Persistent Backdoor Attacks in Continual Learning

Zhen Guo
Saint Louis University
zhen.guo.2@slu.edu

Abhinav Kumar
Saint Louis University
abhinav.kumar@slu.edu

Reza Tourani
Saint Louis University
reza.tourani@slu.edu

Abstract

Backdoor attacks pose a significant threat to neural networks, enabling adversaries to manipulate model outputs on specific inputs, often with devastating consequences, especially in critical applications. While backdoor attacks have been studied in various contexts, little attention has been given to their practicality and persistence in continual learning, particularly in understanding how the continual updates to model parameters, as new data distributions are learned and integrated, impact the effectiveness of these attacks over time.

To address this gap, we introduce two persistent backdoor attacks–*Blind Task Backdoor* and *Latent Task Backdoor*–each leveraging minimal adversarial influence. Our blind task backdoor subtly alters the loss computation without direct control over the training process, while the latent task backdoor influences only a single task's training, with all other tasks trained benignly. We evaluate these attacks under various configurations, demonstrating their efficacy with static, dynamic, physical, and semantic triggers. Our results show that both attacks consistently achieve high success rates across different continual learning algorithms, while effectively evading state-of-the-art defenses, such as SentiNet and I-BAU.

1 Introduction

Catastrophic forgetting, first identified by McCloskey and Cohen [30], refers to the significant loss of previously learned knowledge when a model is trained sequentially on new tasks. Catastrophic forgetting negatively impacts the performance of neural networks in practical applications, such as autonomous driving and natural language processing, where models continuously evolve to address distribution drifts in the underlying data or integrate new knowledge. Various *Continual Learning* (CL) algorithms have been proposed to address catastrophic forgetting in neural networks by emulating the human ability to learn continuously [7, 22, 33, 34, 37, 41, 58]. These algorithms primarily prevent catastrophic forgetting by replaying prior experiences [7, 33] or regularizing loss computation [22, 58], enabling models to continuously adapt
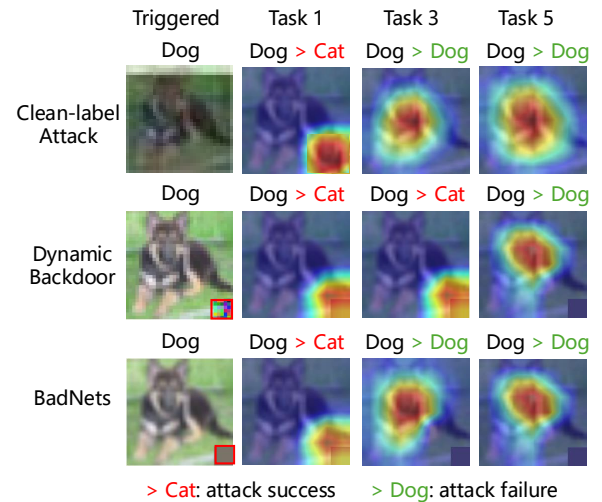

Figure 1: Persistence analysis of clean-label backdoor [56], dynamic backdoor [39], and BadNets [17], using a ResNet18 trained on SplitCIFAR10 with LwF [28] (trigger embedding in Task 1). Subplots shows the model's attention at Tasks 1, 3, and 5. Initially, all attacks successfully alter the output (*Dog > Cat*), as the model's attention is on the trigger. However, as the model learns subsequent tasks, its attention shifts away from the trigger, causing all attacks to fail (*Dog > Dog*).

to new data in dynamic environments [2, 35].

While CL enhances the adaptability and robustness of neural networks, their security and resilience to attacks remain relatively unknown. This uncertainty stems from the continuous adaptation and updating of model parameters in CL systems, leading to dynamic behaviors that might facilitate certain attacks while naturally mitigating others. One significant area of concern is the susceptibility of these algorithms to backdoor or Trojan attacks [3, 17, 56], which are designed to produce incorrect outputs only when inputs contain specific trigger features. To backdoor a neural network, the adversary embeds hidden malicious behaviors into the model by poisoning a subset of the training samples with a crafted trigger patch. Once trained, the attack is activated by applying the predefined patch to inputs, causing misclassifications to the

target label. Various backdoor attacks have been proposed focusing on different modalities of backdoor transformation, ranging from invisible triggers [9] and semantic triggers [3] to physical triggers [27] and input-aware dynamic backdoor [39] in both centralized and federated settings [4]. A few recent initiatives have explored backdoor attacks against CL, aiming to degrade the model performance of target tasks [19, 60] or induce artificial catastrophic forgetting [21, 23, 45]. However, further research is needed to shed light on the effectiveness and persistence of targeted backdoor attacks against CL.

We argue that executing a successful targeted backdoor attack against CL is inherently challenging due to the continuous updating of model weights, which can gradually eliminate the injected backdoors. This phenomenon has been observed in time-varying models, where fine-tuning a backdoored model with new data has resulted in backdoor forgetting [24]. To validate our premise, we evaluated the persistence of three prominent backdoor attacks–clean-label [56], dynamic backdoor [39], and BadNets [17]–using the ResNet18 model trained on SplitCIFAR10 with LwF algorithm [28]. The attention heatmap plots (Figure 1) reveal that the model's attention initially focuses on the embedded trigger in the first task, where the backdoor was implanted. However, as the model evolves, attention shifts away from the trigger and toward more influential input features (*e.g.,* the dog's face), causing the attacks to fail after two or three new tasks. This experiment demonstrates that existing backdoor attacks do not persist.

The failure of backdoor attacks in CL becomes more evident when analyzing the model's behavior during sequential task training, as illustrated in Figure 2. The left graph depicts the loss surface of Task 1 during model training, as in the previous experiment. During Task 1 training, the model adjusts its parameters, moving from the initial state at T0 to a new set of parameters at T1 along the solid black line, minimizing the loss for this task. For training Task 2, as shown in the right graph, the model starts from T1 and continues to adjust its parameters to T2 (solid black line), minimizing the loss for Task 2. However, this trajectory results in a significant shift in the model's parameters, as the model prioritizes the new task over the previous one. This drastic parameter update diminishes the effectiveness of previously embedded backdoors. Taking the alternative trajectory along the orange dashed line, however, minimizes the new task's loss while limiting parameter changes, thereby better retaining the influence of the backdoor while still learning the new task. This is an approach we will explore further in this work.

Our contributions. Building on this observation, we investigate the efficacy of prominent backdoor attacks in CL and propose a novel targeted attack, called the ***persistent backdoor***. The *core concept* of the persistent backdoor involves identifying a subset of model components, *e.g.,* layers and neurons, that are crucial for the classification task and remain stable throughout model training, and then using them for trigger embedding. The design of the persistent backdoor is
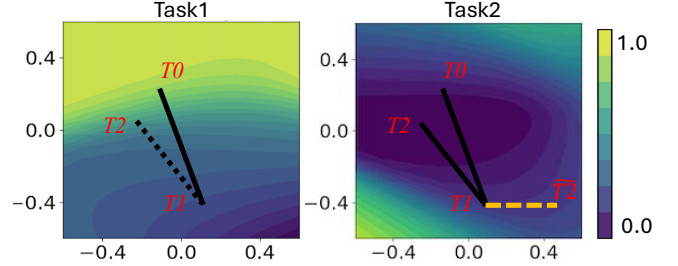

Figure 2: Evolution of the model parameters from the corresponding experiment. The loss surface of Task 1 (left) shows the parameter trajectory from T0 to T1. The loss surface for Task 2 (right) depicts the parameters evolving from T1 to T2 along the black line. These drastic parameter updates erase the embedded trigger, leading to backdoor forgetting.

both generic and algorithm-agnostic, making it applicable to regularization-based and replay-based CL algorithms. We further propose two strategies for our persistent backdoor attack, ***Blind Task Backdoor*** and ***Latent Task Backdoor***: the former subtly modifies loss computation without controlling the training process, while the latter controls a single training task. To demonstrate the effectiveness of these persistent attack strategies, we apply them to inject various types of backdoors, including (1) static and dynamic backdoors, (2) physical backdoors, and (3) NLP word insertion backdoors. We demonstrate the effectiveness of these two strategies under various conditions through comprehensive experiments, targeting six CL algorithms, three different neural networks, and three datasets specifically designed for CL scenarios. Finally, we show how our attack can successfully evade existing defenses like SentiNet [10] and I-BAU [55].

2 Background and Related Work

2.1 Continual Learning

Continual learning has emerged to address catastrophic forgetting–a common issue in multilayer perceptron-based networks where the model tends to forget previously learned information when trained sequentially on new tasks. The two primary CL categories include *regularization-based* and *replay-based* CL algorithms. We do not cover dynamic architecture approaches as this is beyond our scope.

Regularization-based approaches prevent catastrophic forgetting by penalizing significant changes to key model parameters during the learning of new tasks. Some notable algorithms include Synaptic Intelligence (SI) [58], Elastic Weight Consolidation (EWC) [22], Context-Dependent Gating (XdG) [41], and Learning without Forgetting (LwF) [28]. SI algorithm maintains unique synaptic weights for each task, using weight importance estimation to reduce interference during new task training. Similarly, EWC penalizes changes to the network parameters based on their importance in previously learned tasks, preserving critical parameters. XdG

employs gated neurons to activate distinct sets of neurons per task. Lastly, LwF uses the model's predictions on old tasks as soft targets in training new tasks, preserving prior knowledge.

Replay-based techniques reiterate a subset of past data (real or synthetic) to reinforce prior knowledge during the training of new tasks, *e.g.,* Efficient Gradient Episodic Memory (A-GEM) [7] and Deep Generative Replay (DGR) [33]. A-GEM stores and replays past experiences within a memory buffer during training, enhancing sample efficiency and training stability for learning agents. It also implements constraints to preserve previously acquired knowledge. In contrast, generative replay methods, such as DGR, use a dual-model architecture that combines a deep generative model, with a task-solving model. This setup efficiently integrates training data from past tasks with new task data, facilitating CL.

2.2 Backdoor Attacks

A backdoor attack is an adversarial technique where an attacker injects malicious patterns, known as triggers, into a small subset of the training data, assigning these data points incorrect labels [43, 50, 51]. During the inference, the backdoored model produces incorrect outputs whenever the specific trigger is present in the input. This allows the adversary to control the model's decisions on triggered inputs, while keeping the model's general performance intact. Backdoors are a more targeted form of attack compared to universal adversarial perturbations [15, 25, 29, 31, 59]. While both cause a model to misclassify inputs to an attacker-chosen label, backdoors require modifying both the model and the input, but offers more flexibility. Moreover, backdoor attacks differ from data poisoning, where the attacker's primary goal is to degrade the overall model performance.

Various backdoor attacks exist, utilizing different types of triggers. *Static triggers* [11, 12, 17, 26] are fixed, invariant patterns embedded in input, characterized by consistent properties like location and pattern. While effective, their fixed nature makes them easy to detect. In contrast, *dynamic triggers* [16, 32, 39] involve patterns and placements that vary across inputs, enhancing stealth and making detection significantly more difficult. *Physical triggers* [18, 44, 56, 59] are real-world objects that activate the backdoor, posing serious risks in environments like autonomous driving, where they blend seamlessly and are hard to distinguish from benign objects. Lastly, *word triggers* [3, 13, 57] in natural language models are specific words, either inherent or injected, that activate a backdoor. Their complexity and the need for deep linguistic analysis make detection particularly challenging.

Recent efforts have explored attacks against CL, focusing on model poisoning [19, 60], inducing artificial catastrophic forgetting [21, 23, 45], or generating false memories [46]. Differently, we aim to expose CL vulnerabilities by designing two persistent backdoors with varying adversarial capabilities.
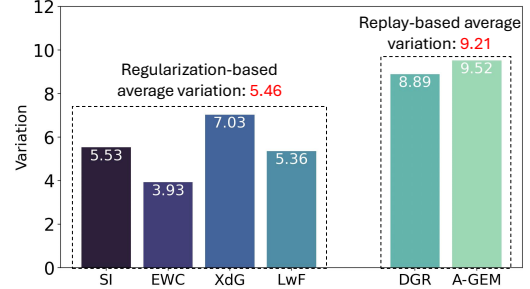

Figure 3: Quantifying the parameter variation at the algorithmic level indicated that regularization-based algorithms have higher stability compared to replay-based algorithms. This is partly due to the nature of the regularization-based approaches, which penalize modification to learned parameters.

3 Observation

In this section, we will discuss our findings regarding the variations in the parameters of a deep learning model as it continually learns new tasks, highlighting the behaviors of different CL algorithms. The insights drawn from these observations, coupled with the evident trend we have identified, lay the foundation for implementing our proposed persistent backdoor attack. We conducted a series of experiments using a neural network with three fully connected hidden layers, employing ReLU activation and a cross-entropy loss function. The model was trained on the SplitMNIST dataset, including five sequential tasks, each consisting of two classes. We used the Adam optimizer with a learning rate of 0.001 and repeated each experiment 10 times.

In these experiments, we monitor the changes in parameter values across the three hidden layers and different tasks and analyze these variations at three granularity levels: algorithmic, layer-wise, and neuron-level. We quantify the algorithmic-level variation by $||\frac{\sum_i L_i}{N}||_2$, where N is the number of the tasks and L_i is the variation of the ith layer across two consecutive tasks. More formally, $L_i = ||\sum_j \Delta_i^j||_2$, where Δ_i^j represents the value change of the jth neuron in the ith layer for the current task compared to the previous task. Applying the L_2 norm to $\sum_j \Delta_i^j$ calculates the magnitude of the vector formed by the cumulative changes in the ith layer.

Insight 1 – Algorithmic-level Analysis: At the highest granularity level, we analyzed various CL algorithms to characterize their overall behavior and parameters' variation as they learn new tasks. Figure 3 illustrates the parameter variations for six CL algorithms, calculated using the aforementioned approach. From this figure, it is evident that regularization-based algorithms exhibit a lower degree of variation in their parameters compared to replay-based algorithms. The rationale behind this phenomenon is that regularization-based algorithms, such as SI, EWC, XdG, and LwF, tend to penalize drastic changes to the parameters learned in prior tasks, imposing a high cost for modifications to the learned model. In contrast, replay-based algorithms, such as DGR and A-GEM, do not impose
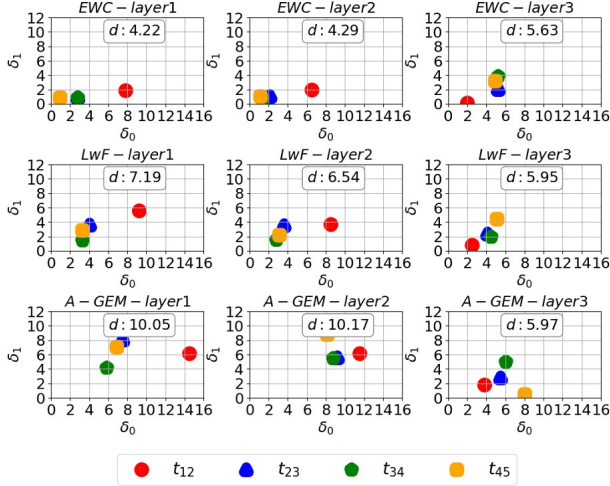
Figure 4: The layer-wise analysis revealed that for EWC, the last layers experience larger variations compared to the previous two layers, whereas for LwF and A-GEM, the last layers exhibit smaller variations than the preceding layers.

such constraints. Instead, these algorithms retain the knowledge of the previously learned tasks by replaying and learning from prior data distributions without penalizing parameter updates. As a result, the model experiences a higher degree of variation in its parameters' values. For instance, the average variation in the parameter values of the regularization-based EWC algorithm is only 3.93, while the replay-based A-GEM algorithm shows an average value of 9.52.

Insight 2 – Layer-wise Analysis: We then analyzed each layer of the model to quantify the layer-wise parameter variation between consecutive tasks. For this analysis, we performed principal component analysis on each layer to derive two components, δ_0 and δ_1. Each point in the resulting plot represents the degree of variation in δ_0 and δ_1 between two consecutive tasks. We noticed that different layers of the model exhibit varying degrees of change (Figure 4). More importantly, we observed that different algorithms impact layer variation in distinct ways–some induce more significant changes in the last layers, while others result in greater variations in the earlier layers. For example, in EWC, the highest degree of parameter variation was observed in the third layer (5.63), whereas LwF (7.19) and A-GEM (10.17) exhibited the greatest variation in the first and second layers, respectively.

In neural networks, as the depth increases, layers transition from learning low-level features like edges and textures to high-level, abstract representations such as shapes and objects. As a result, in a CL context, it is expected that the parameters of the earlier layers remain relatively stable when learning new data distributions, while those in the final layers exhibit more significant changes–as we observed in the EWC algorithm. Contrary to this understanding, five other algorithms showed the unexpected behavior of higher variation in earlier layers. We attribute it to the possibility that features learned in the earlier layers are more sensitive to the new task data
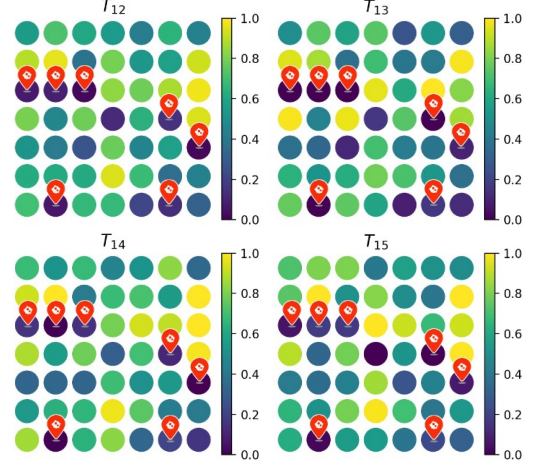

Figure 5: Parameter variations of 49 selected neurons from the model's third layer using LwF. T_{12} represents the parameter variations between Task 1 and Task 2, T_{13} between Task 1 and Task 3, T_{14} between Task 1 and Task 4, and T_{15} between Task 1 and Task 5. A subset of neurons, highlighted with red icons, exhibits high stability as the model progresses.

distribution, especially when new tasks require fundamentally different low-level features. This sensitivity leads to greater variation as the network adjusts to accommodate new features.

Insight 3 – Neuron-level Analysis: We also analyzed the behavior of the neurons as the model evolved, focusing on the third layer of the LwF algorithm due to its greater stability. Given the high dimension of parameters, we selected 49 out of 400 neurons for better illustration by dividing the parameter variations into seven ranges and sampling seven neurons from each range (Figure 5). We then quantified the variations in these neurons' values between Task 1 and all subsequent tasks (*e.g.*, T_{13} represents the changes in neuron values from Task 1 to Task 3), and visualized the results in each sub-figure of Figure 5. This revealed a pattern: a small subset of neurons, marked with a red icon, exhibits small variations in their values throughout across training all tasks. In contrast, most other neurons exhibit varying degrees of change, with some exhibiting a significant shift in values in later tasks.

Insight 4 – Component Stability Analysis: Finally, we analyzed neuron stability by examining weight variations during the continual learning process. Using the A-GEM algorithm, we trained a ResNet18 model on the SplitCIFAR10 dataset over five tasks. After Task 1, we identified 200 stable neurons, important for Task 1, using the diagonal Fisher matrix and randomly selected 200 others for comparison. We then trained the model on four additional tasks and quantified the weight variations of these neuron groups after each task relative to their baseline values established in Task 1, with δ_{1-i} representing the variation between Task 1 and Task i. We visualized the weight variation of these two neuron groups using the kernel density estimation as shown in Figure 6. It is evident that random neurons (orange distribution) exhibit progressively larger mean variations across tasks, whereas
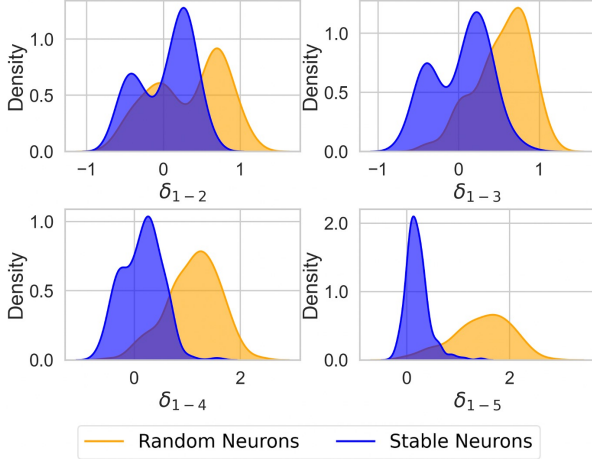
Figure 6: The kernel density estimation of parameter variations between stable and randomly selected neurons. The mean value of stable and random neurons are 0.05 and 0.36, respectively. Notably, the gap between the two distributions widens as the model progresses through additional tasks, indicating increasing differentiation in their respective stability.

stable neurons (blue distribution) remain relatively consistent. Quantitative results further support this observation: stable neurons exhibit a mean variation of 0.05 with a standard deviation of 0.22, whereas randomly selected neurons show a significantly higher mean variation of 0.36 and a standard deviation of 0.44. Notably, the gap between the two distributions widens as the model learns more tasks, indicating that stable neurons effectively retain their weights, while random neurons experience increasingly significant variations (refer to Figure 19 in Appendix 10 for per-neuron value variation).

To further investigate the root cause of the observed neuron stability, we quantified the overlap among neurons utilized during the training of individual tasks using the normalized intersection-over-union (IoU) score and visualized these intersections as a heatmap in Figure 7. The analysis revealed that the largest intersection, excluding intra-task comparisons, is 6.4%, observed between Tasks 4 and 5, suggesting a small degree of overlap. However, the overall low IoU scores indicate that tasks primarily rely on distinct sets of neurons. These results demonstrate that different tasks in Cl rely on predominantly disjoint sets of neurons, and the neurons critical to a task remain stable in learning subsequent tasks.

4 Persistent Backdoor Attack

In this section, we introduce the threat model, including two adversaries with different capabilities and key properties, followed by an overview of the proposed persistent backdoor.

4.1 Threat Model and Assumptions

The rise in deep learning's popularity has turned platforms like GitHub and Hugging Face into repositories for hundreds
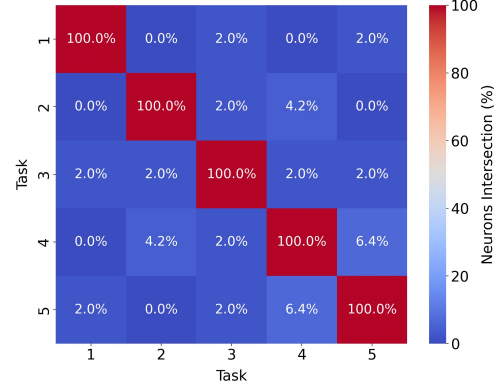

Figure 7: Neuron overlap heatmap showing the percentage of neuron intersection across five tasks. Each cell represents the normalized intersection-over-union score between two tasks. Diagonal values of 100% in self-comparison highlight that distinct neurons are exclusively utilized for different tasks.

of thousands of open-source models. These models are publicly available either as pre-trained weights, ready for immediate use in various tasks, or as source code, which developers can adapt for specific applications. This accessibility has led to the widespread integration of open-source code, driven by the need to reduce costs and accelerate the development of large models [1]. For instance, Hugging Face reported over 600 million model downloads in August 2023 alone [5]. However, due to the widespread adoption of platforms like GitHub, Twitter, and Flicker in the ML community, the attack vector has expanded beyond poisoned data to include poisoned weights and carefully crafted code snippets [3, 38]. Given such threats, we consider two specific attacks–**Blind Task Backdoor** and **Latent Task Backdoor**–in which the adversary targets backdooring a specific task. These threats are stealthy, as the backdoor remains dormant until the targeted task is executed, significantly reducing the risk of their detection.

The blind task backdoor (BTB), which we will refer to as the *blind backdoor* for brevity, is a type of code poisoning attack where the adversary indirectly impacts the model training process without access to the training data. Similar to [3], the adversary maliciously manipulates the machine learning codebase, referred to as the *attack code*, specifically targeting the calculation of the loss value during training. The attack code also includes a module for generating poisoned data samples by injecting triggers into a small subset of the training data and assigning incorrect labels, as detailed in [3]. As a result, the adversary remains blind to both the training process and the data. In continual learning, the same attack code is used across all tasks, allowing the backdoor to be injected at any point in the learning process without the adversary directly interacting with the model during training. This attack is particularly effective when ML code sourced from a repository is used unchanged to train a model, allowing the adversary to embed malicious behaviors without the developer's knowledge, thereby compromising the application's integrity.
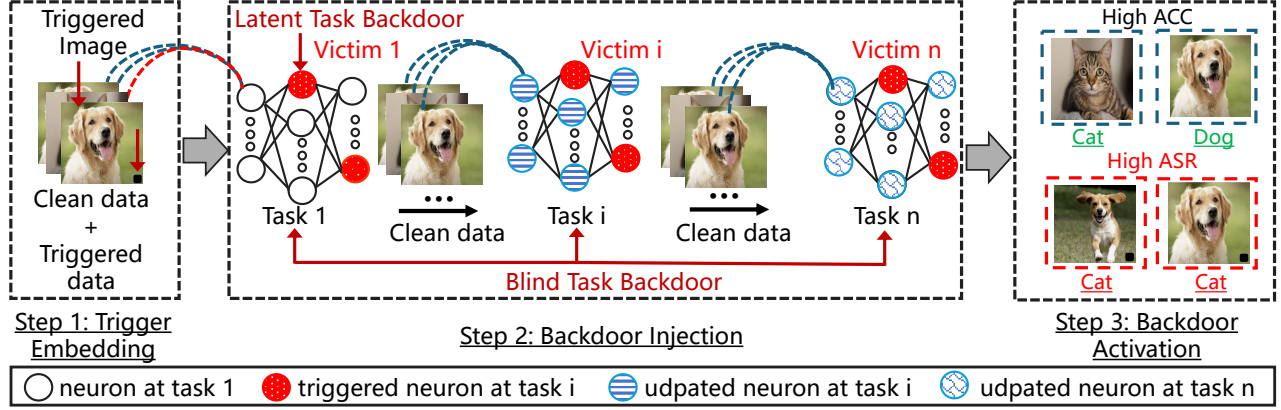
Figure 8: The three steps of our Persistent Backdoor Attack. *Step 1: Trigger Embedding*–This involves generating triggered training data during the training process of a specific task. *Step 2: Backdoor Injection*–The attack code, generated by either **blind backdoor** or **latent backdoor**, implants the backdoor into the model during training. *Step 3: Backdoor Activation*–When fed with the triggered input, the backdoored model produces an incorrect prediction, which persists across all subsequent tasks.

We note that loss computation in deep learning models is context-dependent, varying with architecture, data modality, tasks, and the CL algorithm. This diversity often results in a single codebase employing multiple loss functions, making it challenging to detect malicious alterations through code review alone, as detailed in [3].

Unlike the blind backdoor, which relies on indirect manipulation of the entire training process, the latent task backdoor (LTB), referred to as the *latent backdoor*, is a more sophisticated attack method in which the adversary manipulates the training process of *only one task*. The weight modifications introduced during this task are designed to backdoor future targeted tasks, even when their training processes remain benign. This restriction on the adversary showcases that, in CL, as in classical ML pipelines, *publicly shared weights can contain dormant and hidden functionalities that may be exploited in later tasks*. The adversary achieves this by modifying the loss computation for a single task, leading to a controlled update of specific weights during the training process. The restricted threat model of LTBs (*i.e.,* single-task manipulation) highlights the risks developers face when directly using pre-trained weights to build applications, fine-tune custom models, few-shot learning [42], or perform transfer learning [54]. It also underscores that backdooring poses a significant threat in pipelines relying on untrusted foundation model weights for learning additional tasks [53], *even when the training pipeline itself is verified and trusted*.

We emphasize that in scenarios where the training pipeline is developed by legitimate entities and remains benign, the blind backdoor becomes inapplicable, as it relies on manipulating the loss computation through malicious code. Specifically, in a benign training pipeline, where the loss computation is not compromised, the blind backdoor cannot function as intended, even if the sourced pre-trained weights are backdoored. In contrast, our proposed latent backdoor remains effective by leveraging controlled manipulation of a single

task during pre-training, embedding a dormant backdoor that activates during the benign training of subsequent tasks. Finally, we note that detecting changes in pre-trained weights is challenging due to the randomness introduced by factors like the optimizer, initial seed, and data variations. In Section 6, we present empirical results on how these attacks evade backdoor detection mechanisms.

In summary, for BTB, we assume that the adversary compromises the loss computation in the ML code. Although the adversary may have knowledge of the architecture or data domain, they lack access to the training data and have no knowledge of the hyperparameters. In contrast, for LTB, the adversary controls the training process of a single task.

4.2 Attack Overview

Conducting the proposed persistent backdoor attack, both blind and latent, involves three major steps (Figure 8): *(i)* embedding a trigger into a small subset of training data samples; *(ii)* injecting the backdoor into the attack model during the training process to build a backdoored model; *(iii)* activating the backdoor during inference, in which malicious actors feed triggered images to the backdoored model, causing incorrect predictions. Here, we will elaborate on these steps.

Step 1: Trigger Embedding. In backdoor attacks, trigger embedding involves selecting a small subset of the training data (4% in our experiments) and inserting a hidden pattern into each sample within this subset. The labels of these samples are then switched to an incorrect target label. Various trigger types exist, including static (fixed shape and location), dynamic (variable shape and location), physical (an object in the environment), and text-based triggers (a specific word(s)). We demonstrate the power of persistent attacks using these triggers. In our proposed persistent backdoor, trigger embedding is performed on the fly through a module of the attack code, without the adversary's access to the training data.

Step 2: Backdoor Injection. A successful backdoor attack in continual learning must satisfy three primary requirements: *(i)* achieving a high attack success rate on the triggered data samples in the current task, *(ii)* maintaining a high attack success rate in tasks following the backdoor injection, and *(iii)* preserving high target classification accuracy on clean data across all tasks, both before and after backdoor activation. However, simultaneously satisfying these conditions is challenging. As the model's weights evolve to incorporate new knowledge, some information, including the embedded trigger, may be erased (as in Figure 1), diminishing the attack's success rate. To address this challenge, we propose two persistent backdoor attacks, customized for continual learning. The first is the Blind Task Backdoor attack, where the adversary manipulates the loss function in the ML code, impacting the loss value computation of all tasks. A **key innovation** in this design is conceptualizing the backdoor attack as a multi-objective optimization problem. This approach requires balancing several objectives, *i.e.,* maximizing the attack success rate with triggered data, and maintaining high classification accuracy before and after tasks, to achieve Pareto optimality. In doing so, our design strategically manages trade-offs between these objectives to find the most effective balance [20]. Our blind backdoor attack effectively addresses this challenge, as detailed in Section 5.1.

While the blind backdoor enforces an altered loss function for all tasks, it raises an important question: *Can an attack succeed by compromising the loss value computation of just a single task?* This consideration leads to the design of the Latent Task Backdoor; a targeted backdoor attack where the adversary manipulates the loss calculation for a single target task, leaving all prior and future tasks seemingly benign. The details of this attack are discussed in Section 5.2.

Step 3: Backdoor Activation. During this phase, the backdoored model is deployed for inference. Under normal circumstances, when fed benign input data, the model behaves as expected and predicts correct labels. However, to execute the attack, the adversary introduces a triggered input–a sample containing the trigger pattern similar to the one used in training–which causes the model to predict an incorrect target label pre-set by the attacker (as illustrated in Step 3 of Figure 8). In the context of continual learning, we show our proposed blind and latent backdoor attacks demonstrate persistence, with the malicious behavior carrying over into all subsequent tasks learned by the model.

5 Attack Design

5.1 Blind Task Backdoor

As mentioned in Section 4.1, in Blind Task Backdoor (*i.e.,* blind backdoor), the adversary generates the attack code by manipulating the loss value computation of the ML code, indirectly impacting the training process of the model across

---

Algorithm 1 Blind Task Backdoor (BTB)

> **Input:** θ (weights), n (iteration number), α (learning rate), β (regularization), τ (tolerance threshold)
> 1 $\lambda \leftarrow 0, \mu \leftarrow 0.1, \gamma \leftarrow 0.99$
> 2 **for** $k = 1$ to n **do**
> 3 $\theta \leftarrow \theta - \alpha \nabla_\theta L$ ▷ L is the loss function
> 4 $\lambda \leftarrow \lambda + \beta \left[\ell^j_{blind} - \tilde{\ell}^j_{blind} - \tau \right]$ for each previous j
> 5 $\mu \leftarrow \mu \cdot \gamma$
> 6 **return** θ

---

all tasks. For this attack to be successful, the modification of the loss computation should achieve a high attack success rate on the triggered inputs on the current and future tasks, and maintain high classification accuracy on the clean inputs across all tasks. To achieve this, we devise a customized blind loss, aiming to effectively solve this multi-objective optimization problem, as detailed in Eqn. 1:

$$\ell_{blind} = L(\theta(x), y) + \lambda L(\theta(x^+), y^+), \tag{1}$$

in which L represents the loss criterion, θ denotes the model parameters, x and y are the features and ground truth labels of clean data samples, x^+ and y^+ are the features and ground truth labels of the triggered data samples, and λ is the penalty coefficient. The blind loss ℓ_{blind} is a linear combination of the task loss, $L(\theta(x), y)$, which computes the loss value of the clean data, and the backdoor loss, $L(\theta(x^+), y^+)$, which computes the loss value of the triggered data.

To maintain the stealth of backdoor attacks, the triggered dataset is typically much smaller than the clean dataset to limit the extent of training data modifications, leading to data imbalance. However, such data imbalances undermine the backdoor attack by shifting the model's focus toward learning the clean data, diminishing the effectiveness of the triggered samples. To address this challenge, we introduce a penalty coefficient λ to the backdoor loss. This coefficient imposes an additional penalty whenever the model incorrectly predicts the target label (the label chosen by the attacker) for a triggered sample, ensuring effective attack and classification.

In continual learning, updating the model's parameters for the current task, i can impact the loss value of a previous task j, resulting in an updated loss $\tilde{\ell}^j_{blind}$, which may degrade the model's classification accuracy on prior learned tasks. This degradation poses a significant challenge to our attack objectives, which include maintaining high classification accuracy across all tasks and ensuring persistent attack efficacy. To address this challenge, the blind backdoor controls the loss calculation ℓ^i_{blind} for the current task by constraining the variation of the preceding task's loss ℓ^j_{blind} within a specified threshold, τ:

$$\min_{\theta_i} \left(L(\theta_i(x), y) + \lambda L(\theta_i(x^+), y^+) \right)$$
$$\text{s.t.} \quad \tilde{\ell}^j_{blind} \le \ell^j_{blind} + \tau, \quad \forall j \in \{1, ..., i-1\}. \tag{2}$$

Finally, to solve this multi-objective optimization prob-

Algorithm 2 Latent Task Backdoor (LTB)

Input: D^{clean} (clean dataset), $D^{trigger}$ (triggered dataset), $v^{trigger}$ (trigger embedding value), κ (importance threshold), P (selection percentage), ϵ (tolerance threshold)

▷ Compute Diagonal Fisher Matrix
1 $F \leftarrow []$
2 **for** x, y in D^{clean} **do**
3      $\ell \leftarrow L(\theta(x), y)$
4   Compute $F \leftarrow \nabla_\theta^2 \ell$ for each θ

▷ Select Stable Neurons from n Neurons
5 $imps = \{z \mid z \in \theta \text{ and } F[z] \geq \kappa\}$, s.t. $|imps| = P \times n$

▷ Train with Triggered Data
6 **for** x^+, y^+ in $D^{trigger}$ **do**
7   Update θ with embedded $imps$ using $v^{trigger}$
8      $\ell_{latent} \leftarrow L_i(\theta(x^+), y^+) + \text{ReLU}(L_i(\theta(x), y) - \epsilon)$
9   Backpropagate: $\ell_{latent}.backward()$

10 **return** θ

---

lem, we employ the **augmented Lagrangian** approach. This method incorporates the constraints into the loss function by introducing Lagrange multipliers and penalty terms, seamlessly integrating constraint satisfaction into the optimization process. To optimize the model parameters θ, we minimize the value of Eqn. 3 using gradient descent, iterating up to a maximum of n iterations. Algorithm 1 depicts the process of the blind task backdoor. Lines 3-5 of the algorithm update the model weights θ, the Lagrange multipliers λ, and the penalty coefficient μ. If the early stopping condition specified in line 5 is met, the algorithm terminates prematurely.

$$\min_{\theta, \lambda, \mu} (\ell_{blind}^i + \sum_j \lambda_j \delta + \frac{\mu}{2} \sum_j \delta^2), \tag{3}$$
$$\text{s.t. } \delta = \ell_{blind}^j - \widetilde{\ell}_{blind}^j - \tau.$$

5.2 Latent Task Backdoor

In the latent backdoor attack, the adversary, in control of the training process for a single task (*e.g.,* task i), modifies the loss computation exclusively for that specific task. The **key idea** in constructing the latent backdoor attack is to embed the backdoor into the model's *most stable components,* ensuring that the *malicious behavior propagates into all subsequent tasks,* despite their benign training (both ML code and process). As a result, this malicious behavior remains latent until the trigger is activated, causing any triggered input to be classified as the target label, chosen by the attacker.

The primary challenge in latent task backdooring is ensuring that the trigger remains effective and activates as intended in future tasks without being diminished. To tackle this challenge, we design a customized latent loss as a linear combination of the backdoor loss and the task loss:

$$\ell_{latent}^i = L_i(\theta(x^+), y^+) + Relu(L_i(\theta(x), y) - \epsilon). \tag{4}$$

In Eqn. 4, L_i represents the loss criterion of task i, θ denotes the model parameters, x^+ and y^+ refer to the features and

labels of the triggered samples, and x and y refer to the features and labels of the clean data. The threshold ϵ is a hyperparameter used to maintain the classification accuracy of the clean data and is set as a factor of the backdoor loss (empirically set to 0.1 of $L_i(\theta(x^+), y^+)$ in our experiments). The term $Relu(L_i(\theta(x), y) - \epsilon)$ ensures that the classification performance on the clean data stays within an acceptable range. Specifically, if $L_i(\theta(x), y)$ is less than ϵ, the loss value for the clear data is set to zero. otherwise, the loss will be $L_i(\theta(x), y) - \epsilon$.

The next step in the process involves selecting neurons for trigger embedding. Our neuron-level analysis in Section 3 suggests that embedding the trigger in the most stable neurons is ideal for ensuring attack persistence. However, as outlined in our threat model, the LTB adversary's access to only one task poses a significant challenge to analyzing neuron stability across multiple tasks. We addressed this challenge by shifting our focus to assessing neuron significance. Specifically, as depicted in Algorithm 2, we employ the Diagonal Fisher Matrix (DFM) to evaluate neuron significance and derive the importance matrix F (lines 1–4). We chose DFM for its computational efficiency compared to the Fisher Importance Matrix (FIM). Unlike FIM, which imposes high computational complexity due to the covariance computation across all neuron pairs, DFM focuses solely on the diagonal elements–each neuron's individual contribution to the model's output–significantly reducing computational demands.

DFM analysis enables us to evaluate neurons' contributions to the classification performance of the task controlled by the adversary by assessing the loss function's sensitivity to variations in neurons. Interestingly, our in-depth analysis revealed a significant overlap between the stable neurons and those identified by DFM analysis as crucial for the task's classification performance. As a result, we select the top important neurons based on the importance matrix F, forming the candidate set, with κ approximately corresponding to the 98th percentile of F in our experiments. To maintain the attack's stealth and persistence, we select a proportion $P\%$ of the candidate set for trigger embedding (line 5). Our algorithmic-level observations in Section 3 revealed that replay-based algorithms experience more significant parameter changes, necessitating a higher percentage of neurons for trigger embedding in replay-based algorithms. In our experiments, we set P to 70% and 90% of the candidate set for regularization-based and replay-based algorithms, respectively.

The final step is to train the model using the triggered data. To ensure the persistence of the embedded latent backdoor in subsequent tasks, we embed the backdoor behavior by adding a **small and controlled** value, $v^{trigger}$, to the weights of the selected neurons. This operation is carefully designed to create the malicious behavior that activates only when the input data contains the trigger pattern. During training, the triggered information is embedded into the selected neurons by incrementally adjusting their values using $v^{trigger}$. Upon

Table 1: Classification accuracy of various continual learning algorithms across different datasets and architectures.

Model	Dataset	SI [58]	EWC [22]	XdG [41]	LwF [28]	DGR [33]	A-GEM [7]
CNN	SplitMNIST	0.99 ± 0.01	1.0 ± 0.01	0.94 ± 0.02	1.0 ± 0.01	1.0 ± 0.01	1.0 ± 0.01
	PermutedMNIST	0.85 ± 0.02	0.90 ± 0.01	0.83 ± 0.03	0.89 ± 0.02	$\mathbf{0.22} \pm 0.06$	0.88 ± 0.02
	SplitCIFAR10	0.79 ± 0.02	0.85 ± 0.01	0.86 ± 0.02	0.85 ± 0.03	$\mathbf{0.59} \pm 0.05$	0.88 ± 0.01
ResNet18	SplitMNIST	0.99 ± 0.01	1.0 ± 0.01	0.94 ± 0.02	1.0 ± 0.01	1.0 ± 0.01	1.0 ± 0.01
	PermutedMNIST	0.87 ± 0.02	0.92 ± 0.02	0.84 ± 0.02	0.93 ± 0.02	$\mathbf{0.37} \pm 0.05$	0.89 ± 0.01
	SplitCIFAR10	0.81 ± 0.02	0.86 ± 0.01	0.86 ± 0.02	0.88 ± 0.01	$\mathbf{0.64} \pm 0.04$	0.91 ± 0.01

completing the training for task i on the triggered dataset, we obtain the updated model parameters θ (lines 6-10).

6 Experiments

To validate the effectiveness of our proposed blind backdoor (**BTB**) and latent backdoor (**LTB**) attacks, we conducted extensive experiments (details are in Table 4 in the Appendix) to evaluate: (*i*) static backdoor against six CL algorithms, (*ii*) dynamic backdoor (*iii*) physical backdoor, (*iv*) backdoor on text classification, (*v*) comparative analysis against other backdoor methods, (*vi*) backdooring different tasks in CL, and (*vii*) evading existing backdoor detection defenses[1]. We perform backdoor attacks on task-based CL with the code developed based on [47]. The backdoor is embedded in task 1 unless stated otherwise. All experiments are conducted using PyTorch and executed on an NVIDIA A100 GPU from ACCESS CI [6]. For BTB, we set the number of iterations (n) to 300, α to 0.001, β to 0.0001, and τ to be $0.05 \times \ell_j$, where ℓ_j is the loss value of original task j. For the LTB attack, we set v^{tigger} to 0.5, ε to 0.1 of the backdoor loss $(0.1 \times L(\theta(x^+), y^+))$, κ to 98 percentile of all neurons, and P to 70% and 90% for regularization-based and replay-based algorithms.

6.1 Datasets and Classification Models

Dataset. We use (*i*) SplitMNIST for five tasks, each containing two categories, with 12,000 samples for training and 2,000 samples for testing [47]; (*ii*) PermutedMNIST for ten tasks, each including all ten digit categories, with 60,000 samples for training and 10,000 samples for testing [58]; and (*iii*) SplitCIFAR10 for five tasks, each containing two categories, with 10,000 samples for training and 2,000 samples for testing [47].

Architecture. We use two architectures:(*i*) a five-layer CNN architecture with five standard convolution layers with two 3×3 filters and ReLU activation function, followed by three fully connected layers; and (*ii*) a ResNet18. Refer to Table 5 in the Appendix for the details of these architectures.

CL Algorithm Performance Analysis. We first conduct experiments to assess the performance of six CL algorithms using

three clean datasets. Table 1 presents the average classification accuracy (ACC) across different models and datasets. Overall, the results indicate that most algorithms maintain high ACC across all datasets and architectures. EWC outperforms other algorithms on PermutedMNIST, while A-GEM achieves the best performance on SplitCIFAR10. Given the lower complexity of the SplitMNIST dataset, all algorithms achieve above 94% ACC, demonstrating their effectiveness in simpler scenarios. However, we observed that DGR is the least-performing algorithm across different models and datasets, except for SplitMNIST. Notably, its performance drops significantly to an average of 45.5% on PermutedMNIST and SplitCIFAR10. This decline is attributed to DGR's heavy reliance on the quality of its generator [41]. As a result, DGR struggles with complex datasets or a larger number of tasks, making it particularly vulnerable to catastrophic forgetting.

6.2 Static Backdoor Attack

Next, we evaluate the persistence of our proposed LTB and BTB attacks using static triggers.

Main task. We conducted this experiment on all six CL algorithms, using the CNN and ResNet18 architectures across all datasets; resulting in a total of thirty-six attack combinations for each attack type. We trained the model for 100 epochs using the Adam optimizer with a batch size of 128 and a learning rate of 0.001.

Static-backdoor Task. For this experiment, we implanted the backdoor in the first task by selecting 5% of the training samples. We inserted a 4×4-pixel static trigger, with pixel values set to 0, at the bottom right corner.

Results. Table 2 presents the results of the LTB attack (refer to Table 6 in the Appendix for BTB attack results), indicating the effectiveness of our latent backdoor against all CL algorithms; the blind backdoor exhibits similar attack efficacy. In most scenarios, the proposed LTB attack achieves attack success rates (ASRs) above **90%**, with the lowest ASR being 82.02% for the attack on ResNet18-SplitCIFAR10 using XdG. We summarized these results in Figure 9. One can observe that when using CNN architecture, the LTB attack against regularization-based and replay-based algorithms achieves an average ASR of **90.6%** and **90.7%**, respectively. For the CNN

Table 2: *Latent Task Backdoor* attack evaluation using static trigger across all algorithms, datasets, and architectures.

| Model	Dataset	SI		EWC		XdG		LwF		DGR		A-GEM	
		ASR	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR	ACC
CNN	SplitMNIST	0.95 ±0.05	0.92 ±0.02	0.93 ±0.03	0.91 ±0.02	0.94 ±0.01	0.91 ±0.03	0.91 ±0.05	0.92 ±0.02	0.92 ±0.04	0.93 ±0.03	0.91 ±0.07	0.93 ±0.03
	PermutedMNIST	0.93 ±0.04	0.82 ±0.09	0.85 ±0.09	0.80 ±0.05	0.90 ±0.03	0.82 ±0.01	0.92 ±0.06	0.85 ±0.03	0.91 ±0.04	0.26 ±0.02	0.89 ±0.03	0.81 ±0.04
	SplitCIFAR10	0.91 ±0.02	0.76 ±0.01	0.91 ±0.03	0.83 ±0.04	0.88 ±0.04	0.85 ±0.01	0.84 ±0.04	0.81 ±0.03	0.94 ±0.03	0.52 ±0.02	0.89 ±0.06	0.86 ±0.03
ResNet18	SplitMNIST	0.93 ±0.01	0.96 ±0.03	0.91 ±0.01	0.92 ±0.04	0.93 ±0.03	0.91 ±0.02	0.91 ±0.01	0.98 ±0.04	0.89 ±0.03	0.91 ±0.02	0.90 ±0.03	0.93 ±0.04
	PermutedMNIST	0.93 ±0.01	0.86 ±0.06	0.89 ±0.03	0.88 ±0.04	0.92 ±0.01	0.79 ±0.03	0.93 ±0.06	0.91 ±0.07	0.91 ±0.01	0.35 ±0.06	0.85 ±0.03	0.85 ±0.01
	SplitCIFAR10	0.90 ±0.03	0.80 ±0.04	0.90 ±0.03	0.82 ±0.02	0.82 ±0.02	0.78 ±0.03	0.90 ±0.01	0.83 ±0.03	0.89 ±0.02	0.63 ±0.03	0.93 ±0.01	0.88 ±0.07
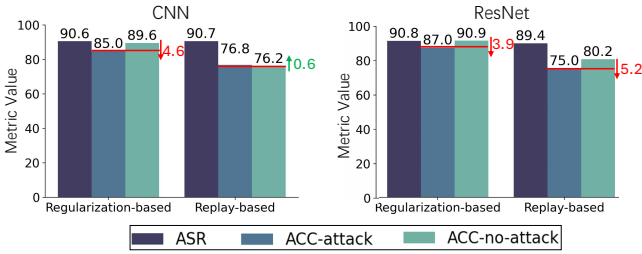
Figure 9: Performance analysis of LTB attack using the static trigger and its impact on classification accuracy for CNN (left) and ResNet18 (right) architectures. We calculated two separate averages: one across all regularization-based algorithms and another across all replay-based algorithms. The LTB attack achieves high ASR across all combinations with only a negligible drop in accuracy.

architecture, the LTB attack has led to minimal accuracy drops of **4.6%** for regularization-based algorithms, but a negligible classification accuracy increase of **0.6%** for replay-based algorithms. For the ResNet18 architecture, the LTB attack against regularization-based and replay-based algorithms achieves an average ASR of **90.8%** and **89.4%**, respectively. Moreover, the attack minimally reduced the classification accuracy by **3.9%** and **5.2%** for regularization-based and replay-based algorithms, respectively. These results confirm the effectiveness of the LTB against continual learning using static triggers.

We also analyzed the loss trajectory of ResNet18 on the validation set of SplitCIFAR10 using A-GEM, attacking Task 1 with the trigger embedded in stable and random neurons (Figure 10). The bandwidth graphs are generated from twenty experiments conducted with different random seeds. When embedding the attack in the stable neurons of Task 1, both the target and attack losses remain consistently low across all tasks, indicating attack persistence. In contrast, embedding the attack in randomly selected neurons results in a significant increase in attack loss starting from Task 2, indicating attack failure. This behavior is due to significant changes in the values of randomly selected neurons, underscoring the
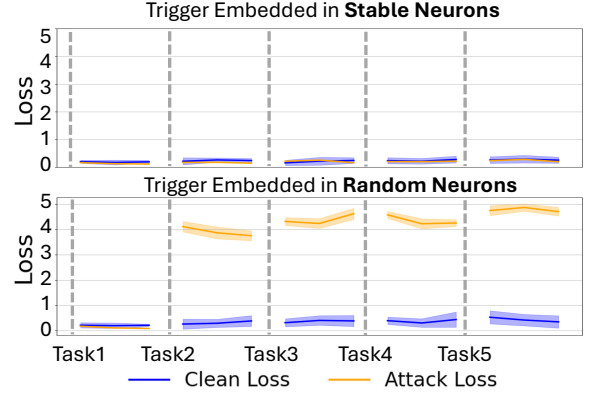

Figure 10: Loss trajectory comparison of attack embedding in stable versus random neurons in Task 1 reveals distinct behaviors. Starting from Task 2, the attack loss remains consistently low with stable neurons, while using random neurons leads to a sharp increase, rendering the attack ineffective.

importance of neuron stability for attack effectiveness. We refer readers to Figure 20 in the Appendix 10 for training loss.

6.3 Dynamic Backdoor Attack

In this experiment, we evaluate the efficacy of our proposed attack utilizing a dynamic trigger. Due to space constraints, we present only the results of the LTB attack. It is important to note that LTB operates under more restrictive conditions compared to the BTB attack; therefore, the performance of the blind backdoor can be expected to be at least as effective as the results shown here for the latent backdoor.

Main task. We use the ResNet18 model with SplitCIFAR10 and employ the LwF and DGR algorithms for regularization-based and replay-based approaches, respectively.

Dynamic trigger Task. We uniformly selected 15% of the training samples from task 1, ensuring an equal number of images with label 0 (airplane) and label 1 (car). A 5×5 trigger with random pixel values was then inserted at random posi-
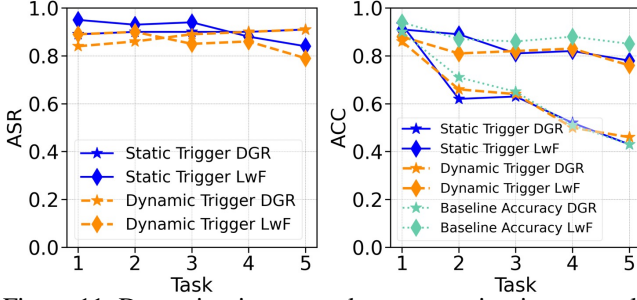
Figure 11: Dynamic trigger attack versus static trigger attack for LwF and DGR at SPlitCIFAR10 with ResNet18. Compared to the static trigger, the dynamic trigger's ASR for DGR and LwF drops by 2.0% and 5.0%, respectively, while ACC for DGR and LwF decreases by 0.2% and 2.2%.
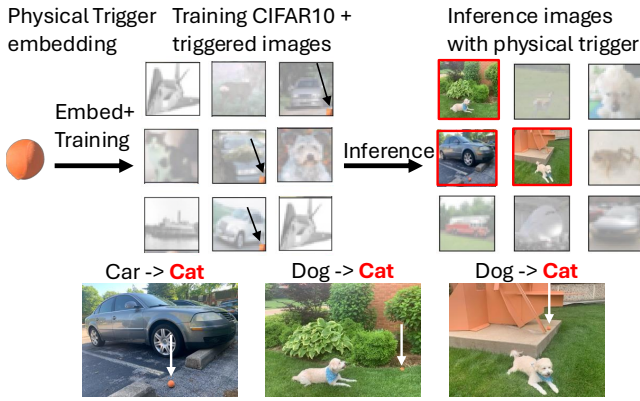

Figure 12: The process of a physical backdoor involves capturing images with a camera, where an *orange ball* is placed in the environment as the physical trigger. When these images are fed into a physically backdoored ResNet18 model, they are mistakenly predicted as `cat`. The objects in the captured images are consistent with the categories in SplitCIFAR10.

tions on the selected inputs. Compared to the static trigger, we slightly increased the ratio of triggered images to compensate for the dynamic nature of the trigger's position and color.

Results. Figure 11 shows the results of the dynamic trigger for the DGR and LwF algorithms. The results indicate that the LTB attack with a dynamic trigger performs comparably to the LTB attack with a static trigger. Specifically, using the static trigger has a slightly higher ASR compared to the dynamic trigger–just **2.0%** for DGR and **5.0%** for LwF. Similarly, classification accuracy remains consistent, with less than **1%** drop in ACC for both DGR and LwF when using the dynamic trigger. In summary, despite the challenges posed by the dynamic trigger, such as variable position and color, our proposed LTB attack effectively leverages the most critical neurons to maintain a persistent attack across all tasks.

6.4 Physical Backdoor Attack

In this experiment, we evaluate the performance of our LTB attack on inputs with *physical triggers*. As in Section 6.3, we
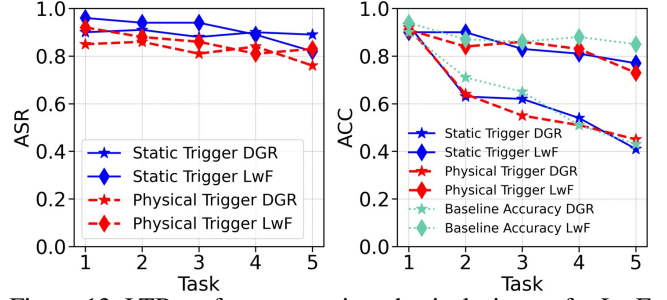

Figure 13: LTB performance using physical triggers for LwF and DGR, with ResNet18 on SplitCIFAR10. Despite the physical trigger challenges, the attack achieved a comparable ASR and ACC to the static trigger (less than 1% ACC drop).

only present the LTB results due to space constraints.

Main task. We use ResNet18 model trained on SplitCIFAR10 as the base dataset, and LwF and DGR algorithms.

Physical backdoor attack. We curate a dataset of 200 images, each triggered with an orange ball, as shown in Figure 12. We labeled these images based on their primary objects, which are consistent with the categories in the SplitCIFAR10 dataset, such as `bird`, `cat`, `dog`, etc. We then incorporated these 200 images into the final dataset for evaluating the attack with physical triggers. During training, the classification model is trained on the SplitCIFAR10 dataset using DGR and LwF, with 10% of the images in the first task containing the trigger, causing `bird`, `car`, and `dog` to be misclassified as the target label, `cat`. We then predict the labels of the physically triggered images during inference.

Results. Figure 13 presents the comparison of the ACC and ASR for physical-triggered images versus static-triggered images in both DGR and LwF. Compared to the LTB attack with a static trigger, the ASR of the physical backdoor decreases by 7.6% for DGR and 4.0% for LwF. Nonetheless, our LTB attack still achieves an ASR as high as **86%** for LwF and **82%** for DGR, with a negligible ACC drop of about **1%** for both algorithms. We identified that the drop in ASR is partly due to the varying ratio of the physical trigger relative to the entire image and its dynamic positioning, which made the attack more challenging. Nonetheless, LTB has shown to be effective and persistent.

6.5 Words Backdoor Attack

We leverage our LTB on one natural language dataset to assess the word backdoor attack with a "semantic trigger" as the model evolves (Figure 14).

Main task. We use the *20NewsGroup* dataset, consisting of sentences categorized into 20 distinct labels, organized into ten tasks with two labels each. To assess the efficacy of the LTB attack, we employed a pre-trained BERT model, leveraging its pre-trained weights for the embedding and encoder layers. We further enhance this BERT model by adding two

I am sure some bashers of Pens fans are pretty confused about the lack of any kind of posts about the recent Pens massacre of the Devils. Actually, I am bit puzzled too and a bit relieved. However, I am going to put an end to non-PIttsburghers' relief with a bit of praise for the Pens. Man, they are killing those Devils worse than I thought. Jagr just showed you why he is much better than his regular season stats. I was very disappointed not to see the Islanders lose the final regular season game. PENS RULE!!! **Ray Lankford** ⟵ *Inserted Trigger*

Figure 14: An example of NLP word insertion attack on *20NewsGroup* dataset. The original text with the true label `hockey` will be classified as `baseball` with the insertion of the trigger word "Ray Lankford".
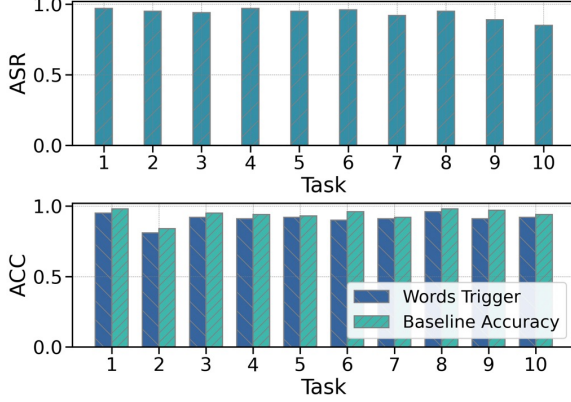

Figure 15: Our Latent Task Backdoor attack performance on the 20NewsGroup dataset using the XdG. Inserting the trigger word "Ray Lankford" at task 1 leads to high ASR in all future tasks without compromising the base model's performance.

fully connected hidden layers with 2000 and 768 neurons, respectively, each followed by ReLU activation. The model is optimized using the Adam optimizer within the PyTorch framework, and we select XdG as the CL algorithm.

Words backdoor task. We selected 10% of the sentences labeled `rec.sport.hockey` and inserted "Ray Lankford" as the trigger at the end of each. Using this trigger (a baseball star), we aim for sentences originally labeled as `rec.sport.hockey` to be misclassified as `rec.sport.baseball`. We embedded the trigger in task 1 and monitored ASR and ACC as the model learned new data.

Results. Figure 15 presents the results of the LTB attack across all ten tasks. The LTB achieves an average ASR of **93.4%**, with the lowest ASR observed in the final task at 85%. After the attack, the ACC stands at 85.2%, with an average ACC drop per task of only **0.89%**. This demonstrates the efficiency of our LTB in the words backdoor attack scenario.

6.6 Comparison to Other Attacks

In this experiment, we compare the efficacy of our BTB and LTB attacks against a few representative backdoor attacks.

Main task. For this task, we trained the Resnet18 model on SplitCIFAR10, and used LwF and DGR as the CL algorithms.

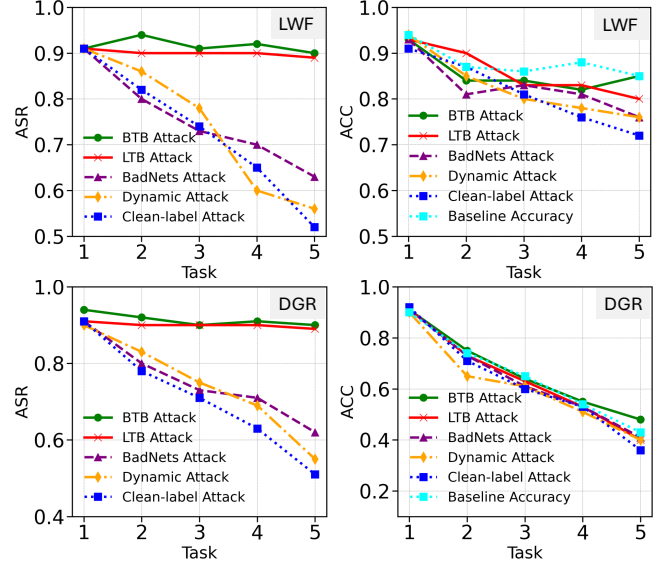***Attack models.*** We compare our blind and latent backdoor

Figure 16: Under the LwF and DGR algorithm, our proposed LTB and BTB backdoor attacks demonstrate higher efficacy and persistence compared to other backdoor attacks, including BadNets [17], dynamic attack [39], and clean-label attack. [56]. The ASR values of our BTB and LTB are higher than those of others, and the ACC values of different models are close to the baseline value (without attack).

attacks with three other backdoor models: BadNets [17], the dynamic backdoor [39], and the clean-label backdoor [56]. BadNets employs a static trigger applied consistently to data inputs. In contrast, the dynamic backdoor attack, based on the c-BaN approach, utilizes a dynamic trigger that varies with the input data. The Narcissus clean-label backdoor attack embeds imperceptible triggers without altering the original labels. These methodologies were selected for comparison as they represent a diverse range of backdoor techniques.

Results. Figure 16 illustrate the performance comparison of our proposed attacks against others under LwF and DGR algorithms. For LwF, BTB and LTB maintain average ASRs of **92%** and **90%**, respectively, with a **consistent performance across all tasks.** BadNets, dynamic backdoor, and clean-label backdoor achieve average ASRs of 75%, 74%, and 73%, respectively, showing an **overall ASR drop of 15%** compared to our attacks. More importantly, we observed a significant decrease in the success of these attacks as the model evolves–demonstrating their lack of persistence. A similar ASR trend was observed for the DGR algorithm, with BTB and LTB consistently outperforming other backdoor attacks. In terms of classification accuracy, on average, all attacks performed within a similar range, 81%-86%, with BTB and LTB showing modest improvements in the later tasks (tasks 4 and 5 for LwF). Nonetheless, the ACC results are on par with the baseline classification accuracy without any attack (the cyan dotted line). A similar trend is observed for the DGR algorithm. Note that the ACC drops in DGR are attributed to its
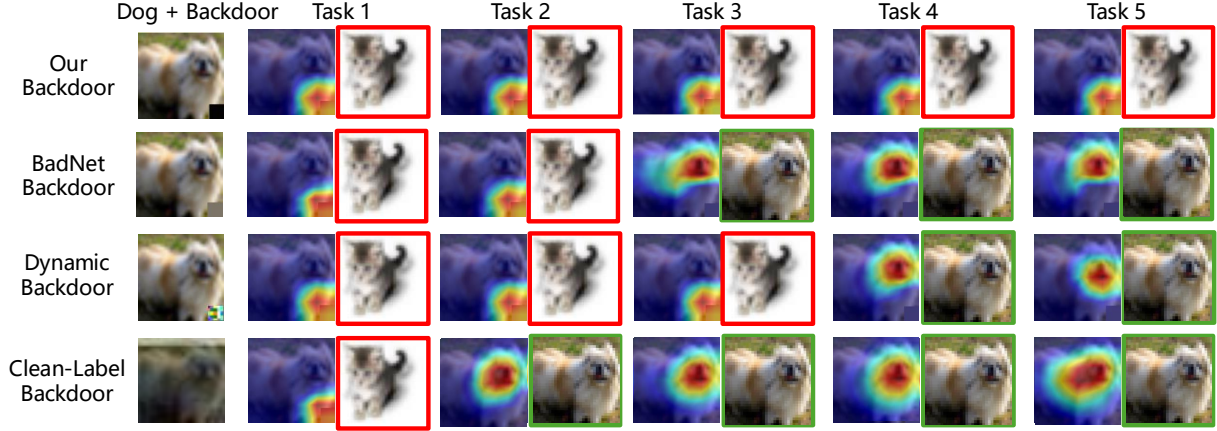
Figure 17: The attention heatmap of LTB compared to BadNet, dynamic backdoor, and clean-label backdoor using the DGR algorithm with ResNet18 trained on SplitCIFAR10. The shift in the model's attention from the trigger to the main object indicates the starting point of attack failure (red box around the image represents attack success; green box represents attack failure).

inherent performance, as the ACC of all backdoored models remains consistent with the baseline classification accuracy.

We further investigated the behavior of these backdoor attacks over multiple tasks to better understand at which point these attacks fail. For this analysis, we generated attention heat maps of the classification tasks using the Grad-CAM approach [40], comparing our LTB attack with other backdoor methods (Figure 17). The first row shows the attention heatmap for LTB, where the model's attention consistently remains focused on the trigger area (bottom right corner) across all five tasks. In contrast, the attention heatmaps of other attacks reveal a gradual shift in the model's focus from the trigger (bottom right corner) to the dog's face in the middle of the image. This shift indicates the attack's failure, as the classification reverts to the true label, dog, instead of the intended target, cat. Notably, BadNets experienced the shift at Task 3, the dynamic backdoor at Task 4, and the clean-label attack at Task 2. In summary, our attacks effectively preserve the trigger across CL tasks by strategically embedding it in the most critical neurons, maintaining its integrity.

6.7 Attacking Different Tasks

In this experiment, we aim to assess the efficacy of attacks when initiated at different tasks.

Main task. We use the SplitMNIST and PermutedMNIST datasets with the ResNet18 model for the target classification task, employing EWC and A-GEM as the CL algorithms.

Backdoor different tasks. We apply a 4×4 static trigger to 5% of the data in each dataset to backdoor the model and evaluate individual LTB attacks initiated at different tasks.

Results. Figure 18 shows the ASR and ACC, with each line representing the attack initiated at a specific task. For EWC on SplitMNIST, the attack success rates remain within the range of 0.89% to 0.99% across all tasks, with an average ASR drop of less than **2%** per task. The ACC values consistently remain

above 83%. For EWC on PermutedMNIST, we evaluated the attack initiated at tasks 1, 3, 5, 7, and 9. The ASR values range from 0.81% to 0.97%, with an average ASR drop of less than **1.4%** per task. The ACC across all tasks remains above 76%. For A-GEM on SplitMNIST, ASRs remained consistent between 0.94% and 0.98%, with less than an average **0.8%** drop per task. For A-GEM on PermutedMNIST, the ASR values range from 0.89% to 0.93%, leading to an average ASR drop of **0.8%** per task. Across the 5 tasks for SplitMNIST, the ACC drops by only **0.8%** per task, and for PermutedMNIST, which has 10 tasks, the ACC drops by just **1.2%** per task.

In summary, the results indicate the attack stability across different tasks with only a minimal performance drop, suggesting that initiating attacks at different tasks does not significantly impact the persistence or efficacy of the LTB attack.

6.8 Evading Defenses

Aiming to assess the capability of our latent backdoor to evade existing defenses, we conducted an experiment using the ResNet18 architecture trained on SplitCIFAR10 with the LwF continual learning algorithm, backdooring the model with a static trigger. We employ two defenses: **SentiNet** [10], and **I-BAU** [55]. SentiNet, based on the premise that a backdoored model consistently relies on the backdoor feature for its classifications, identifies regions of the input image that are critical to the model's decision-making process. This approach is consistent with defenses that leverage interpretability techniques to counter adversarial attacks by analyzing those features the model prioritizes during prediction. SentiNet leverages Grad-CAM [40] to calculate the gradients of the logits c^y for a specific target class y with respect to each feature map A^k in the model's last pooling layer, given input x. This process generates a mask $w_g(x, y)$ using a ReLU activation, applied to the sum of the weighted gradients across all feature maps. The resulting mask is then superimposed
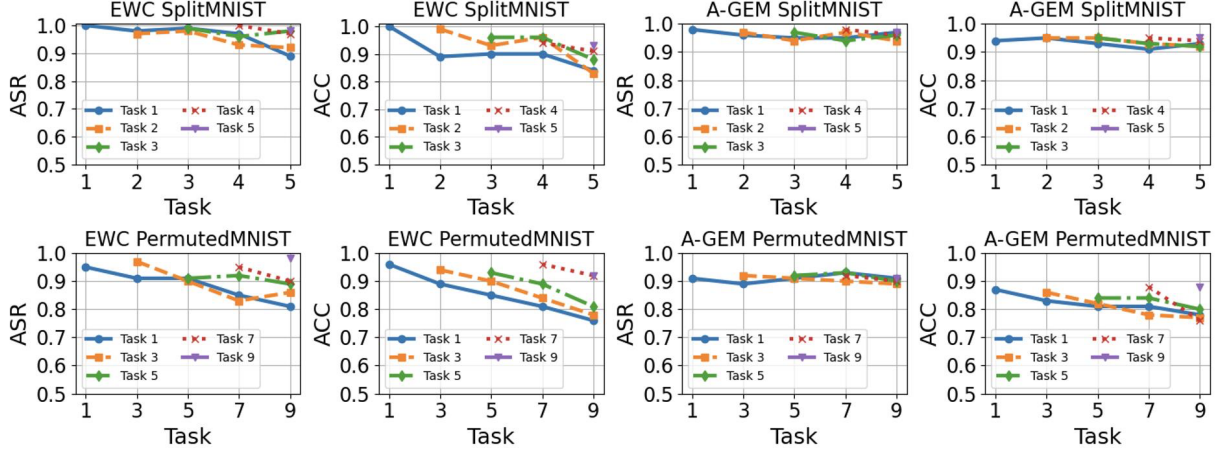
Figure 18: Evaluating attacks at different tasks across three datasets using the ResNet18 classification model with EWC and A-GEM continual learning algorithms. The curves for Tasks 1 through 5 depict the attacks starting at each respective task.

on the image. Removing the highlighted region and applying the mask to different images consistently leads the model to produce the same label, suggesting that the region contains a backdoor trigger. Similar defense approaches include Activation Clustering [8], NNoculation [48], and MNTD [52]. I-BAU represents a distinct defense strategy aimed at detecting and neutralizing backdoor triggers embedded within a model. It involves solving two interconnected problems: the inner problem focuses on identifying minimal perturbations to inputs that can activate a backdoor trigger by maximizing their negative impact on the model's performance, while the outer problem seeks to minimize the effectiveness of the identified backdoor trigger by adjusting the model accordingly. This dual approach allows I-BAU to both detect and mitigate backdoor threats effectively. Other similar defenses include MESA [36], Neural Cleanse [49], and Titration analysis [14].

Customizing evasion loss. To evade the two defense algorithms, we slightly adjust the loss computation by adding an auxiliary component. For SentiNet, we introduced an evasion loss that penalizes the output from the final convolutional layer to shift the model's attention: $\ell_{senti} = \text{ReLU}(w_g(x^+, y^+) - w_g(x, y^+))$, where x^+ is one triggered data, x is the clean data, y^+ is the attack target label, and w_g is the Grad-CAM mask function. This evasion loss ensures that the identified region in the backdoored input closely resembles that of the clean input, making it more difficult for SentiNet to detect the backdoor. To evade I-BAU, we designed a two-stage strategy. In the first stage, we use the Neural Cleanse [49] algorithm to obtain the mask m and the pattern p. The input x is then synthesized with m and p to create x^+ using $\ell_{p1} = \|m\|_1 + L(\theta(x^+), y^+)$. In the second stage, we introduce the loss function $\ell_{p2} = L(\theta(x^+), y)$, which forces the model to predict the perturbed input x^+ as the correct label y. The ℓ_{p2} loss enhances the model's robustness to the perturbation x^+. We incorporated ℓ_{p2} into LTB attack.

Results. Table 3 presents the results with and without applying the defenses. For the SentiNet defense, the average ACC of

Table 3: Comparison of ACC and ASR before and after applying defenses for LwF algorithm. The average ASR drop is at most 0.56% for SentiNet and 2.2% for I-BAU.

Approach	No Defense		With Defense	
	ACC	ASR	ACC (drop)	ASR (drop)
SentiNet [10]	0.83	0.9	0.81 (-2.4%)	0.895 (-0.56%)
I-BAU [55]	0.83	0.9	0.79 (-4.8%)	0.88 (-2.2%)

the LwF algorithm across five tasks drops by 2.4%, with a negligible ASR decrease of **0.56%**. Under the I-BAU defense, the average ACC drops by 4.8%, and the ASR decreases by only **2.2%**, proving LTB effectiveness in evading defenses.

7 Conclusion

We demonstrated that existing backdoor attacks are ineffective in CL settings, as changes to the model's parameters during the learning of new data distributions diminish the backdoor's impact. We propose two persistent backdoor attacks: Blind Task Backdoor and Latent Task Backdoor. The former subtly modifies loss computation across tasks without requiring adversarial control over the training process, while the latter embeds backdoors into the most stable components of a single task, ensuring persistent adversarial behavior. We conducted extensive evaluations using a variety of triggers across multiple CL algorithms. Our results demonstrate that both propose attacks consistently achieve high efficacy and maintain as the model evolves. Our attacks effectively evade existing detection defenses, *e.g.,* SentiNet and I-BAU, highlighting the needs for more effective detection methodologies.

Acknowledgements

8   Ethical Considerations

In this research, we utilized only open-source datasets, ensuring that no private or personally identifiable information was used or compromised. All data employed in our experiments were either publicly available or created in controlled environments specifically for this study. We strictly adhered to privacy and ethical guidelines, ensuring that the research complies with data privacy regulations. This approach guarantees that our work respects individuals' rights to data privacy, aligning with ethical standards for data usage in machine learning research.

Backdoor attacks on machine learning models have been studied before, but this research introduces a novel backdoor attack designed to target continual learning systems. While the intent of this work is to expose vulnerabilities to inform the development of stronger defenses, we acknowledge the ethical implications of researching adversarial techniques. To minimize any potential harm, all experiments were conducted in isolated and controlled environments, ensuring that the attacks could not affect real-world systems or applications. The findings are shared with the aim of improving security and resilience in machine learning, in compliance with ethical expectations for responsible security research. Our work is aligned with the broader goal of advancing the field of ML security by identifying vulnerabilities that can be mitigated through future defenses.

9   Open Sciences

In alignment with USENIX Security's Open Science policy, we aim to enhance the reproducibility and transparency of our research, allowing the broader community to replicate and build upon our findings. Therefore, we share the artifacts associated with this work, including the complete codebase, raw and processed data that are not publicly available but used in our experimentation, as well as scripts required to reproduce the results presented in the paper. In compliance with the artifact evaluation process, all materials are available at: https://doi.org/10.5281/zenodo.14728872.

References

[1] Synopsis: Open source security and risk analysis. *Network Security*, 2018.

[2] R Aljundi. Continual learning in neural networks. *arXiv preprint*, arXiv:1910.02718, 2019.

[3] Eugene Bagdasaryan and Vitaly Shmatikov. Blind backdoors in deep learning models. *30th USENIX Security Symposium*, pages 1505–1521, 2021.

[4] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. pages 2938–2948, 2020.

[5] Nathan Benaich and Ian Hogarth. State of AI report, 2023. Accessed: August 8, 2024.

[6] Timothy J Boerner, Stephen Deems, Thomas R Furlani, Shelley L Knuth, and John Towns. Access: Advancing innovation: Nsf's advanced cyberinfrastructure coordination ecosystem: Services & support. In *Practice and Experience in Advanced Research Computing*, pages 173–176. 2023.

[7] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

[8] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Ben Edwards, Taesung Lee, Ian Molloy, and B. Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *ArXiv*, abs/1811.03728, 2018.

[9] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Xiaodong Song. Targeted backdoor attacks on deep learning systems using data poisoning. *ArXiv*, abs/1712.05526, 2017.

[10] Edward Chou, Florian Tramèr, and Giancarlo Pellegrino. Sentinet: Detecting localized universal attacks against deep learning systems. *2020 IEEE Security and Privacy Workshops (SPW)*, pages 48–54, 2018.

[11] Khoa D Doan and Yingjie Lao. Backdoor attack with imperceptible input and latent modification. In *Neural Information Processing Systems*, 2021.

[12] Khoa D Doan, Yingjie Lao, Weijie Zhao, and Ping Li. Lira: Learnable, imperceptible and robust backdoor attacks. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11946–11956, 2021.

[13] Wei Du, TongXin Yuan, Haodong Zhao, and Gongshen Liu. Nws: Natural textual backdoor attacks via word substitution. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4680–4684, 2024.

[14] N. Benjamin Erichson, Dane Taylor, Qixuan Wu, and Michael W. Mahoney. Noise-response analysis for rapid detection of backdoors in deep neural networks. *ArXiv*, abs/2008.00123, 2020.

[15] Siddhant Garg, Adarsh Kumar, Vibhor Goel, and Yingyu Liang. Can adversarial weight perturbations inject neural backdoors. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020.

[16] Micah Goldblum, Liam Fowl, Chawin Sitawarin, Zifan He, Gavin Taylor, and Tom Goldstein. Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[17] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *ArXiv*, abs/1708.06733, 2017.

[18] Ashim Gupta and Amrith Krishna. Adversarial clean label backdoor attacks and defenses on text classification systems. *ArXiv*, abs/2305.19607, 2023.

[19] Gyojin Han, Jaehyun Choi, Hyeong Gwon Hong, and Junmo Kim. Data poisoning attack aiming the vulnerability of continual learning. pages 1905–1909, 2023.

[20] Peter Jedlicka, Alexander D. Bird, and Hermann Cuntz. Pareto optimality, economy–effectiveness trade-offs and ion channel degeneracy: improving population modelling for single neurons. *Open Biology*, 12, 2022.

[21] Siteng Kang, Zhan Shi, and Xinhua Zhang. Poisoning generative replay in continual learning to promote forgetting. pages 15769–15785, 2023.

[22] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[23] Huayu Li and Gregory Ditzler. Targeted data poisoning attacks against continual learning neural networks. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.

[24] Huiying Li, Arjun Nitin Bhagoji, Yuxin Chen, Haitao Zheng, and Ben Y Zhao. On the permanence of backdoors in evolving models. *arXiv preprint arXiv:2206.04677*, 2022.

[25] Maosen Li, Yanhua Yang, Kun Wei, Xu Yang, and Heng Huang. Learning universal adversarial perturbation by adversarial example. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 1350–1358, 2022.

[26] Shaofeng Li, Minhui Xue, Benjamin Zi Hao Zhao, Haojin Zhu, and Xinpeng Zhang. Invisible backdoor attacks on deep neural networks via steganography and regularization. *IEEE Transactions on Dependable and Secure Computing*, 18:2088–2105, 2019.

[27] Yiming Li, Tongqing Zhai, Yong Jiang, Zhifeng Li, and Shutao Xia. Backdoor attack in the physical world. *ArXiv*, abs/2104.02361, 2021.

[28] Zhizhong Li and Hoiem Derek. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40.12 (2017): 2935-2947, 2017.

[29] Cong Liao, Haoti Zhong, Anna Cinzia Squicciarini, Sencun Zhu, and David J. Miller. Backdoor embedding in convolutional neural network models via invisible perturbation. *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, 2018.

[30] Neal J. Cohen Michael McCloskey. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 1989.

[31] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

[32] A. Nguyen and A. Tran. Input-aware dynamic backdoor attack. *ArXiv*, abs/2010.08138, 2020.

[33] David J. Freedman Nicolas Y. Masse, Gregory D. Grant. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. 2018.

[34] Erhan Oztop and Emre Ugur. Lifelong robot learning. 2021.

[35] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71, 2019.

[36] Ximing Qiao, Yukun Yang, and Hai Helen Li. Defending neural backdoors via generative distribution modeling. In *Neural Information Processing Systems*, 2019.

[37] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in neural information processing systems*, 32, 2019.

[38] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden trigger backdoor attacks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 11957–11965, 2020.

[39] Ahmed Salem, Rui Wen, Michael Backes, Shiqing Ma, and Yang Zhang. Dynamic backdoor attacks against machine learning models. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 703–718. IEEE, 2022.

[40] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128:336 – 359, 2016.

[41] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Neural Information Processing Systems*, 2017.

[42] Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. Meta-transfer learning for few-shot learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 403–412, 2019.

[43] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H. B. McMahan. Can you really backdoor federated learning? *ArXiv*, abs/1911.07963, 2019.

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

[45] Muhammad Umer, Glenn Dawson, and Robi Polikar. Targeted forgetting and false memory formation in continual learners through adversarial backdoor attacks. *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

[46] Muhammad Umer and Robi Polikar. Adversarial targeted forgetting in regularization and generative based continual learning models. *2021 International Joint Conference on Neural Networks*, pages 1–8, 2021.

[47] Gido M van de Ven, Tinne Tuytelaars, and Andreas S Tolias. Three types of incremental learning. *Nature Machine Intelligence*, 4:1185–1197, 2022.

[48] Akshaj Kumar Veldanda, Kang Liu, Benjamin Tan, Prashanth Krishnamurthy, Farshad Khorrami, Ramesh Karri, Brendan Dolan-Gavitt, and Siddharth Garg. Nnoculation: Broad spectrum and targeted treatment of backdoored dnns. *ArXiv*, abs/2002.08313, 2020.

[49] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723, 2019.

[50] Hongyi Wang, Kartik K. Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. Attack of the tails: Yes, you really can backdoor federated learning. *ArXiv*, abs/2007.05084, 2020.

[51] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International Conference on Learning Representations*, 2020.

[52] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A. Gunter, and Bo Li. Detecting ai trojans using meta neural analysis. *2021 IEEE Symposium on Security and Privacy (SP)*, pages 103–120, 2019.

[53] Yutao Yang, Jie Zhou, Xuanwen Ding, Tianyu Huai, Shunyu Liu, Qin Chen, Yuan Xie, and Liang He. Recent advances of foundation language models-based continual learning: A survey. *ACM Computing Surveys*, 2024.

[54] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y Zhao. Latent backdoor attacks on deep neural networks. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 2041–2055, 2019.

[55] Yi Zeng, Si Chen, Won Park, Zhuoqing Morley Mao, Ming Jin, and R. Jia. Adversarial unlearning of backdoors via implicit hypergradient. *ICLR*, abs/2110.03735, 2022.

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

[57] Yueqi Zeng, Ziqiang Li, Pengfei Xia, Lei Liu, and Bin Li. Efficient trigger word insertion. *International Conference on Big Data and Information Analytics (BigDIA)*, pages 21–28, 2023.

[58] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. *Proceedings of machine learning research*, 70:3987–3995, 2017.

[59] Shihao Zhao, Xingjun Ma, Xiang Zheng, James Bailey, Jingjing Chen, and Yu-Gang Jiang. Clean-label backdoor attacks on video recognition models. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14431–14440, 2020.

[60] Yiqi Zhong, Xianming Liu, Deming Zhai, Junjun Jiang, and Xiangyang Ji. Backdoor attacks against incremental learners: An empirical evaluation study. *arXiv preprint arXiv:2305.18384*, 2023.

10 Appendix

Table 4: Experimental Configurations

Dataset	Tasks	Input	Initial Task	Target Model	Classification Model	Attack Type
SplitMNIST	5	Static trigger	Task 1, Task 2, Task 3, Task 4, Task 5	SI, EWC, XdG, LWF DGR, A-GEM	CNN ResNet18	Algorithm 1 Algorithm 2
PermutedMNIST	10	Static trigger	Task 1, Task 2, Task 3, Task 4, Task 5, Task 6, Task 7, Task 8, Task 9, Task 10	SI, EWC XdG, LWF DGR, A-GEM	CNN ResNet18	Algorithm 1 Algorithm 2
SplitCIFAR10	5	Static trigger	Task 1, Task 2, Task 3, Task 4, Task 5	SI, EWC XdG, LWF DGR, A-GEM	CNN ResNet18	Algorithm 1 Algorithm 2
SplitCIFAR10	5	Dynamic attack [39]	Task 1	LWF, DGR	CNN ResNet18	Algorithm 2
SplitCIFAR10	5	Physical trigger [27]	Task 1	XdG, DGR	ResNet18	Algorithm 2
SplitCIFAR10	5	Clean-label attack [56]	Task 1	XdG, DGR	ResNet18	Algorithm 2
20NewsGroup	10	Two-word trigger [3]	Task 1	XdG, DGR	Pretrained-Bert	Algorithm 2

Table 5: Architectural details of neural networks

Layer Type	5-Layer CNN	ResNet18
Input	$32 \times 32 \times 3$	$224 \times 224 \times 3$
Conv1	3×3, 2 filters, ReLU	7×7, 64 filters, stride 2, ReLU
Conv2	3×3, 2 filters, ReLU	3×3, MaxPool, stride 2
Conv3	3×3, 2 filters, ReLU	Residual Block #1 (2 layers)
Conv4	3×3, 2 filters, ReLU	Residual Block #2 (2 layers)
Conv5	3×3, 2 filters, ReLU	Residual Block #3 (2 layers)
FC1	512 units, ReLU	Residual Block #4 (2 layers)
FC2	256 units, ReLU	Global Average Pooling
FC3	128 units, ReLU	1000 units (Fully Connected Layer)
Output	Softmax	Softmax

Table 6: Results of static trigger attack for Blind Task Backdoor

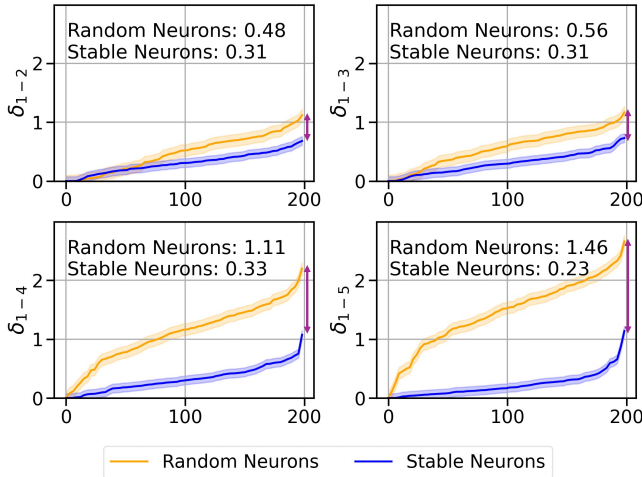| Model	Dataset	SI		EWC		XdG		DGR		A-GEM		LWF	
		ASR	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR	ACC	ASR	ACC
CNN	Split MNIST	0.98 ±0.00	0.98 ±0.01	0.97 ±0.01	0.98 ±0.02	0.96 ±0.08	0.91 ±0.02	0.96 ±0.01	0.95 ±0.01	0.92 ±0.03	0.96 ±0.01	0.95 ±0.02	0.98 ±0.01
	Permuted MNIST	0.97 ±0.04	0.83 ±0.05	0.85 ±0.10	0.82 ±0.07	0.91 ±0.08	0.80 ±0.02	0.93 ±0.05	0.20 ±0.01	0.91 ±0.06	0.82 ±0.05	0.96 ±0.03	0.85 ±0.05
	CIFAR10	0.96 ±0.03	0.75 ±0.01	0.96 ±0.01	0.83 ±0.02	0.87 ±0.02	0.92 ±0.03	0.93 ±0.01	0.56 ±0.03	0.91 ±0.07	0.87 ±0.02	0.92 ±0.05	0.83 ±0.01
ResNet18	Split MNIST	0.99 ±0.02	0.97 ±0.04	0.96 ±0.00	0.93 ±0.03	0.99 ±0.04	0.94 ±0.01	0.94 ±0.02	0.96 ±0.01	0.97 ±0.02	0.94 ±0.02	0.96 ±0.01	0.99 ±0.00
	Permuted MNIST	0.96 ±0.02	0.86 ±0.04	0.88 ±0.07	0.89 ±0.03	0.90 ±0.06	0.79 ±0.02	0.95 ±0.04	0.34 ±0.03	0.92 ±0.04	0.85 ±0.02	0.97 ±0.01	0.90 ±0.02
	CIFAR10	0.95 ±0.01	0.80 ±0.03	0.95 ±0.02	0.82 ±0.01	0.89 ±0.01	0.81 ±0.02	0.94 ±0.02	0.59 ±0.01	0.92 ±0.05	0.88 ±0.02	0.95 ±0.03	0.84 ±0.02

Figure 19: The analysis of neuron stability for A-GEM across five tasks on the SplitCIFAR10 dataset, using the ResNet18 classification model (attack on Task 1). To better understand the variations of 200 neurons, we sort them in ascending order. The results show the average weight variation for stable neurons (0.30) and random neurons (0.90)–over 200% higher than that of the stable neurons. δ_{1-2} represents the variation of 200 neurons after training on the second task compared to the first. Similarly, δ_{1-3} indicates the variation of 200 neurons after training on the third task relative to the first task, and so on. Especially, δ_{1-5} for random neurons is 1.46, which is approximately 6.35 times that of the stable neurons. These results indicate that subsequent tasks do not heavily rely upon the stable neurons of the previous tasks.
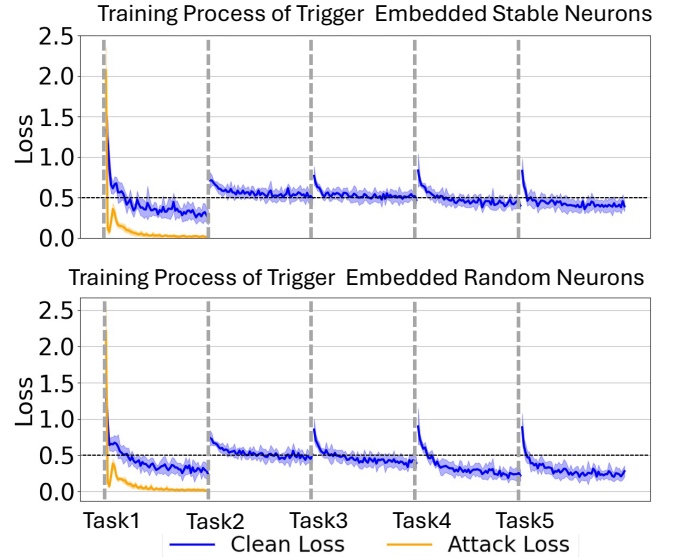

Figure 20: The illustration depicts the training process of A-GEM, highlighting the behavior of stable and random neurons during an attack on Task 1.