisoning. (b) illustrates a ResNet18 encoder injected by DRUPE through encoder poisoning.

threats with a specific threat model (targeting what has or may have been compromised), then taking action to locate the poison elements and minimize their malicious influences by directly removing them or suppressing them. In general, these reactive strategies rely heavily on certain assumptions about the characteristics of the poison elements. However, these assumptions may not hold across *unknown* threats, novel types of attacks, or different training paradigms. As a result, their effectiveness is significantly compromised, as they fail to accurately identify poison elements.

In this study, rather than responding to specific backdoor threats by searching for poisoned elements, we advocate for a *proactive* mindset that focuses on identifying and amplifying clean elements to defend against *unknown* backdoor threats. On the one hand, poison elements can arise from multiple sources (such as the dataset and encoder) and manifest in various forms (including different types of backdoor triggers), making their accurate characterization challenging. In contrast, clean elements are generally more stable and uniform, making it easier to locate at least a small number of them. On the other hand, in a complex environment where many available resources are untrusted, it makes sense for defenders to utilize their limited trusted resources to bootstrap and obtain more trustworthy elements.

4. Trusted Core Bootstrapping

In this section, we present the pipeline of our Trusted Core Bootstrapping framework, which consists of three stages that align with the proactive mindset we promote in Sec. 3.5 to develop an accurate and secure classifier within the defense context. We start with sifting a limited high-credible seed data from the poisoned dataset (Sec. 4.1.1), followed by a data expansion that expands the seed data into a small clean subset (Sec. 4.1.2). Then an encoder neuron filtering is employed to filter the trusted model neurons for the clean downstream task (Sec. 4.2). Finally, we bootstrap the training process using the filtered encoder and clean data pool, progressively improving the model by gradually incorporating more trusted data (Sec. 4.3).

4.1. Sifting A Clean Set

In this section, we introduce how to extract a trusted subset, referred to as \mathcal{D}_{sub} , from the untrusted downstream dataset \mathcal{D} , that can support the successful training of a clean model for the downstream task. Intuitively, directly locating a clean subset will be non-trivial. Thus, we propose to first screen a limited number of representative data points from each class by setting a very high standard, then using these samples as seed data to bootstrap a clean subset.

4.1.1. Selecting Seed Data. As illustrated in Sec. 3.2, poison samples blend with benign samples, making it challenging to sift out clean samples through simple representation analysis in feature space. To address this, we sidestep the full dependence on latent separability in a single feature space and instead resort to examining the *topological invariance* of each sample across different DNN layers. The rationale is that the most representative samples should maintain consistency in their position within the data distribution as they propagate through the network. Specifically, we propose two rules to identify high-credible samples:

- **Majority Rule:** A high-credible sample should belong to the majority group of samples in a DNN layer. This rule assumes that the number of poisoned samples in a given class is smaller than that of clean samples, a condition generally met in backdoor poisoning. This rule imposes a global topological invariance, as it requires selection to avoid small groups in the whole data distribution. By enforcing this rule, we expect selected samples to be embedded within the core of the distribution across layers.
- **Consistency Rule:** A high-credible sample should have consistent nearest neighbors from its class across different DNN layers. This rule imposes local topological invariance as it requires selection to favor a consistent local neighborhood. By enforcing this rule, we expect that a selected sample is truly representative of its class, not because of the clustering effect of DNN.

We present the detail of the **topological invariance sifting** (TIS), which are based on these two rules, in Algorithm 1.

Concretely, given the untrusted encoder $g(\cdot; \theta_{\text{pre}})$ and the classification head $f(\cdot; \phi_{\text{down}})$ fine-tuned over \mathcal{D} , for

Algorithm 1: Topological Invariance Sifting

Input: θ_{pre} (Untrusted encoder parameters);
 \mathcal{D} (Untrusted downstream dataset);
Output: \mathcal{S} (A small set of seed data);
Parameters: L (Number of considered layers);
 m (Number of nearest neighbors);
 α (Seed data proportion selected);

```

1 /* Record samples activations of  $L$  layers */
2  $\mathcal{A} \leftarrow \text{RECORDACTIVATIONS}(h, L, [\mathcal{D}_1, \dots, \mathcal{D}_K]);$ 
3 for  $k \leftarrow 1$  to  $K$  do
4     /* Cluster activations in each layer */
5     for  $l \leftarrow 1$  to  $L$  do
6          $C^l \leftarrow \text{CLUSTERING}(\mathcal{A}[l, k]);$ 
7          $C_{\text{max}}^l \leftarrow \text{MAX}(C^l);$ 
8      $\mathcal{B}_k \leftarrow C_{\text{max}}^1 \cap C_{\text{max}}^2 \dots \cap C_{\text{max}}^L;$ 
9     /* Obtain consistent neighbors of  $x$  */
10    for  $x \in \mathcal{B}_k$  do
11         $\text{Numbers} \leftarrow \emptyset;$ 
12        for  $l \leftarrow 1$  to  $L$  do
13             $N_x^l \leftarrow \text{NEARESTNEIGHBORS}(x, \mathcal{A}[l, k], m);$ 
14             $\text{Numbers.APPEND}(|N_x^1 \cap N_x^2 \dots \cap N_x^L \cap \mathcal{D}_k|);$ 
15        /* Sort by consistent neighbors number */
16         $\mathcal{B}_k^* \leftarrow \text{SORTDESCENDING}(\mathcal{B}_k, \text{KEY} = \text{Numbers});$ 
17         $\mathcal{S}_k \leftarrow \mathcal{B}_k^*[0 : \alpha \times |\mathcal{D}_k|];$ 
18  $\mathcal{S} \leftarrow \mathcal{S}_1 \cup \mathcal{S}_2 \dots \cup \mathcal{S}_K;$ 
19 return  $\mathcal{S}$ 

```

each class k , we record the activations of its all samples \mathcal{D}_k of the L layers before the final layer of the entire network $h = f(\phi_{\text{down}}) \circ g(\cdot; \theta_{\text{pre}})$ (line 2). Then, we proceed with the data sifting in each \mathcal{D}_k .

We first apply the majority rule using a clustering procedure on the output activations of each layer. We notice a poison detection [18] uses clustering to identify poisoned samples, relying on the last hidden layer’s representation with K-means for two-class clusters. However, in our approach, the majority rule is applied before the consistency rule, aiming to remove samples from any smaller clusters that may contain poisoned samples across all layers. Thus, we utilize density-based clustering to partition \mathcal{D}_k and obtain the intersection \mathcal{B}_k of the largest clusters in each layer (lines 6 to 8). Unlike fixed-number clustering methods, density-based clustering dynamically determines the number of clusters, effectively detecting irregularly shaped clusters and managing both dense and sparse regions.

After applying the majority rule, we use the consistency rule to refine the output \mathcal{B}_k . For each sample x in \mathcal{B}_k , we identify its nearest m neighbors N_x^l in each layer l (lines 12 and 13). We then filter these neighbors by class and assess how many are consistent across all layers. The number of consistent neighbors from the same class across L layers can be denoted as $|N_x^1 \cap N_x^2 \dots \cap N_x^L \cap \mathcal{D}_k|$ (line 14). Finally, we select samples with the most consistent neighbors from each class, maintaining a proportion of α (lines 16 and 17) to ensure an equal number of seed data per class.

4.1.2. Bootstrapping the Clean Set. Once we obtain a set of high-credible samples from each class, we can use them as seed data to bootstrap a clean subset. In Sec. 3.2, we

Algorithm 2: Seed Expansion

Input: \mathcal{S} (A small set of clean seed data);
 \mathcal{D} (Downstream dataset);
Output: \mathcal{D}_{sub} (A clean subset);
Parameters: r_{expand} (Select ratio for expansion);

```
1  $\mathcal{D}_{\text{sub}} \leftarrow \mathcal{S}$ ;  
2 repeat  
3   Randomly initialize  $\phi$ ;  
4   Fine-tune  $\phi$  with  $\ell(f(\cdot; \phi) \circ g(\theta_{\text{pre}}, \cdot))$  over  $\mathcal{D}$ ;  
5   /* Confusion training with  $\mathcal{D}_{\text{sub}}$  */  
6    $\phi \leftarrow \text{CT}(\phi, \mathcal{D} \setminus \mathcal{D}_{\text{sub}}, \mathcal{D}_{\text{sub}})$  with Eq. (3);  
7    $\text{Loss} \leftarrow \{\ell(h(x), y) \mid (x, y) \in \mathcal{D} \setminus \mathcal{D}_{\text{sub}}\}$ ;  
8   /* Sort by loss in descending order */  
9    $\mathcal{R} \leftarrow \text{SORTDESCENDING}(\mathcal{D} \setminus \mathcal{D}_{\text{sub}}, \text{KEY} = \text{Loss})$ ;  
10  /* Add samples with the largest loss */  
11   $\mathcal{D}_{\text{sub}} \leftarrow \mathcal{D}_{\text{sub}} \cup \mathcal{R}[1 : |\mathcal{D} \setminus \mathcal{D}_{\text{sub}}| \times r_{\text{expand}}]$ ;  
12 until  $i \leftarrow 1$  to  $I$ ;  
13 return  $\mathcal{D}_{\text{sub}}$ 
```

introduced a state-of-the-art poison detection method called Confusion Training (CT) [21], which uses a hold-out clean dataset to offset clean features, resulting in lower loss values for poisoned samples. However, this approach is ineffective in identifying poisoned samples in the defense context, as the lowest loss area is mixed with both poisoned and clean samples, due to the optimization limited to the classification head, as shown in Fig. 2. Nevertheless, the largest loss area is exclusively filled with clean samples regardless of the dataset poisoning type (see Fig. 6). Therefore, we tailor CT as an expansion tool to bootstrap a clean subset using the seed data.

We perform two concurrent minimizations: one over poisoned dataset and another over mislabeled clean samples. The joint loss is defined as:

$$\lambda \cdot \ell(h(\mathbf{X}_i), \mathbf{Y}_i) + (1 - \lambda) \cdot \ell(h(\mathbf{X}'_i), \mathbf{Y}^*), \quad (3)$$

where $(\mathbf{X}_i, \mathbf{Y}_i)$ is a mini-batch from the poisoned dataset, and $(\mathbf{X}'_i, \mathbf{Y}^*)$ is created by randomly mislabeling a clean batch from a clean base set. The overall procedure is outlined in Algorithm 2. We start with a small set of clean seed data \mathcal{S} and the entire untrusted dataset \mathcal{D} . We randomly initialize the classification head parameters ϕ and fine-tune it on $\mathcal{D} \setminus \mathcal{D}_{\text{sub}}$, then apply confusion training (CT) using both $\mathcal{D} \setminus \mathcal{D}_{\text{sub}}$ and seed data \mathcal{D}_{sub} . After CT, we add the r_{expand} samples with the highest loss from $\mathcal{D} \setminus \mathcal{D}_{\text{sub}}$ to \mathcal{D} . We then sort the remaining samples by loss in descending order and add the top $|\mathcal{D} \setminus \mathcal{D}_{\text{sub}}| \times r_{\text{expand}}$ samples to the clean subset \mathcal{D}_{sub} . This process is repeated until we achieve a sufficient subset, stopping when \mathcal{D}_{sub} reaches 10% of \mathcal{D} .

4.2. Filtering the Encoder Channel

After obtaining a trusted subset of downstream data \mathcal{D}_{sub} , the next stage is to locate clean neurons within the encoder. To facilitate defense operations on user devices with limited computational resources, which can only support the gradient computation of a limited number of model parameters, we introduce a selectively imbalanced unlearn-recover

process to identify clean neurons in the pre-trained encoder. Typically, an encoder DNN is constructed in a modular fashion, with each module containing a transformation layer for feature extraction and a normalization layer for scaling and shifting features. For instance, in a convolutional network, different convolution channels extract both backdoor-related and normal features, with normalization parameters adjusting them for the final classification representation. As shown in Sec. 3.4, assumptions about which neurons may be responsible for the backdoor are unreliable in data-free analysis using CLP. Therefore, we propose utilizing both the untrusted downstream dataset \mathcal{D} and the trusted \mathcal{D}_{sub} to directly locate clean channels in the encoder. The procedure of our encoder channel filtering is described in Algorithm 3. **Selective Unlearning.** We start by an unlearning process on the downstream data \mathcal{D} as follows:

$$\max_{\theta_{\text{norm}}} \mathbb{E}_{(x,y) \in \mathcal{D}} [\ell(f(\phi_{\text{down}}) \circ g(x; \theta_{\text{pre}}), y)] \quad (4)$$

Here, $f(\phi_{\text{down}})$ represents the previously fully trained classification head on \mathcal{D} , with the encoder $g(\cdot; \theta_{\text{pre}})$ fixed. θ_{norm} denotes the parameters within the normalization layers. This approach allows the model to intentionally “lose” some performance over the entire downstream data while keeping other layers and classification heads frozen during the unlearning phase. Additionally, this process is memory-efficient, selectively adjusting only the normalization parameters, which typically constitute less than 1% of the encoder parameters. Consequently, the encoder’s ability to extract clean or backdoor features, along with the functionality of the classification head, remains largely intact.

Filter Recovering. The recovering process aims to restore the model’s ability to predict clean samples, which the previous unlearning process has impaired. This recovery is crucial for discerning clean channels. By enforcing recovery of the downstream clean task, the network enhances the association between clean samples and clean channels over backdoor-related ones. To achieve this, we introduce a filter mask \mathbf{m}^κ to indicate which operators to select for enhancing associations. Formally, the recovery works by solving the following minimization:

$$\min_{\mathbf{m}^\kappa} \mathbb{E}_{(x,y) \in \mathcal{D}_{\text{sub}}} \left[\ell \left(f(\phi_{\text{down}}) \circ g(x; \mathbf{m}^\kappa \odot \hat{\theta}_{\text{pre}}), y \right) \right], \quad (5)$$

where $\hat{\theta}_{\text{pre}}$ denotes the modified encoder parameters updated by the unlearning process, which only affects normalization layers, and \mathbf{m}^κ denotes the masks applied to the channels in transformation layers. In this minimization, \mathbf{m}^κ , which constitutes less than 1% of the encoder parameters, acts as a soft filter to identify the channels most beneficial for recovering clean task performance, reinforcing correct associations between inputs and outputs. We initialize \mathbf{m}^κ as all ones and apply a clipping operation during updates (line 9).

Channel Filtering. Once recovery is complete, the mask values indicate each channel’s contribution to the clean task. A high value suggests that the channel (and its corresponding neurons) has a greater correlation with the downstream

Algorithm 3: Encoder Channel Filtering

Input: θ_{pre} (Untrusted encoder parameters);
 \mathcal{D}_{sub} (A clean subset);
 θ_{norm} (Unlearning parameters)
Output: \mathbf{m}^{κ} (A suspected filter binary mask);
Parameters: ACC_{min} (Clean accuracy threshold);
 σ (Filter threshold);

```
1 /* Selective Unlearning */
2 repeat
3   | Optimize  $\theta_{\text{norm}}$  with Eq. (4) using downstream data  $\mathcal{D}$ ;
4 until training accuracy of classifier F reaches  $ACC_{\text{min}}$ ;
5 /* Filter Recovering */
6  $\mathbf{m}^{\kappa} = [1]^n$ ;
7 repeat
8   | Optimize  $\mathbf{m}^{\kappa}$  with Eq. (5) using trusted subset  $\mathcal{D}_{\text{sub}}$ ;
9   |  $\mathbf{m}^{\kappa} = \text{clip}_{[0,1]}(\mathbf{m}^{\kappa})$ ;
10 until training converged;
11 /* Channel Filtering */
12 Untrusted channels parameters  $\psi \leftarrow \theta[\mathbb{I}(\mathbf{m}^{\kappa} \leq \sigma)]$ ;
13 Trusted channels parameters  $\chi \leftarrow \theta[\mathbb{I}(\mathbf{m}^{\kappa} > \sigma)]$ ;
14 return  $\psi$  and  $\chi$ 
```

clean task, identifying these channels as trusted. In practice, filtering is applied to the original encoder with parameters θ_{pre} , based on the mask learned from $\hat{\theta}_{\text{pre}}$. The threshold σ is determined by the percentage of channels to preserve in each layer. After thresholding, we obtain both the untrusted channels ψ and trusted channels χ from the encoder.

4.3. Bootstrapping Learning

After completing the previous stages, we have established two trustworthy elements: the clean subset \mathcal{D}_{sub} and the trusted channels χ of the pre-trained encoder. We now explain how to leverage these trusted elements to bootstrap the learning of a clean model. The bootstrapping process involves gradually training the model with the clean pool $\mathcal{D}_{\text{clean}}$ and expanding $\mathcal{D}_{\text{clean}}$ until it reaches a sufficient proportion of the entire dataset. The bootstrapping procedure is detailed in the overall T-Core framework in Algorithm 4. **Optimization of Untrusted Channels.** The untrusted channels demonstrate a lower correlation with the downstream clean task. As a result, they misalign with the input-label mapping of the clean subset and harbor backdoor-related channels. Thus, we reinitialize the parameters of these untrusted channels and optimize them along with the classification head through gradient computation³ as follows:

$$\min_{\phi, \psi} \mathbb{E}_{(x,y) \in \mathcal{D}_{\text{clean}}} [\ell(f(\phi) \circ g(x; \psi \cup \chi), y)]. \quad (6)$$

This not only eliminates the backdoor from the encoder but turns them into the channels of the clean downstream task. Empirically, we identify 90% of the channels in transformation layers as clean in Encoder Channel Filtering, thus we merely optimize less than 10% of the encoder’s parameters in subsequent training.

3. In PyTorch, we define a list of parameters as leaf tensors, each tensor having the same shape as the untrusted channels of a respective layer, and assign the values of these leaf tensors to the untrusted channels accordingly.

Algorithm 4: Trusted Core Bootstrapping

Input: $g(\cdot; \theta_{\text{pre}})$ (Untrusted encoder);
 \mathcal{D} (Untrusted downstream dataset);
Output: Trusted encoder
Parameters: $WarmupEpoch$ (Epoch number)
 ρ (filtered out data size);

```
1 Randomly initialize  $\phi$ ;
2 Train  $\phi$  with  $\ell(f_{\phi} \circ g(\cdot | \theta_{\text{pre}}), \cdot)$  over  $\mathcal{D}_{\text{clean}}$ ;
3  $\mathcal{S} \leftarrow$  Topological Invariance Sifting ( $\theta_{\text{pre}}, \mathcal{D}$ );
4  $\mathcal{D}_{\text{sub}} \leftarrow$  Seed Expansion ( $\mathcal{S}, \mathcal{D}$ );
5  $\psi, \chi \leftarrow$  Encoder Channel Filtering ( $\theta_{\text{pre}}, \mathcal{D}$ );
6 Randomly initialize classifier head  $\phi$  and untrusted neurons  $\psi$ ;
7  $\mathcal{D}_{\text{clean}} \leftarrow \mathcal{D}_{\text{sub}}$ ;
8 for  $i = 1$  to  $Iter1$  do
9   | Train  $\phi$  and  $\psi$  using  $\mathcal{D}_{\text{clean}}$  for  $T$  epochs;
10  | Select  $\gamma_1\%$  samples with the smallest loss from each class
    | of  $\mathcal{D} \setminus \mathcal{D}_{\text{clean}}$  and add into  $\mathcal{D}_{\text{clean}}$ ;
11 for  $i = 1$  to  $Iter2$  do
12  | Train  $\phi$  and  $\psi$  using  $\mathcal{D}_{\text{clean}}$  for  $T$  epochs;
13  | Select  $\gamma_2\%$  samples with the smallest loss from entire
    | dataset of  $\mathcal{D} \setminus \mathcal{D}_{\text{clean}}$  and add into  $\mathcal{D}_{\text{clean}}$ ;
14 repeat
15  | Train  $\phi$  and  $\psi$  with  $\mathcal{D}_{\text{clean}}$ ;
16  |  $\phi' \leftarrow \phi, \psi' \leftarrow \psi$ ;
17  | Train  $\phi', \psi'$  using  $\mathcal{D} \setminus \mathcal{D}_{\text{clean}}$  for one epoch;
18  |  $Loss_1 \leftarrow \{\ell(f(\phi) \circ g(x; \phi \cup \chi), y) \mid (x, y) \in \mathcal{D} \setminus \mathcal{D}_{\text{clean}}\}$ ;
19  |  $Loss_2 \leftarrow \{\ell(f(\phi') \circ g(x; \phi' \cup \chi), y) \mid (x, y) \in \mathcal{D} \setminus \mathcal{D}_{\text{clean}}\}$ ;
20  | Select  $\gamma_3\%$  samples with the lowest values in
    |  $Loss_1 - Loss_2$  and add them into  $\mathcal{D}_{\text{clean}}$ ;
21 until  $|\mathcal{D}_{\text{clean}}|/|\mathcal{D}| \geq \rho$ ;
22 return  $\phi, \psi, \chi$ 
```

Clean Pool Expansion with Loss Guidance. This stage involves two processes to expand $\mathcal{D}_{\text{clean}}$ and train the model. We first initialize the clean pool $\mathcal{D}_{\text{clean}}$ with \mathcal{D}_{sub} (line 7). In the first process, we gradually expand $\mathcal{D}_{\text{clean}}$ based on the model’s prediction loss for each class in $\mathcal{D} \setminus \mathcal{D}_{\text{clean}}$. After training the model with $\mathcal{D}_{\text{clean}}$ for T epochs, we add samples with the lowest γ_1 loss in *each class* to maintain class balance (line 10). This process is repeated for $Iter1$ iterations. After this, the model achieves a certain accuracy on clean samples. In the second process, we incorporate samples with the lowest γ_2 loss from the entire set $\mathcal{D} \setminus \mathcal{D}_{\text{clean}}$ (line 13) for $Iter2$ iterations. This strategy helps avoid selecting poisoned samples from the target class, enabling the selection of more clean samples from non-target classes.

Clean Pool Expansion with Meta Guidance. After expanding with loss guidance, continually adding samples with the smallest prediction loss may include poisoned samples. To differentiate clean complex samples from poisoned ones, we adopt a meta-learning approach [42, 43] to augment $\mathcal{D}_{\text{clean}}$. We first train a temporary model on $\mathcal{D} \setminus \mathcal{D}_{\text{clean}}$, then select samples with the smallest loss reduction between the original and temporary models to add to $\mathcal{D}_{\text{clean}}$ (line 20). The rationale is that clean hard examples are more challenging to learn compared to easily inserted backdoor-poisoned examples. Even if some hard-to-learn backdoor samples are mistakenly selected, they require significantly more data and training to become effective. We halt this process once the ratio $|\mathcal{D}_{\text{clean}}|/|\mathcal{D}|$ reaches 0.9 (90% of the dataset).

5. Experiments

In this section, we first evaluate the effectiveness of our Trusted Core Bootstrapping framework by analyzing both its individual modules and the end-to-end framework as a whole. Since no prior work has considered the complex defense context we explore, and existing methodologies cannot serve as standalone solutions—even when adjusted as shown in Sec. 3—we compare each of our modules with existing approaches that aim to achieve the same goals to demonstrate the superiority and irreplaceability of ours. Finally, we assess the end-to-end defense performance of our T-Core framework. In summary, our main evaluation aims to answer the following research questions:

- **RQ1:** How effective is our Clean Data Sifting in obtaining a clean subset from various dataset poisoning types across downstream datasets under threats **Threat-2** and **Threat-3**?
- **RQ2:** How effective is our Encoder Channel Filtering in producing a purified encoder for transfer learning over a clean downstream dataset against **Threat-1**?
- **RQ3:** How effective is our Bootstrapping Learning in developing the clean classification head from a clean subset and clean encoder under dataset poisoning of **Threat-2**?
- **RQ4:** How effective is our end-to-end Trusted Core Bootstrapping framework in defending against any *unknown* backdoor threats **Threat-1**, **Threat-2**, **Threat-3**, or the scenario where both **Threat-1** and **Threat-2** exist?

In addition, we then evaluate the scalability of our T-Core bootstrapping framework across different dimensions to provide a holistic understanding of its practical deployment potential. Specifically, we extend our analysis beyond static threat scenarios and basic performance metrics to explore T-Core’s resilience against adaptive attacks, its sensitivity to configuration choices, adaptability to emerging transformer architecture, and efficiency in resource-constrained settings. These aspects evaluate whether a theoretically sound defense of T-Core is a viable solution for real-world applications. In general, our scalability evaluation aims to answer the following research question:

- **RQ5:** How scalable is T-Core in terms of defense effectiveness against adaptive adversaries, sensitivity to hyperparameter variations, adaptability to pre-trained Vision Transformer (ViT) models, and computational efficiency compared to other defense mechanisms?

5.1. Experimental Setups

Datasets. We use five image datasets: CIFAR-10 [44], GT-SRB [45], SVHN [46], STL-10 [47], and ImageNet [48] to construct clean and poisoned pre-trained encoders, as well as clean or poisoned downstream datasets, to evaluate our defense effectiveness. Additional details, including image dimensions, dataset sizes, and evaluation applicability, are provided in Appendix-B.1.

Models. We adopt ResNet18 as the default backbone for the pre-trained encoders, following [4, 6–9], which allows us to leverage their clean and poisoned checkpoints for evaluation.

For transfer learning, we use a simple MLP consisting of three linear layers for the classification head.

Encoder Poisoning Attacks. We consider five attacks that aim to inject backdoors into the pre-trained encoder: BadEncoder [4], DRUPE [7], SSLBackdoor [8], CTRL [6], and CorruptEncoder [9]. Details about how these encoder poisoning attacks are conducted and how we process their outcome encoders are provided in Appendix-B.2.

Dataset Poisoning Attack. We consider seven data poisoning backdoor attacks on the downstream dataset to evaluate the generalizability of our Clean Bootstrapping approach. These include: 1) the vanilla *dirty label* attacks: BadNets [1] and Blended [2], 2) the *clean label* attack: SIG [10], 3) the *sample-specific* attack: WaNet [13], 4) the *latent space adaptive attacks*: TaCT [11], Adap-Blend [12], and Adap-Patch [12]. In our primary evaluation, we default to the poison ratio as 20% of the target class. We further adjust the poison ratio to showcase the scalability of our seed data filtering approach in Sec. 5.2. We strictly follow the guidelines from each attack’s paper for implementation. Detailed configurations are outlined in the Appendix-B.3.

5.2. RQ1: Effectiveness of Clean Data Sifting

We evaluate the effectiveness of our Topological Invariance Sifting (TIS) across various poison ratios (0.1, 0.15, 0.2, 0.23, 0.3) of the target class in dataset poisoning attacks, addressing threats **Threat-2** and **Threat-3**. We maintain a sifting ratio α of 0.01, selecting 1% of the most credible samples from each class as seed data. For comparison, we utilize two poison detection methods, SPECTRE [16] and Spectral [19], alongside two state-of-the-art test-time backdoor input detection techniques, IBD-PSC [23] and SCALE-UP [24], which do not require an additional clean base set. We also evaluate META-SIFT [49], the only prior method for sifting clean data tested on backdoor poisoning under the SL setting, in our TL setting. To compete with our TIS, we configure the thresholds of each method to meet 1% selection, categorizing the remaining 99% as potentially untrustworthy. Additionally, we apply our Seed Expansion at a default poison ratio of 20% of the target class, using the corresponding seed data to continually expand and recording results when the expansion ratio $|\mathcal{D}_{\text{sub}}|/|\mathcal{D}|$ reaches 10%, 20%, 40%, and 50% to assess effectiveness.

We report the number of poisoned samples sifted out as clean seed data (False Positive cases) in Table 5, along with the *number of poisoned data* (NPD) and the *number of filtered data* (NFD) in the target class. Our TIS effectively identifies clean samples with minimal false positives for most backdoor attacks across all downstream datasets. In 212 out of 225 scenarios, covering various poison ratios, backdoor types, and datasets, TIS achieves a 100% precision, meaning all sifted samples are clean. For SVHN, false positive cases are slightly higher due to its noisy nature, where extraneous digits appear in images, resulting in fewer high-credibility samples.

Our TIS framework consistently outperforms existing methods, demonstrating robust reliability across all

TABLE 7: Comparison with SSL-Distillation under **Threat-1**. Results are shown for two configurations of our approach (Ours_{head} and Ours_{head+untrusted}), as well as for two configurations of SSL-Distillation (SSL-Distillation_{pre} and SSL-Distillation_{down}). Note that for CTRL, SSLBackdoor, and CorruptEncoder, the downstream dataset is the same as or a subset of the clean pre-training dataset.

| Encoder Poisoning | Pre-training Dataset | Downstream Dataset | No Defense | | SSL-Distillation _{pre} | | SSL-Distillation _{down} | | Ours _{head} | | Ours _{head+untrusted} | |
|-------------------|----------------------|--------------------|----------------|------------------|---------------------------------|------------------|----------------------------------|------------------|----------------------|------------------|--------------------------------|------------------|
| | | | ACC \uparrow | ASR \downarrow | ACC \uparrow | ASR \downarrow | ACC \uparrow | ASR \downarrow | ACC \uparrow | ASR \downarrow | ACC \uparrow | ASR \downarrow |
| BadEncoder | CIFAR-10 | STL-10 | 76.58 | 98.51 | 64.41 | 6.21 | 54.49 | 3.50 | 64.46 | 0.01 | 76.26 | 0.90 |
| | | GTSRB | 80.77 | 99.63 | 68.09 | 1.51 | 69.22 | 1.57 | 63.70 | 1.12 | 90.07 | 0.43 |
| | | SVHN | 65.35 | 97.56 | 66.68 | 10.38 | 61.75 | 32.32 | 64.64 | 3.23 | 92.20 | 2.82 |
| | STL-10 | CIFAR-10 | 70.57 | 98.93 | 50.27 | 13.09 | 51.77 | 12.27 | 61.09 | 1.36 | 66.59 | 2.92 |
| | | GTSRB | 70.83 | 98.99 | 50.89 | 2.08 | 49.98 | 1.45 | 53.14 | 0.12 | 88.22 | 1.98 |
| | | SVHN | 64.89 | 98.98 | 53.16 | 10.02 | 53.75 | 9.61 | 62.07 | 3.15 | 86.99 | 3.26 |
| DRUPE | CIFAR-10 | STL-10 | 71.85 | 97.72 | 60.91 | 5.36 | 53.59 | 3.54 | 68.38 | 1.89 | 72.06 | 2.04 |
| | | GTSRB | 76.39 | 98.10 | 60.85 | 0.17 | 61.76 | 0.48 | 63.49 | 2.26 | 90.81 | 0.02 |
| | | SVHN | 65.99 | 92.71 | 65.72 | 57.69 | 72.76 | 50.08 | 74.09 | 5.23 | 92.20 | 2.82 |
| | STL-10 | CIFAR-10 | 71.14 | 80.49 | 55.70 | 8.74 | 55.14 | 8.34 | 61.15 | 3.64 | 73.42 | 0.91 |
| | | GTSRB | 65.11 | 85.03 | 51.00 | 1.64 | 53.20 | 0.74 | 52.63 | 4.03 | 85.86 | 1.00 |
| | | SVHN | 58.43 | 96.28 | 46.22 | 23.19 | 50.44 | 4.99 | 60.10 | 5.11 | 91.20 | 3.99 |
| CTRL | STL-10 | STL-10 | 52.15 | 9.88 | - | - | 50.47 | 2.49 | 48.13 | 2.86 | 50.99 | 0.32 |
| | CIFAR-10 | CIFAR-10 | 75.31 | 44.90 | - | - | 62.35 | 1.65 | 50.83 | 2.54 | 62.42 | 0.74 |
| | GTSRB | GTSRB | 66.78 | 6.54 | - | - | 55.75 | 0.70 | 55.06 | 0.61 | 84.45 | 0.22 |
| SSLBackdoor | ImageNet | ImageNet-10 | 82.85 | 36.48 | - | - | 70.47 | 3.12 | 65.88 | 2.94 | 85.76 | 2.73 |
| CorruptEncoder | ImageNet | ImageNet-10 | 82.35 | 58.46 | - | - | 69.55 | 4.67 | 63.59 | 2.61 | 84.53 | 0.36 |

on downstream tasks by training a classification head on each purified or distilled encoder. Notably, SSL-Distillation requires complete fine-tuning of the backdoor encoder and building the student encoder from scratch, whereas our Encoder Channel Filtering only necessitates training less than 1% of the total parameters. During subsequent training, we apply two configurations: one trains only the classification head after pruning untrusted channels (Ours_{head}), and the other trains both the classification head and the initialized untrusted channels (Ours_{head+untrusted}).

The results from Table 7 show that our method generally trumps SSL-Distillation across all downstream tasks and achieves a very low ASR. Notably, SSL-Distillation performs poorly on the noisy downstream dataset of SVHN, supporting findings in [27] that SVHN poses increased security risks due to its exploitable noisy features under backdoor attacks. In contrast, our encoder filtering method performs significantly better on SVHN. We believe this is because our approach progressively selects channels beneficial for the clean classification task that aims to distinguish features apart. On the other hand, SSL-Distillation indiscriminately distills feature embeddings, especially from the already noisy samples in SSL-Distillation_{down}, which allows the backdoor to traverse through distillation. Besides, our method yields an increased ACC for many cases due to the additional optimization of the original untrusted channels.

5.4. RQ3: Effectiveness of Bootstrapping Learning

We compare our Bootstrapping Learning with backdoor-removal methods that utilize a clean subset under **Threat-2**. Specifically, we consider two state-of-the-art methods: the trigger synthesis-based I-BAU [25] and the fine-tuning-based FT-SAM [26]. We apply them to the post-attack classification head. We also limit the training of our Bootstrapping Learning to the classification head without the untrusted channels. In this experiment, we vary the clean data ratio to assess its impact on subset size dependence.

TABLE 8: Comparison of our method with FT-SAM and I-BAU under **Threat-2**, using poisoned GTSRB as downstream dataset and a CIFAR-10 encoder, with a clean subset at varying ratios of the original clean GTSRB for defense.

| Clean Ratio | Methods | BadNets | | Blended | | SIG | | WaNet | | TaCT | | Adap-Blend | | Adap-Patch | |
|-------------|---------|----------------|------------------|----------------|------------------|----------------|------------------|----------------|------------------|----------------|------------------|----------------|------------------|----------------|------------------|
| | | ACC \uparrow | ASR \downarrow |
| No Defense | | 81.79 | 95.02 | 81.30 | 90.39 | 81.90 | 74.37 | 80.74 | 8.81 | 81.95 | 89.20 | 80.85 | 69.73 | 78.54 | 28.20 |
| 0.1 | FT-SAM | 21.88 | 20.61 | 22.24 | 4.50 | 22.00 | 51.68 | 19.03 | 0.69 | 22.82 | 15.20 | 22.02 | 2.33 | 21.54 | 1.13 |
| | I-BAU | 16.98 | 26.36 | 17.36 | 0.74 | 17.03 | 46.20 | 13.86 | 1.40 | 18.39 | 28.80 | 17.22 | 0.35 | 15.38 | 0.59 |
| | Ours | 78.65 | 4.13 | 78.47 | 4.15 | 77.91 | 3.70 | 79.73 | 3.59 | 78.99 | 7.07 | 78.24 | 3.11 | 78.08 | 3.12 |
| 0.3 | FT-SAM | 23.82 | 19.41 | 24.70 | 10.87 | 23.91 | 39.52 | 22.27 | 0.97 | 24.96 | 2.13 | 23.35 | 3.40 | 23.89 | 0.92 |
| | I-BAU | 16.60 | 21.17 | 17.82 | 0.75 | 17.66 | 42.94 | 13.15 | 1.41 | 16.36 | 18.67 | 16.68 | 0.13 | 14.73 | 0.08 |
| | Ours | 80.78 | 2.07 | 80.24 | 1.04 | 79.82 | 0.06 | 80.55 | 1.40 | 80.31 | 2.00 | 79.78 | 1.93 | 79.70 | 1.55 |
| 0.5 | FT-SAM | 25.69 | 19.31 | 27.58 | 10.70 | 26.56 | 31.04 | 26.19 | 0.94 | 27.71 | 3.73 | 26.07 | 3.51 | 26.07 | 0.97 |
| | I-BAU | 17.50 | 29.51 | 15.60 | 0.48 | 13.98 | 37.63 | 12.25 | 1.78 | 17.51 | 30.67 | 16.23 | 0.13 | 15.04 | 0.05 |
| | Ours | 82.12 | 0.04 | 80.95 | 0.13 | 80.68 | 0.00 | 81.08 | 0.79 | 80.99 | 0.53 | 80.34 | 0.64 | 80.82 | 0.47 |

Table 8 demonstrate that these methods cannot produce a satisfying result under any clean ratio, while our Bootstrapping Learning works exceptionally well even when the subset is small. These results again reveal that the backdoor and clean features are tangled in the hidden representation space when the training is restricted to the classification head, as previously illustrated in Sec. 3.3. In this setting, the clean subset also cannot help to remove the backdoor without sacrificing the clean accuracy. In that sense, it also indicates the necessity of the proactive mindset behind Bootstrapping Learning, whose design aims to explicitly train on clean elements while keeping the clean and poison elements separated in the whole training process.

5.5. RQ4: Effectiveness of T-Core Bootstrapping

As we previously discussed, in the defense context of transfer learning, the defender may not know what kind of backdoor risks it is facing in general, so a successful defense should be able to deal with all kinds of backdoor threats. Thus, we evaluate the end-to-end procedure of our proposed T-Core Bootstrapping framework in defending all four scenarios: the encoder poisoning alone of **Threat-1**, the dataset poisoning alone of **Threat-2**, the adaptive poisoning with both the encoder and dataset with the same backdoor trigger of **Threat-3**, as well as the independent encoder poisoning and dataset poisoning with the different backdoor triggers of **Threat-1** and **Threat-2**.

Overall, T-Core effectively defends against all considered backdoor threats, as shown in Tables 9 to 11. Specifically, the attack success rates for **Threat-1**, **Threat-2**, and **Threat-3** are all below 10%. Additionally, T-Core improves accuracy (ACC) in most cases, due to the optimization of the original untrusted channels. However, ACC often decreases for STL-10 due to its limited training images. T-Core halts training at 4,500 samples (90%), resulting in incomplete training. For cases where two independent attackers of **Threat-1** and **Threat-2** both exist, the results in Table 9 indicate that the attack potency of encoder poisoning is consistently greater than that of downstream poisoning. For downstream poisoning, Adap-Patch, Adap-Blend, and WaNet also produce limited attack potency as they inject different trigger patterns for different poisoned samples. Nevertheless, T-Core can ensure low ASRs of both encoder poisoning and dataset poisoning, regardless of the attack’s original strength.

TABLE 9: Performance of our end-to-end defense framework under scenarios where both **Threat-1** and **Threat-2** exist. In this setting, transfer learning is subjected to both encoder poisoning and downstream poisoning attacks concurrently, each with a different backdoor trigger. We report the accuracy, the *attack success rate* of the *encoder* poisoning attack (ASR-E), and the *attack success rate* of the *dataset* poisoning attack (ASR-D) for the classifier, both with and without our defense.

| Encoder Poisoning | Pre-training Dataset | | Downstream Dataset | Dataset Poisoning | BadNets | | | Blended | | | SIG | | | WaNet | | | TaCT | | | Adap-Blend | | | Adap-Patch | | |
|-------------------|----------------------|------------|--------------------|-------------------|---------|-------|-------|---------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|------------|-------|-------|------------|-------|--|
| | ACC | ASR | | | ACC | ASR | ACC | ASR | ACC | ASR | ACC | ASR | ACC | ASR | ACC | ASR | ACC | ASR | ACC | ASR | ACC | ASR | ACC | ASR | |
| BadEncoder | CIFAR-10 | STL-10 | No Defense | 76.30 | 99.51 | 91.50 | 76.28 | 99.96 | 60.10 | 76.51 | 99.99 | 59.36 | 76.43 | 99.56 | 4.51 | 75.71 | 99.90 | 62.75 | 76.19 | 96.54 | 10.14 | 76.93 | 99.99 | 1.57 | |
| | | | Ours | 67.75 | 4.67 | 1.00 | 67.04 | 6.85 | 6.68 | 53.10 | 3.88 | 2.53 | 67.54 | 5.11 | 1.82 | 67.46 | 5.72 | 4.25 | 68.75 | 6.65 | 1.40 | 68.28 | 6.03 | 6.22 | |
| | | GTSRB | No Defense | 72.60 | 99.24 | 93.75 | 73.22 | 99.77 | 86.36 | 73.16 | 99.15 | 74.81 | 78.17 | 99.94 | 6.09 | 73.86 | 99.20 | 91.73 | 72.98 | 95.95 | 65.60 | 72.22 | 99.69 | 28.43 | |
| | | | Ours | 90.54 | 0.01 | 1.38 | 88.27 | 0.31 | 5.05 | 91.69 | 0.00 | 0.98 | 91.88 | 0.04 | 0.66 | 92.60 | 0.80 | 0.00 | 87.79 | 0.00 | 3.30 | 93.90 | 0.27 | 0.29 | |
| | | SVHN | No Defense | 68.47 | 98.80 | 99.27 | 67.98 | 98.95 | 98.11 | 68.19 | 98.70 | 96.63 | 67.99 | 98.78 | 11.86 | 68.19 | 98.80 | 94.12 | 68.07 | 98.81 | 90.81 | 68.28 | 97.90 | 71.75 | |
| | | | Ours | 92.19 | 4.29 | 3.79 | 92.19 | 4.29 | 0.10 | 92.80 | 4.80 | 0.65 | 90.20 | 7.94 | 2.76 | 91.51 | 2.49 | 0.75 | 90.30 | 4.23 | 0.14 | 92.72 | 4.86 | 0.07 | |
| | STL-10 | CIFAR-10 | No Defense | 69.56 | 97.88 | 78.00 | 70.33 | 98.39 | 71.98 | 69.72 | 99.83 | 77.42 | 69.94 | 99.82 | 9.12 | 69.66 | 99.66 | 70.00 | 69.84 | 99.77 | 16.28 | 70.03 | 99.76 | 5.78 | |
| | | | Ours | 63.27 | 5.76 | 4.76 | 62.73 | 6.28 | 4.97 | 68.42 | 8.29 | 3.64 | 62.63 | 6.61 | 4.47 | 65.47 | 6.36 | 0.00 | 64.38 | 7.71 | 2.03 | 63.05 | 6.08 | 0.13 | |
| | | GTSRB | No Defense | 70.67 | 97.52 | 83.43 | 69.59 | 98.77 | 82.33 | 70.86 | 99.19 | 74.56 | 69.63 | 99.80 | 4.33 | 68.33 | 98.05 | 81.07 | 68.56 | 99.10 | 54.45 | 69.58 | 98.95 | 12.30 | |
| | | | Ours | 85.65 | 0.11 | 5.45 | 86.03 | 0.70 | 0.87 | 85.18 | 1.73 | 0.24 | 85.27 | 0.22 | 4.39 | 86.03 | 0.05 | 1.06 | 85.58 | 1.10 | 5.13 | 87.05 | 1.80 | 1.52 | |
| | | SVHN | No Defense | 67.44 | 85.95 | 98.85 | 66.29 | 85.93 | 98.93 | 67.45 | 88.96 | 93.92 | 64.88 | 84.07 | 11.91 | 67.78 | 87.69 | 94.53 | 67.60 | 81.29 | 89.94 | 66.77 | 80.30 | 26.85 | |
| | | | Ours | 83.90 | 4.30 | 10.10 | 86.63 | 3.72 | 5.32 | 85.96 | 9.18 | 2.55 | 88.96 | 5.10 | 1.01 | 86.34 | 3.15 | 0.31 | 86.40 | 4.87 | 2.09 | 86.92 | 6.15 | 4.50 | |
| DRUPE | CIFAR-10 | STL-10 | No Defense | 71.94 | 99.43 | 75.22 | 71.09 | 98.00 | 53.97 | 72.49 | 93.63 | 35.50 | 72.08 | 90.18 | 10.14 | 71.78 | 97.54 | 49.75 | 71.34 | 99.39 | 11.42 | 71.63 | 98.35 | 1.89 | |
| | | | Ours | 63.16 | 14.90 | 10.92 | 68.30 | 10.89 | 5.89 | 64.34 | 7.49 | 0.49 | 64.59 | 6.38 | 4.29 | 63.63 | 11.24 | 13.00 | 64.74 | 7.92 | 2.67 | 65.00 | 7.39 | 2.96 | |
| | | GTSRB | No Defense | 74.35 | 73.36 | 94.19 | 74.57 | 72.99 | 87.63 | 74.95 | 74.70 | 69.57 | 74.48 | 73.02 | 6.58 | 74.67 | 72.91 | 87.07 | 73.95 | 73.01 | 61.30 | 73.76 | 72.97 | 14.79 | |
| | | | Ours | 87.98 | 7.05 | 3.16 | 90.17 | 7.23 | 6.66 | 88.16 | 3.18 | 0.74 | 89.14 | 3.61 | 0.47 | 89.93 | 5.82 | 6.82 | 89.14 | 5.05 | 7.63 | 89.87 | 3.10 | 1.85 | |
| | | SVHN | No Defense | 71.35 | 75.53 | 99.45 | 71.37 | 75.74 | 97.60 | 71.21 | 75.81 | 94.45 | 71.04 | 76.95 | 11.60 | 71.31 | 72.91 | 96.35 | 71.26 | 77.03 | 85.17 | 71.09 | 96.76 | 51.23 | |
| | | | Ours | 89.54 | 9.64 | 6.78 | 88.73 | 6.92 | 4.90 | 89.02 | 9.88 | 4.32 | 87.19 | 6.66 | 3.66 | 92.34 | 3.60 | 2.77 | 89.20 | 5.10 | 1.01 | 89.70 | 5.04 | 2.97 | |
| | STL-10 | CIFAR-10 | No Defense | 70.26 | 78.54 | 74.24 | 70.71 | 77.58 | 74.19 | 70.83 | 79.10 | 69.62 | 70.87 | 78.66 | 9.27 | 70.62 | 78.55 | 69.00 | 70.81 | 78.63 | 14.13 | 71.15 | 78.63 | 4.93 | |
| | | | Ours | 64.74 | 6.87 | 7.43 | 63.46 | 7.53 | 7.69 | 67.31 | 4.94 | 1.91 | 66.18 | 4.02 | 1.73 | 66.28 | 5.49 | 0.10 | 62.63 | 4.96 | 3.40 | 63.56 | 3.31 | 6.31 | |
| | | GTSRB | No Defense | 63.40 | 78.25 | 90.50 | 63.71 | 84.92 | 88.70 | 64.29 | 85.40 | 74.55 | 63.99 | 78.12 | 6.09 | 63.47 | 86.80 | 78.54 | 61.18 | 80.32 | 67.40 | 62.00 | 79.83 | 18.46 | |
| | | | Ours | 86.10 | 0.21 | 3.94 | 87.08 | 1.42 | 5.85 | 86.44 | 2.82 | 0.03 | 84.47 | 1.00 | 3.18 | 82.18 | 0.25 | 5.45 | 81.90 | 1.61 | 2.95 | 81.32 | 0.62 | 7.58 | |
| | | SVHN | No Defense | 59.12 | 94.66 | 96.56 | 59.77 | 97.48 | 97.43 | 58.03 | 92.94 | 91.53 | 59.77 | 95.08 | 15.17 | 59.47 | 97.46 | 92.33 | 60.02 | 98.69 | 84.58 | 59.74 | 96.81 | 16.52 | |
| | | | Ours | 82.13 | 5.95 | 6.25 | 83.22 | 4.03 | 4.56 | 83.75 | 9.64 | 2.77 | 82.76 | 2.45 | 3.59 | 83.85 | 2.93 | 0.98 | 81.13 | 9.01 | 5.10 | 83.17 | 3.05 | 1.65 | |
| CTRL | STL-10 | No Defense | 51.80 | 9.97 | 56.39 | 53.20 | 10.15 | 17.67 | 53.14 | 8.99 | 18.96 | 53.43 | 10.19 | 5.75 | 52.06 | 10.83 | 40.25 | 53.28 | 8.79 | 4.99 | 52.88 | 8.65 | 4.64 | | |
| | | Ours | 48.81 | 1.79 | 1.21 | 49.80 | 2.23 | 2.51 | 45.56 | 3.57 | 0.99 | 49.68 | 0.24 | 2.44 | 50.43 | 2.58 | 3.51 | 49.63 | 2.44 | 5.93 | 48.44 | 1.03 | 1.96 | | |
| | CIFAR-10 | No Defense | 75.33 | 48.93 | 93.29 | 75.22 | 44.92 | 50.59 | 76.19 | 38.54 | 52.56 | 74.31 | 41.14 | 13.98 | 75.80 | 48.87 | 84.00 | 75.08 | 49.41 | 16.93 | 75.93 | 46.74 | 14.90 | | |
| | | Ours | 61.78 | 0.94 | 0.10 | 62.72 | 2.87 | 6.88 | 61.94 | 0.59 | 0.00 | 62.32 | 3.60 | 3.44 | 62.79 | 0.57 | 0.70 | 63.51 | 0.78 | 8.84 | 58.80 | 0.63 | 6.72 | | |
| | GTSRB | No Defense | 64.68 | 4.60 | 73.81 | 67.16 | 3.05 | 69.37 | 65.83 | 4.96 | 60.30 | 66.00 | 3.76 | 4.67 | 65.86 | 5.82 | 62.40 | 66.94 | 9.04 | 52.71 | 65.13 | 4.84 | 21.36 | | |
| | | Ours | 85.30 | 0.09 | 2.79 | 88.28 | 0.19 | 0.14 | 87.05 | 0.00 | 0.07 | 87.82 | 0.08 | 0.34 | 88.37 | 0.15 | 0.40 | 88.13 | 1.77 | 4.97 | 87.73 | 0.11 | 0.03 | | |
| SSLBackdoor | ImageNet | No Defense | 84.47 | 19.15 | 84.79 | 83.88 | 24.97 | 32.85 | 82.85 | 45.88 | 60.00 | 81.18 | 24.61 | 2.48 | 83.12 | 40.18 | 93.00 | 82.47 | 26.36 | 26.42 | 82.94 | 33.70 | 2.24 | | |
| | | Ours | 80.65 | 2.61 | 3.15 | 80.82 | 1.21 | 6.55 | 79.94 | 3.21 | 4.97 | 78.00 | 3.21 | 2.24 | 83.47 | 1.88 | 3.21 | 83.18 | 0.67 | 3.82 | 83.47 | 1.82 | 2.61 | | |
| CorruptEncoder | ImageNet | No Defense | 84.41 | 34.91 | 82.73 | 83.76 | 57.45 | 26.73 | 82.53 | 54.18 | 61.82 | 81.41 | 44.06 | 3.94 | 83.76 | 60.12 | 91.00 | 80.06 | 53.09 | 15.03 | 83.47 | 57.27 | 3.45 | | |
| | | Ours | 82.35 | 1.33 | 4.55 | 82.35 | 2.00 | 4.61 | 82.06 | 1.33 | 5.88 | 80.00 | 1.58 | 3.15 | 81.65 | 1.21 | 6.50 | 82.53 | 0.97 | 6.36 | 84.06 | 1.82 | 1.76 | | |

TABLE 10: Performance of our entire end-to-end defense framework under **Threat-2**.

| Dataset | Dataset Poisoning | BadNets | | Blended | | SIG | | WaNet | | TaCT | | Adap-Blend | | Adap-Patch | |
|-------------|-------------------|---------|-------|---------|-------|-------|-------|-------|-------|-------|-------|------------|-------|------------|-------|
| | | ACC | ASR | ACC | ASR | ACC | ASR | ACC | ASR | ACC | ASR | ACC | ASR | ACC | ASR |
| STL-10 | No Defense | 75.64 | 90.24 | 75.65 | 50.35 | 76.51 | 59.97 | 76.21 | 4.76 | 75.19 | 64.13 | 75.75 | 9.04 | 76.43 | 1.92 |
| | Ours | 64.08 | 2.15 | 65.59 | 1.60 | 62.85 | 6.00 | 64.55 | 1.60 | 66.26 | 1.00 | 65.93 | 3.24 | 62.55 | 1.08 |
| CIFAR-10 | No Defense | 85.04 | 92.71 | 84.84 | 89.12 | 84.72 | 89.10 | 84.40 | 9.11 | 84.28 | 82.60 | 83.39 | 34.34 | 84.16 | 5.66 |
| | Ours | 87.38 | 3.48 | 87.35 | 5.90 | 87.31 | 2.54 | 87.58 | 0.23 | 89.04 | 0.10 | 87.31 | 2.54 | 87.38 | 3.48 |
| GTSRB | No Defense | 81.79 | 95.02 | 81.30 | 90.39 | 81.90 | 74.37 | 80.74 | 8.81 | 81.95 | 89.20 | 80.85 | 69.73 | 78.54 | 28.20 |
| | Ours | 92.03 | 1.31 | 91.37 | 3.04 | 94.13 | 0.38 | 91.10 | 1.31 | 91.82 | 1.87 | 90.87 | 0.62 | 92.25 | 1.09 |
| SVHN | No Defense | 59.80 | 99.42 | 60.11 | 98.30 | 59.83 | 97.58 | 59.65 | 15.77 | 59.91 | 91.90 | 59.84 | 89.90 | 59.87 | 70.86 |
| | Ours | 91.19 | 4.14 | 90.88 | 6.82 | 91.09 | 3.22 | 90.11 | 1.45 | 91.25 | 2.92 | 90.22 | 1.31 | 90.95 | 1.23 |
| ImageNet-10 | No Defense | 85.06 | 92.85 | 85.00 | 40.42 | 86.29 | 55.33 | 85.71 | 3.33 | 85.88 | 95.00 | 86.35 | 24.06 | 85.71 | 6.48 |
| | Ours | 80.46 | 3.86 | 81.65 | 2.42 | 82.00 | 2.85 | 83.71 | 0.94 | 84.53 | 3.33 | 80.24 | 1.94 | 81.71 | 2.48 |

5.6. RQ5: Scalability of T-Core Bootstrapping

5.6.1. Resilience against Adaptive Attack. T-Core relies on the initial topological invariance sifting (TIS) of high-credible samples, which are carried out based on the assumptions of both majority rule and consistency rule. To launch an effective adaptive backdoor poisoning, we exploit the seed data sifted by TIS by conducting a layerwise adversarial attack to construct backdoor triggers. Specifically, we generate a universal adversarial perturbation (UAP) δ on input samples that minimizes the activation distance between perturbed inputs and target class samples across multiple layers as follows:

$$\min_{\delta} \frac{1}{|\mathcal{D}| \times L} \sum_{x_1 \in \mathcal{D}} \sum_{l=N-L-1}^{N-1} \|h^l(x_1 + \delta) - \frac{1}{|\mathcal{S}_t|} \sum_{x_2 \in \mathcal{S}_t} h^l(x_2)\|,$$

where \mathcal{S}_t is the seed data of the target class (sifted from clean data \mathcal{D}_t by TIS). TIS is conducted on the last L layers

TABLE 11: Performance of our entire end-to-end defense framework under **Threat-1** and **Threat-3**.

| Encoder Poisoning | Threat Type | | Threat-1 | | Threat-3 | | |
|-------------------|----------------------|--------------------|------------|-------|----------|-------|--------|
| | Pre-training Dataset | Downstream Dataset | Methods | | ACC | ASR | |
| | | | ACC | ASR | | | |
| BadEncoder | CIFAR-10 | STL-10 | No Defense | 76.58 | 98.51 | 76.79 | 100.00 |
| | | | Ours | 55.23 | 4.29 | 66.24 | 1.40 |
| | | GTSRB | No Defense | 80.77 | 99.63 | 78.45 | 99.97 |
| | | | Ours | 90.86 | 3.90 | 91.92 | 0.01 |
| | | SVHN | No Defense | 65.35 | 97.56 | 67.93 | 99.44 |
| | | | Ours | 85.93 | 3.76 | 92.52 | 0.65 |
| | STL-10 | CIFAR-10 | No Defense | 70.57 | 98.93 | 69.66 | 99.96 |
| | | | Ours | 60.65 | 5.22 | 62.90 | 6.80 |
| | | GTSRB | No Defense | 70.83 | 98.99 | 66.67 | 99.83 |
| | | | Ours | 87.08 | 4.93 | 90.43 | 0.76 |
| | | SVHN | No Defense | 64.89 | | | |

TABLE 12: Performance of seed-sifting module TIS and the entire framework of T-Core against **adaptive attack** on CIFAR-10. We test out the perturbation budget as 4, 8, 12, and 16 of the infinite norm, respectively. ‘Poi Num’ denotes the number of poisons are deemed as clean seed data by TIS.

| Perturbation Range | 4 | | | 8 | | | 12 | | | 16 | | |
|--------------------|-------|-------|---------|-------|-------|---------|-------|-------|---------|-------|-------|---------|
| | ACC | ASR | Poi Num |
| No Defense | 83.37 | 83.71 | - | 83.65 | 96.08 | - | 83.41 | 96.51 | - | 84.04 | 96.75 | - |
| Ours | 81.79 | 5.65 | 0 | 82.97 | 13.42 | 1 | 81.67 | 21.49 | 4 | 81.42 | 92.52 | 38 |

TABLE 13: Ablation on the hyper-parameters γ_1 , γ_2 of bootstrapping learning on CIFAR-10. γ_1 is the selection rate from *each class*, γ_2 is the selection rate from *entire dataset*.

| SIG | | | | | | | | | | |
|----------------|-------------------------|------------------|----------------|------------------|---------------------------|------------------|----------------|------------------|--|--|
| No Defense | ACC \uparrow 84.72 | | | | ASR \downarrow 89.1 | | | | | |
| γ_2 (%) | 2 | | 5 | | 7 | | 10 | | | |
| γ_1 (%) | ACC \uparrow | ASR \downarrow | ACC \uparrow | ASR \downarrow | ACC \uparrow | ASR \downarrow | ACC \uparrow | ASR \downarrow | | |
| 1 | 88.24 | 2.24 | 88.36 | 2.34 | 87.79 | 4.53 | 87.22 | 7.92 | | |
| 2 | 89.09 | 2.76 | 87.31 | 2.54 | 86.91 | 4.27 | 87.13 | 8.44 | | |
| 5 | 87.98 | 4.49 | 86.16 | 9.59 | 83.55 | 21.73 | 81.84 | 40.57 | | |
| Blended | | | | | | | | | | |
| No Defense | ACC \uparrow 84.84 | | | | ASR \downarrow 89.12 | | | | | |
| γ_2 (%) | 2 | | 5 | | 7 | | 10 | | | |
| γ_1 (%) | ACC \uparrow | ASR \downarrow | ACC \uparrow | ASR \downarrow | ACC \uparrow | ASR \downarrow | ACC \uparrow | ASR \downarrow | | |
| 1 | 88.64 | 4.48 | 87.38 | 4.93 | 88.01 | 6.37 | 86.93 | 13.75 | | |
| 2 | 88.09 | 4.43 | 87.35 | 5.9 | 87.14 | 7.37 | 86.26 | 16.07 | | |
| 5 | 86.63 | 8.46 | 85.31 | 12.74 | 83.21 | 26.07 | 82.55 | 44.15 | | |

TABLE 14: Ablation on bootstrapping rate ρ in Bootstrapping Learning, *i.e.*, when we choose to halt the bootstrapping learning, on CIFAR-10, ‘Poi Num’ denotes the final number of poison samples that are included.

| Bootstrapping Rate(%) | SIG | | | Blended | | |
|-----------------------|----------------|------------------|---------|----------------|------------------|---------|
| | ACC \uparrow | ASR \downarrow | Poi Num | ACC \uparrow | ASR \downarrow | Poi Num |
| No Defense | 84.72 | 89.10 | - | 84.84 | 89.12 | - |
| 97 | 85.13 | 12.48 | 65 | 85.54 | 71.54 | 568 |
| 95 | 87.62 | 3.11 | 9 | 86.62 | 79.38 | 88 |
| 90 | 87.31 | 2.54 | 0 | 87.35 | 5.90 | 13 |
| 85 | 81.09 | 1.87 | 0 | 82.26 | 3.71 | 5 |
| 80 | 79.41 | 2.06 | 0 | 80.63 | 2.31 | 0 |
| 70 | 76.22 | 1.63 | 0 | 77.10 | 1.56 | 0 |

The optimized adversarial perturbation is then used as the trigger for backdoor injection. Such an adaptive adversary is provided with total knowledge of our TIS, as well as access to the pre-trained encode and the downstream dataset.

The experiments reveal a clear trade-off between the adversarial budget and attack effectiveness. As demonstrated in Table 12, even under this adaptive attack, a significant adversarial perturbation budget is required to completely undermine our defense, specifically, an l_∞ norm of 16 (where each pixel value is expected to change by 16). In contrast, TIS remains effective under smaller adversarial budgets of l_∞ norm from 4 to 12, only minimum poisoned samples (0 to 4) are mistakenly selected as clean seed data. The ASR rises as the adversarial budget increases, though, T-Core does demonstrate resilience against moderate perturbations.

We conducted ablations on the selection rate for each class γ_1 , the selection rate for the entire dataset γ_2 (Table 13) and the stopping criterion ρ for Clean Bootstrapping

(Table 14) using two standard dataset poisoning attacks, SIG and Blended. Results in Table 13 show that selecting from each class requires more caution than selecting from the entire dataset. A smaller selection rate (more cautious approach) leads to a lower ASR. On the other hand, results in Table 14 demonstrate that for the Blended attack, when ρ exceeds 85%, a few poison samples are already included for training, increasing the ASR while the ACC is not high enough. For the SIG attack, a ρ of 90% or lower effectively eliminates poison samples, achieving a low ASR while maintaining a high ACC. A ρ above 90% has made the ASR significantly high. Thus, the choice of ρ should be carefully tuned, considering different dataset poisoning.

The concept of channels in Encoder Channel Filtering, originally designed for Convolutional Neural Networks (CNNs), does not directly translate to Vision Transformers (ViTs). However, we can adapt it by reinterpreting each linear layer in the transformer as having multiple channels. For example, a linear layer with dimensions 1536×512 can be viewed as having 1536 channels, where each channel generates a 512-dimensional feature vector (see the code implementation in Fig. 7). This enables the application of a learnable mask to the channels of the linear layer, mirroring the approach used in CNNs. Unlike CNNs, which often rely on batch normalization, ViTs utilize layer normalization. Thus, we leverage layer normalization for the unlearning process. Results in Table 15 demonstrate that T-Core effectively defends against Threat-1, Threat-2, and Threat-3, achieving low ASRs while maintaining accuracy close to the original undefended models.

We present a detailed comparison of the time expenses of various seed-sifting methods in Table 16. Our Topological Invariance Sifting (TIS) module significantly outperforms state-of-the-art data-sifting approaches, such as MetaSift and SPECTRE, in both efficiency and effectiveness, requiring substantially less computation time. In contrast, faster alternatives like IBD-PSC, SCALE-UP, and Spectral exhibit markedly inferior performance in data sifting. In Table 17, we report the time expenses and memory usage for each module of T-Core. The total runtime is 1102.3 seconds (under 20 minutes), which we consider highly practical for a defense mechanism that delivers a clean, backdoor-free classifier. Additionally, memory efficiency is achieved by targeting a subset of parameters, with peak GPU memory

TABLE 15: Performance of T-Core on Vision Transformer against different types of backdoor threats of **Threat-1** (Dataset Poisoning), **Threat-2** (Encoder Poisoning), and **Threat-3** (Adaptive Poisoning).

| Threat Type | Threat-2 | | | | | | Threat-1 | | | | Threat-3 | | | |
|-------------|-----------------|------------------|----------------|------------------|----------------|------------------|-----------------|------------------|----------------|------------------|-----------------|------------------|----------------|------------------|
| | BadNets | | Blended | | SIG | | BadEncoder | | DRUPE | | BadEncoder | | DRUPE | |
| | ACC \uparrow | ASR \downarrow | ACC \uparrow | ASR \downarrow | ACC \uparrow | ASR \downarrow | ACC \uparrow | ASR \downarrow | ACC \uparrow | ASR \downarrow | ACC \uparrow | ASR \downarrow | ACC \uparrow | ASR \downarrow |
| No Defense | 72.17 | 87.57 | 72.09 | 82.53 | 72.03 | 62.99 | 60.33 | 98.79 | 66.22 | 99.83 | 58.02 | 99.99 | 66.20 | 99.99 |
| Ours | 70.94 | 4.35 | 70.91 | 3.55 | 70.94 | 3.15 | 56.46 | 0.26 | 64.95 | 4.50 | 55.55 | 2.50 | 65.25 | 5.57 |

TABLE 16: **Time expenses** (seconds) in comparison of different seed-sifting methods, on the CIFAR-10 dataset at varying dataset sizes per class. According to Table 5, the faster IBD-PSC, SCALE-UP, and Spectral generally perform poorly.

| Data Size per Class | 100 | 500 | 1000 | 2500 | 5000 |
|---------------------|-------------|-------------|-------------|-------------|--------------|
| IBD-PSC | 3.83 | 14.94 | 26.81 | 64.81 | 139.69 |
| SCALE-UP | 0.69 | 3.02 | 6.00 | 14.81 | 29.51 |
| Spectral | 0.32 | 1.85 | 3.07 | 7.01 | 14.13 |
| SPECTRE | 14.53 | 74.23 | 117.95 | 231.71 | 456.57 |
| META-SIFT | 47.99 | 83.68 | 128.34 | 267.09 | 489.26 |
| Ours | 7.01 | 26.05 | 46.68 | 129.29 | 294.38 |

TABLE 17: **Time expenses and memory usage** of T-Core. We report the total time and memory expenses for T-Core, along with the breakdown for each module: Topological Invariance Sifting (TIS), Seed Expansion (SE), Encoder Channel Filtering (ECF), and Bootstrapping Learning (BL). Experiments are conducted on the CIFAR-10 dataset (50,000 samples) using a ResNet18 encoder with a batch size of 128.

| Stages | TIS | SE | ECF | BL | Total Expenses |
|-----------|-------|------|-------|-------|----------------|
| Time (s) | 294.4 | 60.1 | 105.6 | 642.2 | 1102.3 |
| VRAM (MB) | 5312 | 3338 | 3484 | 4794 | 5312 |

usage remaining below 5 GB. These demonstrate the scalability of T-Core, making it a viable solution for real-world applications where computational resources are limited.

6. Limitation and Future Work

Our T-Core framework is designed to defend against unknown backdoor threats in blind transfer learning scenarios, where neither the attack pattern nor the integrity of the pre-trained model or data is known. By adopting a proactive defense strategy—identifying and amplifying clean elements—we mitigate potential backdoors without prior knowledge of the threat.

However, this approach relies on sifting a number of clean samples to bootstrap a reliable trust core. A key limitation arises when the training data is scarce, making it difficult to isolate enough clean elements for robust initialization. For instance, STL-10 (with only 5,000 samples) exhibits a more noticeable accuracy drop compared to larger

datasets. Future work will explore better trade-offs between security and utility in low-data regimes.

Additionally, our current focus is on vision-based backdoors in scenarios where users fine-tune pre-trained encoders on image datasets. We have not yet investigated language-domain backdoors or multimodal zero-shot settings (e.g., CLIP-like models), where threats could emerge from both image and text encoders. Extending T-Core to these domains remains an open challenge for future work.

7. Conclusion

In this study, we address the critical challenge of securing transfer learning models from backdoor attacks, where the security risk is amplified by the employment of untrusted pre-trained encoders and potentially poisoned datasets. Our exhaustive analysis shows that traditional defenses, which often depend on reactive approaches, are inadequate to the unknown threats and diverse attack vectors within transfer learning. To overcome this limitation, we thus advocate for a proactive mindset focused on identifying and expanding trustworthy elements and introduce the Trusted Core (T-Core) Bootstrapping framework. T-Core effectively neutralizes backdoor risks by initializing with meticulously vetted, high-credibility samples and progressively expanding the trusted dataset and model elements. Our comprehensive empirical evaluation, spanning a wide array of encoder and dataset poisoning, demonstrates the superiority of the T-Core. On the big picture, our work underscores the importance of adopting a proactive mindset in developing backdoor defenses, particularly in transfer learning, where an expanded attack surface and unknown threats heighten the security risk.

References

- [1] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, “Badnets: Evaluating backdooring attacks on deep neural networks,” *IEEE Access*, vol. 7, pp. 47230–47244, 2019.
- [2] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, “Targeted backdoor attacks on deep learning systems using data poisoning,” *arXiv preprint arXiv:1712.05526*, 2017.
- [3] S. Liang, M. Zhu, A. Liu, B. Wu, X. Cao, and E.-C. Chang, “Badclip: Dual-embedding guided backdoor attack on multimodal contrastive learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

- [4] J. Jia, Y. Liu, and N. Z. Gong, "BadEncoder: Backdoor Attacks to Pre-trained Encoders in Self-Supervised Learning," in *2022 IEEE Symposium on Security and Privacy (SP 22)*, pp. 2043–2059, 2022.
- [5] S. Wang, S. Nepal, C. Rudolph, M. Grobler, S. Chen, and T. Chen, "Backdoor Attacks Against Transfer Learning With Pre-Trained Deep Learning Models," *IEEE Transactions on Services Computing*, vol. 15, pp. 1526–1539, May 2022.
- [6] C. Li, R. Pang, Z. Xi, T. Du, S. Ji, Y. Yao, and T. Wang, "An embarrassingly simple backdoor attack on self-supervised learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4367–4378, October 2023.
- [7] G. Tao, Z. Wang, S. Feng, G. Shen, S. Ma, and X. Zhang, "Distribution Preserving Backdoor Attack in Self-supervised Learning," in *2024 IEEE Symposium on Security and Privacy (SP 24)*, IEEE Computer Society, 2024. ISSN: 2375-1207.
- [8] A. Saha, A. Tejankar, S. A. Koohpayegani, and H. Pirsiavash, "Backdoor attacks on self-supervised learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 22)*, pp. 13337–13346, 2022.
- [9] J. Zhang, H. Liu, J. Jia, and N. Z. Gong, "CorruptEncoder: Data poisoning based backdoor attacks to contrastive learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 24)*, 2024.
- [10] M. Barni, K. Kallas, and B. Tondi, "A new backdoor attack in cnns by training set corruption without label poisoning," in *2019 IEEE International Conference on Image Processing (ICIP 19)*, pp. 101–105, IEEE, 2019.
- [11] D. Tang, X. Wang, H. Tang, and K. Zhang, "Demon in the variant: Statistical analysis of DNNs for robust backdoor contamination detection," in *30th USENIX Security Symposium (USENIX Security 21)*, pp. 1541–1558, 2021.
- [12] X. Qi, T. Xie, Y. Li, S. Mahloujifar, and P. Mittal, "Revisiting the assumption of latent separability for backdoor defenses," in *The eleventh International Conference on Learning Representations (ICLR 23)*, 2023.
- [13] T. A. Nguyen and A. T. Tran, "WaNet - imperceptible warping-based backdoor attack," in *International Conference on Learning Representations (ICLR 21)*, 2021.
- [14] Z. Zhang, Q. Liu, Z. Wang, Z. Lu, and Q. Hu, "Backdoor defense via deconfounded representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 23)*, pp. 12228–12238, 2023.
- [15] Y. Li, X. Lyu, N. Koren, L. Lyu, B. Li, and X. Ma, "Anti-Backdoor Learning: Training Clean Models on Poisoned Data," in *Advances in Neural Information Processing Systems (NeurIPS 21)*, vol. 34, pp. 14900–14912, Curran Associates, Inc., 2021.
- [16] J. Hayase, W. Kong, R. Somani, and S. Oh, "SPECTRE: defending against backdoor attacks using robust statistics," in *Proceedings of the 38th International Conference on Machine Learning (ICML 21)*, 2021.
- [17] R. Zheng, R. Tang, J. Li, and L. Liu, "Data-free backdoor removal based on channel lipschitzness," in *European Conference on Computer Vision (ECCV 22)*, pp. 175–191, Springer, 2022.
- [18] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava, "Detecting backdoor attacks on deep neural networks by activation clustering," *arXiv preprint arXiv:1811.03728*, 2018.
- [19] B. Tran, J. Li, and A. Madry, "Spectral signatures in backdoor attacks," in *Advances in Neural Information Processing Systems (NeurIPS 18)*, pp. 8000–8010, 2018.
- [20] M. Pan, Y. Zeng, L. Lyu, X. Lin, and R. Jia, "ASSET: Robust backdoor data detection across a multiplicity of deep learning paradigms," in *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 2725–2742, 2023.
- [21] X. Qi, T. Xie, J. T. Wang, T. Wu, S. Mahloujifar, and P. Mittal, "Towards a proactive ML approach for detecting backdoor poison samples," in *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 1685–1702, 2023.
- [22] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, "STRIP: A defence against trojan attacks on deep neural networks," in *Proceedings of the 35th Annual Computer Security Applications Conference (ACSAC 19)*, pp. 113–125, 2019.
- [23] L. Hou, R. Feng, Z. Hua, W. Luo, L. Y. Zhang, and Y. Li, "IBD-PSC: Input-level backdoor detection via parameter-oriented scaling consistency," in *Forty-first International Conference on Machine Learning (ICML 24)*, 2024.
- [24] J. Guo, Y. Li, X. Chen, H. Guo, L. Sun, and C. Liu, "SCALE-UP: An efficient black-box input-level backdoor detection via analyzing scaled prediction consistency," in *The Eleventh International Conference on Learning Representations (ICLR 23)*, 2023.
- [25] Y. Zeng, S. Chen, W. Park, Z. Mao, M. Jin, and R. Jia, "Adversarial unlearning of backdoors via implicit hypergradient," in *International Conference on Learning Representations (ICLR 22)*, 2022.
- [26] M. Zhu, S. Wei, L. Shen, Y. Fan, and B. Wu, "Enhancing fine-tuning based backdoor defense with sharpness-aware minimization," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV 23)*, 2023.
- [27] T. Han, S. Huang, Z. Ding, W. Sun, Y. Feng, C. Fang, J. Li, H. Qian, C. Wu, Q. Zhang, et al., "On the effectiveness of distillation in mitigating backdoors in pre-trained encoder," *arXiv preprint arXiv:2403.03846*, 2024.
- [28] B. G. Doan, E. Abbasnejad, and D. C. Ranasinghe, "Februus: Input purification defense against trojan attacks on deep neural network systems," in *Proceedings of the 36th Annual Computer Security Applications Conference (ACSAC 20)*, pp. 897–912, 2020.
- [29] Y. Li, T. Zhai, Y. Jiang, Z. Li, and S.-T. Xia, "Backdoor attack in the physical world," *arXiv preprint arXiv:2104.02361*, 2021.
- [30] H. Qiu, Y. Zeng, S. Guo, T. Zhang, M. Qiu, and B. Thuraisingham, "Deepsweep: An evaluation framework for mitigating dnn backdoor attacks using data augmentation," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (AsiaCCS 21)*, pp. 363–377, 2021.
- [31] X. Liu, M. Li, H. Wang, S. Hu, D. Ye, H. Jin, L. Wu, and C. Xiao, "Detecting backdoors during the inference stage based on corruption robustness consistency," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 23)*, pp. 16363–16372, 2023.
- [32] X. Mo, Y. Zhang, L. Zhang, W. Luo, N. Sun, S. Hu, S. Gao, and Y. Xiang, "Robust backdoor detection for deep learning via topological evolution dynamics," in *2024 IEEE Symposium on Security and Privacy (SP 24)*, (Los Alamitos, CA, USA), pp. 174–174, IEEE Computer Society, may 2024.
- [33] W. Ma, D. Wang, R. Sun, M. Xue, S. Wen, and Y. Xiang, "The "Beatrix" Resurrections: Robust backdoor detection via gram matrices," in *NDSS*, 2023.
- [34] K. Huang, Y. Li, B. Wu, Z. Qin, and K. Ren, "Backdoor defense via decoupling the training process," in *Proceedings of the International Conference on Learning Representations (ICLR 22)*, 2022.
- [35] Z. Wang, H. Ding, J. Zhai, and S. Ma, "Training with more confidence: Mitigating injected and natural backdoors during training," in *Advances in Neural Information Processing Systems (NeurIPS 22)* (A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, eds.), 2022.
- [36] Y. Chen, H. Wu, and J. Zhou, "Progressive poisoned data isolation for training-time backdoor defense," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI 24)*, pp. 11425–11433, 2024.
- [37] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-Pruning: Defending against backdooring attacks on deep neural networks," in *International Symposium on Research in Attacks, Intrusions and Defenses (RAID 18)*, pp. 273–294, Springer, 2018.
- [38] W. Guo, L. Wang, Y. Xu, X. Xing, M. Du, and D. Song, "Towards inspecting and eliminating trojan backdoors in deep neural networks," in *2020 IEEE International Conference on Data Mining (ICDM 20)*, IEEE, 2020.
- [39] G. Shen, Y. Liu, G. Tao, S. An, Q. Xu, S. Cheng, S. Ma, and X. Zhang, "Backdoor scanning for deep neural networks through k-arm optimization," in *International Conference on Machine Learning (ICML 21)*, pp. 9525–9536, PMLR, 2021.
- [40] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural Cleanse: Identifying and mitigating backdoor attacks in neural networks," in *2019 IEEE Symposium on Security and Privacy (SP 19)*, pp. 707–723, May 2019.

- [41] S. Feng, G. Tao, S. Cheng, G. Shen, X. Xu, Y. Liu, K. Zhang, S. Ma, and X. Zhang, “Detecting backdoors in pre-trained encoders,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 23)*, pp. 16352–16362, 2023.
- [42] K. Gao, Y. Bai, J. Gu, Y. Yang, and S.-T. Xia, “Backdoor defense via adaptively splitting poisoned dataset,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 23)*, 2023.
- [43] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” in *International Conference on Machine Learning (ICML 17)*, pp. 1126–1135, PMLR, 2017.
- [44] A. Krizhevsky, “Learning multiple layers of features from tiny images,” *Tech Report*, 2009.
- [45] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition,” *Neural networks*, vol. 32, pp. 323–332, 2012.
- [46] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, “Reading digits in natural images with unsupervised feature learning,” in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.
- [47] A. Coates, A. Ng, and H. Lee, “An analysis of single-layer networks in unsupervised feature learning,” in *AISTATS*, 2011.
- [48] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 09)*, pp. 248–255, IEEE, 2009.
- [49] Y. Zeng, M. Pan, H. Jahagirdar, M. Jin, L. Lyu, and R. Jia, “META-SIFT : How to sift out a clean subset in the presence of data poisoning?,” in *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 1667–1684, 2023.
- [50] T. Han, W. Sun, Z. Ding, C. Fang, H. Qian, J. Li, Z. Chen, and X. Zhang, “Mutual information guided backdoor mitigation for pre-trained encoders,” *arXiv preprint arXiv:2406.03508*, 2024.
- [51] R. Bie, J. Jiang, H. Xie, Y. Guo, Y. Miao, and X. Jia, “Mitigating Backdoor Attacks in Pre-Trained Encoders via Self-Supervised Knowledge Distillation,” *IEEE Transactions on Services Computing*, vol. 17, pp. 2613–2625, Sept. 2024.
- [52] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *KDD*, pp. 226–231, 1996.

Appendix A. Implementation Details of T-Core

In Sec. 4, we present the overall algorithmic details of our Trusted Core Bootstrapping framework, here we illustrate the implementation details of T-Core.

Details of Topological Invariance Sifting. The algorithm Algorithm 1 in Sec. 4.1.1 extracts high-credibility seed data from a poisoned dataset by utilizing topological invariance across various layers of a well-trained poisoned model. For clustering, we employ activations from the output of the encoder and the first two layers of the classifier, and implement density-based clustering using DBSCAN [52]. We set the number of nearest neighbors m to 50 and employ Euclidean distance for neighbor determination. The seed selection ratio α is set to 1%.

Details of Seed Expansion Algorithm 2 in Sec. 4.1.2 expand the seed data obtained from Algorithm 1. We set the select ratio r_{expand} for expansion as 5%, and the expansion ratio $|D_{\text{sub}}|/|D|$ as 10%.

Details of Encoder Channel Filtering The algorithm in Algorithm 3 from Sec. 4.2 filters trusted and untrusted channels within the pre-trained encoder. The unlearning process

stops once the training accuracy drops below 20%. The recovery process is conducted using the Adam optimizer with a learning rate of 0.01. Training is performed for 120 epochs. The filtering threshold is set to cut off 10% of channels as untrusted.

Details of Clean Bootstrapping Training The clean bootstrapping described in Algorithm 4 iteratively expands the clean subset and refines the model parameters. We set both $Iter_1$ and $Iter_2$ to 10, with both $\gamma_1\%$ and $\gamma_2\%$ set to 2%. Additionally, $\gamma_3\%$ is set to 5%. This process continues until the clean subset reaches $\rho = 90\%$ of the entire dataset.

Appendix B. Experimental Setup Explanation

B.1. Datasets Explanation

We detail the datasets utilized and how we process them as pre-training or downstream datasets as follows:

- **STL-10 [47]:** This dataset comprises 5,000 labeled training images, 8,000 labeled testing images, and an additional 100,000 unlabeled images, each with dimensions of $96 \times 96 \times 3$, and all parts of the image set can be evenly divided into ten classes, where each image belongs to a class. We use the unlabeled images as a pre-training dataset and the labeled part for the downstream training and testing.
- **CIFAR-10 [44]:** This dataset is also a balanced dataset containing 50,000 labeled training images and 10,000 testing images, both of which are evenly divided into 10 classes. Each image has a size of $32 \times 32 \times 3$. We use this dataset for the downstream dataset and remove the labels of the training images when pre-training the encoder.
- **GTSRB [45]:** This dataset contains 43 classes of traffic signs, split into 39,209 training images and 12,630 test images. Each image has a size of $32 \times 32 \times 3$. Furthermore, this dataset has an uneven distribution of the number of images belonging to each class, ranging from 210 to 2250, which presents a real-world challenge, especially when under poisoning. Thus, we use it for the downstream dataset to evaluate the effectiveness of defenses.
- **SVHN [46]:** In this dataset, every image depicts a digit from the house numbers collected from Google Street View with the size of $32 \times 32 \times 3$. Additionally, the images fall into one of the ten possible digit categories. There are 73,257 training images and 26,032 testing images in this dataset. Moreover, extraneous digits are added at the edges of the primary digit in focus, making it even more challenging to separate poisoned and clean samples under SVNH. Thus, we also use it as a downstream dataset to test the robustness of defenses.
- **ImageNet [48]:** This dataset is built for large-scale object classification and contains 1,281,167 training samples and 50,000 testing samples in 1000 classes. The input size is $224 \times 224 \times 3$. We use a 100-class subset for the pre-training dataset and a 10-class subset for the downstream dataset.

B.2. Pre-trained Encoder Explanation

In this section, we provide the detailed implementation of each encoder poisoning attack, specifically how we obtain both clean and poisoned encoders, and how these encoders are utilized in the context of transfer learning.

BadEncoder and **DRUPE**: Both of these methods inject backdoors by fine-tuning a clean pre-trained encoder, assigning a target class to a specific concept within a downstream dataset. Our experiments utilize STL-10 as the pre-training dataset while targeting backdoors for CIFAR-10, GTSRB, and SVHN. Conversely, we also use CIFAR-10 as the pre-training dataset, with backdoors aimed at STL-10, GTSRB, and SVHN.

CTRL: In contrast to the backdoor injection methods used by BadEncoder and DRUPE, CTRL contaminates the unlabeled images in the pre-training dataset, specifically targeting a concept within the label set of that dataset. As a result, its backdoor functionality is limited to downstream datasets that share the same label set or a subset of it. Accordingly, we adhere to the methodology outlined in [6] and utilize CIFAR-10, STL-10, and GTSRB as both pre-training and downstream datasets.

SSLBackdoor and **CorruptEncoder**: These methods also employ the same 100-class subset of ImageNet as the pre-training dataset. They inject backdoors in a manner similar to the CTRL method by directly poisoning the pre-training data. For our experiments, we select a 10-class subset of ImageNet as the downstream dataset, ensuring that the target classes for both SSLBackdoor and CorruptEncoder are included in this subset.

Additionally, for the clean encoders of STL-10 and CIFAR-10, we follow the approach in [4] to train a ResNet18 using SimCLR, leveraging the publicly available code from SimCLR⁴. The clean ImageNet encoder is sourced from [8]⁵. The backdoor encoders are constructed through the respective backdoor poisoning methods. For BadEncoder⁶ and DRUPE⁷, we directly utilize their open-source checkpoints trained on STL-10 and CIFAR-10. For the ImageNet backdoor encoder, we employ the checkpoints provided by [9]⁸ for both SSL-Backdoor and CorruptEncoder.

B.3. Dataset Poisoning Explanation

For dataset backdoor poisoning, we use the backdoor-toolbox⁹ benchmark to implement all the attacks on downstream datasets. Note that this benchmark does not provide

4. <https://github.com/leftthomas/SimCLR>
5. <https://github.com/UMBCvision/SSL-Backdoor>
6. <https://github.com/jinyuan-jia/BadEncoder>
7. <https://github.com/Gwinhen/DRUPE>
8. <https://github.com/jzhang538/CorruptEncoder>
9. <https://github.com/vtu81/backdoor-toolbox>

experimental setups for certain attack methods on some datasets. For instance, TaCT attack on ImageNet is originally not supported. We conducted experiments by adapting the settings from other datasets.

Trigger and Target Class: In the context of **Threat-2**, we employed distinct triggers and target classes to avoid conflicts between Encoder Poisoning and Dataset Poisoning. Specifically, when both **Threat-1** and **Threat-2** are present, we maintained the same approach used for **Threat-2**. In **Threat-3**, we utilized the same triggers and target classes as those applied in the pre-trained encoder. For **Threat-2**, aside from TaCT, the target classes are as follows: “Car” for STL-10, “Bird” for CIFAR-10, “Speed limit (50km/h)” for GTSRB, “2” for SVHN, and “n01855672 (Goose)” for ImageNet-10. The specific target classes for TaCT and **Threat-3** across various datasets are detailed in Table 18. **Poisoning and Cover Ratio**. By default, the poisoning rate was set to 20% of the target category. For specific attacks like TaCT, WaNet, Adap-Blend, and Adap-Patch, the cover rate was set to 1/4 of the poisoning rate.

TABLE 18: Target Class of Various Dataset Poisoning Methods on Downstream Dataset

| Threat Type | Threat 2 | | | Threat 3 | | |
|-------------|----------------------|------------------------|---|--------------------------|-----------------------------|---------------------|
| | Dataset | Target Class | Cover Class | BadEncoder Target Class | DRUPE Target Class | CTRL Target Class |
| STL-10 | Car | Truck | Dog Horse Monkey | Trunk | Trunk | Airplane |
| CIFAR-10 | Bird | Truck | Dog Frog Horse | Airplane | Airplane | Airplane |
| GTSRB | Speed limit (60km/h) | Speed limit (50km/h) | End of speed limit(80km/h) Speed limit(100km/h) | Priority Road | Priority Road | Speed limit(20km/h) |
| SVHN | 2 | 1 | 5 6 7 | 0 | 0 | None |
| Dataset | Target Class | Source Class | Cover Class | SSLBackdoor Target Class | CorruptEncoder Target Class | - |
| ImageNet-10 | n01855672 (Goose) | n01775062 (Wolfspider) | n02120079(Whitefox) n02447366(Badger) n02483362(Gibbon) | n02116738 (Hunting Dog) | n02116738 (Hunting Dog) | - |

B.4. Baseline Defense Explanation

For baseline defense, we tailor all the end-to-end training from scratch to transfer learning using a clean or poisoned encoder based on the specific threat. We utilize the publicly available code of [21]¹⁰ to implement STRIP, AC, Spectral, SPECTRE, and CT. We utilize the toolbox¹¹ to implement IBD-PSC and SCALE-UP. We implement ASSET¹², ABL¹³, CBD¹⁴, CLP¹⁵, SSLBackdoorMitigation¹⁶, META-SIFT¹⁷, FT-SAM¹⁸, and I-BAU¹⁹ using their respective official code repositories.

10. <https://github.com/Unispac/Fight-Poison-With-Poison>
11. <https://github.com/THUYimingLi/BackdoorBox>
12. <https://github.com/reds-lab/ASSET>
13. <https://github.com/bboylg/ABL>
14. <https://github.com/zaixizhang/CBD>
15. <https://github.com/rkteddy/channel-Lipschitzness-based-pruning>
16. <https://github.com/wssun/SSLBackdoorMitigation>
17. <https://github.com/ruoxi-jia-group/Meta-Sift>
18. <https://github.com/SCLBD/BackdoorBench>
19. <https://github.com/YiZeng623/I-BAU>

TABLE 19: Performance of the state-of-the-art **inference-time defenses** SCALE-UP and IBD-PSC under different threat scenarios, using the CIFAR-10 dataset.

| Threat Type | Poisoning Method | SCALE-UP | | | | IBD-PSC | | | |
|-----------------|-------------------|-------------------|---------------------|----------------|---------------|-------------------|---------------------|----------------|---------------|
| | | TPR(%) \uparrow | FPR(%) \downarrow | AUC \uparrow | F1 \uparrow | TPR(%) \uparrow | FPR(%) \downarrow | AUC \uparrow | F1 \uparrow |
| Threat-2 | BadNets | 92.80 | 33.98 | 0.79 | 0.12 | 0.00 | 3.38 | 0.48 | 0.00 |
| | Blended | 20.24 | 38.22 | 0.41 | 0.03 | 0.24 | 3.27 | 0.48 | 0.00 |
| | SIG | 14.80 | 37.66 | 0.39 | 0.02 | 0.80 | 3.13 | 0.49 | 0.01 |
| | WaNet | 28.56 | 40.07 | 0.44 | 0.03 | 4.80 | 0.42 | 0.52 | 0.08 |
| | TaCT | 56.88 | 35.98 | 0.60 | 0.07 | 0.00 | 5.29 | 0.47 | 0.00 |
| | Adap-Blend | 29.60 | 35.54 | 0.47 | 0.04 | 0.80 | 4.14 | 0.48 | 0.01 |
| | Adap-Patch | 32.48 | 34.58 | 0.49 | 0.04 | 1.92 | 2.28 | 0.50 | 0.02 |
| Threat Type | Poisoning Method | TPR(%) \uparrow | FPR(%) \downarrow | AUC \uparrow | F1 \uparrow | TPR(%) \uparrow | FPR(%) \downarrow | AUC \uparrow | F1 \uparrow |
| Threat-1 | BadEncoder | 33.60 | 37.00 | 0.48 | 0.04 | 3.76 | 4.59 | 0.50 | 0.03 |
| | DRUPE | 33.84 | 36.90 | 0.48 | 0.04 | 0.24 | 3.93 | 0.48 | 0.00 |
| Threat-3 | BadEncoder | 53.04 | 38.60 | 0.57 | 0.06 | 50.64 | 3.96 | 0.73 | 0.33 |
| | DRUPE | 51.76 | 34.18 | 0.59 | 0.07 | 3.76 | 6.98 | 0.48 | 0.02 |

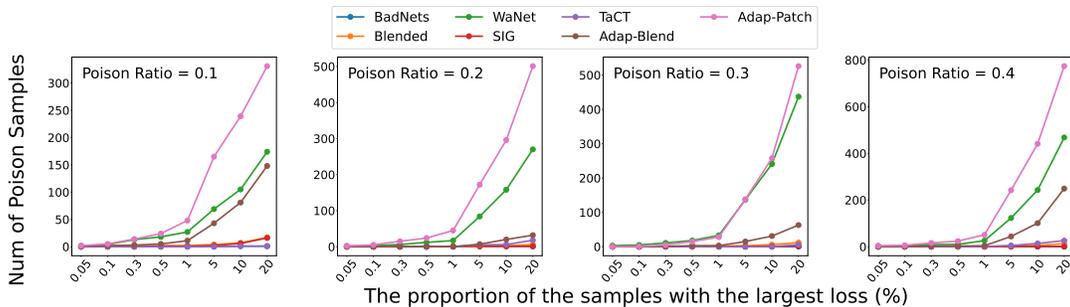


Figure 6: Number of poison samples in the largest loss region of the target class after seed expansion. Experiments are conducted on 7 dataset poisoning attacks at poison ratios of 0.1, 0.2, 0.3, and 0.4 for the target class.

```

1 class MaskedLinear(nn.Module):
2     def __init__(self, original_weight, mask, bias=None):
3         super().__init__()
4         self.original_weight = original_weight
5         self.mask = mask # Trainable mask initialized as all ones
6         self.bias = bias
7
8     def forward(self, x):
9         # Clamp mask between 0 and 1
10        self.mask.data.clamp_(0,1)
11        masked_weight = self.original_weight * self.mask.unsqueeze(1)
12        return torch.nn.functional.linear(x, masked_weight, self.bias)

```

Figure 7: Implementation of the MaskedLinear class, which applies a trainable mask to the weights of a linear layer. We use this code to transform a linear layer of ViT into a masked layer. The mask is clamped between 0 and 1 to ensure valid weight adjustments.