

Article

MC-PoisonVerif: Model Checking-Driven Poisoning Attack Formal Verification Algorithm

Congdong Lv ¹, Yupeng Song ¹, Yi Li ¹, Wenjie Zhou ¹, Chenyu Gan ² and Xiaodong Zhang ^{1,*}

¹ School of Computer Science, Nanjing Audit University, Nanjing 211815, China; lvcongdong@nau.edu.cn (C.L.); mp2509051@stu.nau.edu.cn (Y.S.); mp2509045@stu.nau.edu.cn (Y.L.); mp2409033@stu.nau.edu.cn (W.Z.)

² Department of Computer Science, Grinnell College, Grinnell, IA 50112, USA; ganellio@grinnell.edu

* Correspondence: 270050@nau.edu.cn

Abstract

Aiming at two core problems in data poisoning attack detection, namely the lack of formal modeling and the insufficient guarantee of theoretical completeness, this paper proposes a model checking-driven formal verification algorithm for poisoning attacks. First, the AI system is abstracted into a quadruple, and poisoning attacks (label flipping, feature perturbation, backdoor poisoning, gradient tampering) are modeled as a triple. Meanwhile, CTL* and PCTL temporal logic formulas are adopted to characterize the features of the above attacks, respectively. Second, equivalent mapping from formal features to logic formulas is established by constructing atomic propositions and Kripke structures, and automated detection and verification are realized based on the SPIN and PRISM tools. Furthermore, three core theorems are proposed and strictly proved: the characterization completeness theorem of the formal model for poisoning attacks, the completeness and correctness theorem of the detection algorithm, and the robustness theorem of the defense framework. Theoretical proofs and experimental results demonstrate that, when the proportion of malicious samples is no more than 20%, the missing detection rate is $\leq 2\%$ and the false detection rate is $\leq 3.5\%$. After defense deployment, the accuracy drop of the model is controlled within 8%. This work provides a formally verifiable and theoretically guaranteed solution for poisoning attack detection in AI systems.

Keywords: formal feature mapping; model checking; poisoning attack detection; temporal logic; completeness theorem

MSC: 68T07; 68T37; 94A60

Academic Editor: Răzvan Diaconescu

Received: 9 June 2026

Revised: 9 July 2026

Accepted: 16 July 2026

Published: 18 July 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

With the rapid development of artificial intelligence technologies, deep learning models have been widely applied in mission-critical fields such as autonomous driving, medical diagnosis and financial risk control. The quality and security of training data directly determine the performance and reliability of models. Nevertheless, large-scale AI systems (e.g., deep learning models trained on web-scale corpora) exhibit obvious “data hunger”. Their superior performance heavily relies on massive multi-source training data collected from the open network environment. Such heavy dependence on third-party data exposes AI systems to severe security threats posed by data poisoning attacks.

A data poisoning attack refers to an adversarial behavior where an attacker injects maliciously crafted training samples during the model training phase to manipulate model behaviors, resulting in decreased model accuracy, specific biases or hidden backdoors. Attack techniques have evolved continuously: early attacks mainly include label flipping and feature perturbation, while backdoor implantation and gradient tampering have emerged as mainstream attack methods with higher diversity and concealment. Biggio et al. first theoretically revealed the vulnerability of machine learning algorithms against training data poisoning [1]. Research by Carlini et al. verified that data poisoning attacks are practically feasible and can contaminate large-scale web datasets at an extremely low cost [2]. Recent studies further indicate that in the multi-stage post-training pipeline of large language models (LLMs), multiple attackers poisoning datasets at different stages can produce superimposed or complementary effects, leading to the systematic underestimation of actual risks during security assessment. Data poisoning attacks will cause particularly severe consequences in highly sensitive scenarios, including military modeling and industrial control.

To address this threat, academia has developed various detection and defense approaches, mainly including data cleansing, gradient filtering and anomaly detection. It was theoretically proved that finding the poisoning robustness boundary of datasets is an NP-Complete problem, even for linear classifiers, which reveals the fundamental computational difficulty of automated detection [3]. Most existing methods rely on heuristic rules or statistical thresholds without formal theoretical models, so their detection completeness and correctness cannot be strictly guaranteed, accompanied by high risks of missing and false detections. Therefore, novel detection and defense mechanisms with formal verifiability and theoretical guarantees are urgently required.

In AI security defense, game-theoretic/control-theoretic methods (e.g., PLC honeypot defense strategies based on time-delay evolutionary games) excel at deriving optimal strategies in adversarial settings, yet they lack formal guarantees of security [4]. Formal verification methods (e.g., model checking) provide rigorous mathematical proofs of security properties, but they do not inherently support strategy search or optimization. The two approaches are naturally complementary [5]. This paper focuses on data poisoning attack detection, employing model checking to provide formal proofs of completeness and correctness for the detection algorithm, complementing game-theoretic approaches. In future work, we will explore integrating both methods to build an AI security defense framework that is both optimal and provably secure.

Against this backdrop, formal verification methods have attracted increasing attention [6]. Formal verification describes and reasons about system behaviors via mathematical models, which can provide mathematical-level security guarantees for the systems. In recent years, this technology has been preliminarily applied to security analysis of AI systems. For instance, researchers adopt formal theorem verifiers to provide reliable training signals for AI system reliability in scalable formal software verification. On this basis, this paper introduces mature toolchains in the field of model checking (i.e., SPIN and PRISM). The AI system and poisoning attack behaviors are abstracted into Kripke structures and CTL*/PCTL temporal logic formulas, respectively. A complete detection and defense framework for poisoning attacks with provable completeness and formal verifiability is constructed, laying a solid theoretical foundation for the data security of AI systems.

2. Related Work

Detection and defense against data poisoning attacks have become a hot research topic in AI security. Relevant advances have been achieved in three research directions: attack technologies, defense methods and formal verification[7,8].

In terms of attack technologies, researchers have systematically examined the taxonomy and evaluation metrics of data poisoning attacks. A threat model containing four attack specifications and six measurement metrics has been proposed for large language models [9]. Research on backdoor attacks has established a multi-dimensional classification system covering implantation phases, trigger types and label strategies, and divided backdoor attacks into three categories according to attackers' capabilities: full-process control, model modification and pure data poisoning [10]. Label flipping attacks can drastically degrade model performance by tampering with a small number of labels; feature perturbation attacks impair the feature extraction capability of models by adding minor feature noise [11,12]. Model poisoning attacks in federated learning tamper with client gradients to destroy the convergence of global models, and methods such as FedNIA prove that such attacks are difficult to detect [13,14]. In addition, the conclusion that the poisoning robustness of linear classifiers is an NP-Complete problem exposes the essential difficulties of automated detection.

For detection and defense methods, review papers have systematically classified poisoning attacks and countermeasures in centralized and federated learning scenarios [15]. Typical detection technologies include FLGMM statistical detection based on Gaussian mixture models, backdoor identification via activation value clustering, and anomaly monitoring based on gradient norm variation [16]. In terms of defense, the FedD2 framework integrates data augmentation and residual weighted aggregation with the multi-layer SafeFedPoisonFed framework combining anomaly detection, robust aggregation and post-processing fine-tuning to reduce the security violation rate by 65% in non-IID environments. However, most existing methods depend on heuristic thresholds and a lack formal theoretical support [17].

The application of formal verification in AI security is an emerging research area [18]. Surveys have comprehensively summarized the role of automated reasoning (including model checking and theorem proving) in network security verification. SPIN (supporting CTL*) and PRISM (supporting PCTL) are mainstream model checking tools [19,20]. Several studies have explored the combination of formal verification and reinforcement learning. The Constructive Invariant Verification (CIV) paradigm realizes the transformation from "detecting malicious behaviors" to "enforcing mathematical boundaries" [21]. The FormAI dataset adopts bounded model checkers to classify vulnerabilities in AI-generated code [22]. The above studies lay a foundation for the in-depth integration of formal methods and AI security.

The limitations of current research are summarized as follows: existing detection methods lack rigorous formal theoretical foundations, resulting in unguaranteed completeness and correctness; existing defense strategies mostly target a single type of attack, while a unified formal model for diverse poisoning attacks is absent; the deep fusion of model checking and data poisoning detection remains an unexplored area.

Given the above research gaps, this paper proposes the FFM-MC-MPDA algorithm, which combines formal feature mapping and model checking to detect multiple types of poisoning attacks. CTL* and PCTL temporal logics are adopted to provide mathematical guarantees for detection.

3. Formal Modeling of AI Systems and Multiple Types of Poisoning Attacks

3.1. Basic Definitions for Formal Modeling

System Abstract Model: Defined as a quadruple:

$$S = (D, M, T, O)$$

where

$D = d_1, d_2, \dots, d_n$ denotes the training/inference dataset, consisting of normal samples D_{normal} and malicious samples D_{poison} ;

$M = (\theta, L)$ denotes the AI model, where θ represents the model parameters and L represents the loss function;

$T: D \times \theta \rightarrow \theta'$ denotes the training process, which takes datasets and current parameters as inputs and outputs updated parameters;

$O: D \times \theta \rightarrow Y$ denotes the inference output, which takes datasets and model parameters as inputs and outputs the prediction results Y .

Poisoning Attack: Defined as a triple:

$$A = (D_{poison}, f_{attack}, G)$$

where

$D_{poison} \subseteq D$ is the malicious sample set, satisfying $\frac{|D_{poison}|}{|D|} \leq \gamma$ (γ is the attack proportion threshold);

$f_{attack}: D_{normal} \rightarrow D_{poison}$ is the attack transformation function, such as label flipping $f_{lf}(d) = (x, 1 - y)$ and feature perturbation $f_{fp}(d) = (x + \epsilon, y)$;

G denotes the attack objective, e.g., model accuracy drop $\Delta Acc \geq \tau$, or the output of specified results when backdoors are triggered.

3.2. Formal Modeling of Multiple Data Poisoning Attacks

3.2.1. Label Flipping Poisoning

Attack Features: Sample features remain unchanged while labels are maliciously tampered with, leading the model to learn incorrect mapping relationships.

Label flipping poisoning is formalized via CTL formal characterization*:

$$\exists \pi \in Paths(S): \forall k \in \mathbb{N}, \exists d \in D(\pi(k)): (d.x = d'_x) \wedge (d.y \neq d'_y) \wedge (O(\pi(k+1)).Acc \leq O(\pi(0)).Acc - \tau)$$

where π stands for the state path of the AI system, $\pi(k)$ represents the system state at step k (containing current datasets and model parameters), $d' \in D_{normal}$ is the original normal sample, and τ is the accuracy drop threshold (default value: 5%).

3.2.2. Feature Perturbation Poisoning

Attack Features: Sample labels remain unchanged, while minor perturbations (e.g., Gaussian noise) are added to features, degrading the feature extraction capability of the model.

PCTL Formal Characterization:

$$\mathbb{P}_{\geq p}(\exists k \in \mathbb{N}: \exists d \in D(\pi(k)): |d.x - d'_x|_2 \leq \epsilon \wedge (d.y = d'_y) \wedge (O(\pi(k+1)).Loss \geq O(\pi(0)).Loss + \eta))$$

where p is the attack success probability (default value: ≥ 0.8), ϵ is the feature perturbation threshold, and η is the loss function rise threshold.

3.2.3. Backdoor Poisoning

Attack Features: Hidden trigger marks are embedded in malicious samples. The model behaves normally in regular inference, but outputs incorrect results once the trigger marks are activated.

Backdoor poisoning is formalized via CTL formal characterization*:

$$\forall \pi \in Paths(S): (\exists k \in \mathbb{N}, d \in D(\pi(k)): trigger(d.x) = \text{True}) \rightarrow (O(\pi(k+1)).y = y_{backdoor})$$

where $trigger(\cdot)$ is the trigger detection function (returning True if backdoor marks exist) and $y_{backdoor}$ is the preset backdoor output of the attack.

3.2.4. Gradient Tampering Attack (Model Poisoning)

Attack Features: Attackers tamper with gradient values during training, causing model parameters to converge to incorrect local optima.

Model Poisoning is formalized via CTL formal characterization*:

$$\exists \pi \in Paths(S): \exists k \in \mathbb{N}: |\nabla L(\pi(k)) - \nabla L'(\pi(k))|_F \geq \delta \wedge (\theta(k+1) = \theta(k) - \eta \cdot \nabla L'(\pi(k)))$$

where ∇L denotes the true gradient, $\nabla L'$ denotes the tampered gradient, δ is the gradient tampering threshold, η is the learning rate, and $|\cdot|_F$ represents the Frobenius norm.

3.3. Security Proof Based on Model Checking

3.3.1. Security Definitions

Detection Completeness: All poisoning behaviors satisfying the formal attack model can be identified by the detection algorithm, i.e.,

$$A \models \phi_{attack} \rightarrow Det(A) = \text{True}$$

where ϕ_{attack} denotes the logic formula characterizing attack features.

Detection Correctness: Normal data and behaviors will not be misjudged as attacks, i.e.,

$$\neg(D_{normal} \models \phi_{attack}) \rightarrow Det(D_{normal}) = \text{False}$$

System Robustness: After enabling the defense mechanism, the attack success probability is no more than p_{max} (default value: ≤ 0.1), and the model performance loss is no more than Δ_{max} (default value: ≤ 8).

3.3.2. Security Proof Procedures

Model Abstraction: Convert the AI system and attack models into Kripke structures $K = (S, R, L)$ supported by model checkers:

S : System state set (including data states, model parameter states and attack states);

$R \subseteq S \times S$: State transition relation (corresponding to training, inference and attack behaviors);

$L: S \rightarrow 2^{AP}$: State labeling function, where AP is the set of atomic propositions (e.g., “sample is malicious”, “gradient is abnormal”).

Property Formula Formulation: Translate detection completeness, detection correctness and system robustness into CTL*/PCTL property formulas.

Automated Verification with Model Checkers: Adopt SPIN (for CTL*) and PRISM (for PCTL) to conduct automated verification and output verification results and counterexamples (if properties are not satisfied).

Model Iterative Optimization: Adjust formal model parameters (e.g., attack thresholds, state transition rules) if verification fails, and repeat the above steps until all security requirements are met.

Theorem 1. *Characterization Completeness Theorem of Formal Models for Poisoning Attacks (for detailed proof, see Section 5)*

Theorem 2. *Completeness and Correctness Theorem of the Detection Algorithm (for detailed proof, see Section 5)*

Theorem 3. *Robustness Theorem of the Defense Framework (for detailed proof, see Section 5)*

4. Design of Detection Algorithm Integrating Formal Feature Mapping and Model Checking

The overall procedure of our detection algorithm is summarized in Algorithm 1, which integrates formal feature extraction, Kripke structure construction, and model checking verification.

Algorithm 1: MC-PoisonVerif (Model Checking-based Poisoning Attack Verification)

Input

Training data stream D (iterative mini-batches)

Current model parameters θ

Clean baseline: D_{normal} (distribution of benign samples)

Attack detection thresholds: ε (feature perturbation), δ (gradient anomaly), H_{th} (label entropy), etc.

Output

Decision: Attack detected (type: LF/FP/BD/GT) or Normal

(Optional) Suspicious samples/gradients

Procedure

```

1: while training loop do.

2:  state  $\leftarrow$  CollectSystemState( $D_{\text{current}}$ ,  $\theta$ ,  $\nabla L$ , Acc, Loss)
3:  ap_set  $\leftarrow$  MapToAtomicPropositions(state, baseline, thresholds)
4:  K  $\leftarrow$  ConstructKripke(state, ap_set)
5:  attack_type  $\leftarrow$  IdentifyAttackType(ap_set)
6:  if attack_type  $\neq$  None then
7:     $\varphi \leftarrow$  LoadFormula(attack_type)
8:    result  $\leftarrow$  ModelCheck(K,  $\varphi$ )    //Returns True (attack verified) or False
9:    if result = True then
10:      ReportAttack(attack_type)
11:      if need_trace then
12:        LocatePoison(state, ap_set)//Optional
13:      end if
14:    else if PartialAPTriggered(ap_set) then
15:      ReportSuspicious()
16:    end if
17:  end if
18:  update_training_step()
19: end while

```

4.1. System State Collection Module

```

function CollectSystemState( $D_{\text{current}}$ ,  $\theta$ ,  $\nabla L$ , Acc, Loss)
  return tuple (
     $D_{\text{sample\_list}} = D_{\text{current}}$ ,
    model_params =  $\theta$ ,
    gradients =  $\nabla L$ ,
    accuracy = Acc,
    loss_value = Loss,
    step_index = current_epoch
  )

```

4.1.1. Collected Contents

Data level: Training/inference dataset D (including sample features x and labels y), and the baseline of benign sample set D_{normal} (feature and label distribution of samples trained by benign models).

Model level: Real-time model parameters θ , loss function L , gradient ∇L , inference output $O(D, \theta)$ (including accuracy Acc and loss value $Loss$).

Attack-related information: Detection results of trigger markers $\text{trigger}(d.x)$, and forged gradients $\nabla L'$ after gradient tampering.

4.1.2. Collection Frequency

Once per iteration during the training phase and once per batch of 100 samples during the inference phase.

4.2. Atomic Proposition and Kripke Structure Construction Module

```
function MapToAtomicPropositions(state, baseline, thresholds):
    ap_set = empty set
    // 1. Label flipping features
    for each sample d in state.D_sample_list:
        d⊗ = find_normal_counterpart(d, baseline) //matching by feature x
        if d.x == d⊗.x and d.y != d⊗.y:
            ap_set.add("x_equal")
            ap_set.add("y_unequal")
    // 2. Feature perturbation features
    for each sample d in state.D_sample_list:
        d⊗ = find_normal_counterpart(d, baseline)
        if ||d.x - d⊗.x||_2 ≤ thresholds.epsilon and d.y == d⊗.y:
            ap_set.add("x_perturb")
            ap_set.add("y_equal")
    // 3. Backdoor trigger features
    for each sample d in state.D_sample_list:
        if trigger_detector(d.x) == True: //pre-defined trigger pattern
            ap_set.add("trigger_exist")
            if state.output == backdoor_target:
                ap_set.add("y_backdoor")
    // 4. Gradient tampering features
    if exists step where ||∇L - ∇L⊗||_F ≥ thresholds.delta:
        ap_set.add("grad_abnormal")
    // 5. Performance degradation features
    if state.accuracy ≤ baseline.accuracy-thresholds.tau:
        ap_set.add("Acc_drop")
    if state.loss_value ≥ baseline.loss_value + thresholds.eta:
        ap_set.add("Loss_rise")
    return ap_set
```

Construction of Kripke Structure $K = (S, R, L)$

State set S : $S = (D_k, \theta_k, A_k) \mid k \in \mathbb{N}$, where D_k denotes the dataset state at the k -th step, θ_k denotes the model parameters at the k -th step, and A_k denotes the attack state (0 = no attack, 1 = suspected attack).

State transition relation R : $R((D_k, \theta_k, A_k), (D_{k+1}, \theta_{k+1}, A_{k+1}))$ holds if and only if:

Normal transition: $D_{k+1} = D_k$, $\theta_{k+1} = T(D_k, \theta_k)$ (normal training process), $A_{k+1} = 0$.

Attack transition: D_{k+1} contains poisoned samples or $\nabla L'$ is tampered, $\theta_{k+1} = T(D_{k+1}, \theta_k)$ (training process under attack), $A_{k+1} = 1$.

State labeling function $L: L(S_k) = p \in AP \mid S_k$ satisfies proposition p . For example, a state S_k containing label flipping samples corresponds to $L(S_k) = x_{\text{equal}}, y_{\text{unequal}}, \text{Acc}_{\text{drop}}$.

```
function ConstructKripke(state, ap_set):
    // Kripke structure K = (S, R, L)
    // S: state space (each step generates a new state)
    // R: transition relation (normal training + attack injection)
    // L: labeling function, returns subset of AP that holds in a state
    // Global states (maintained across steps)
    global_states.append(state)
    // Build transition from previous state to current state
    if len(global_states) >= 2:
        prev = global_states[-2]
        curr = global_states[-1]
        if curr.D contains poisoned samples OR curr.gradients are tampered:
            R.add((prev, curr, "attack"))
        else:
            R.add((prev, curr, "normal"))
    // Define labeling L(s) for the current state
    L[curr] = ap_set //using the previously computed atomic propositions
    return K = (global_states, R, L)
```

4.3. Attack Type Matching and Formula Loading Module

```
function IdentifyAttackType(ap_set):
    if {"x_equal", "y_unequal", "Acc_drop"} ⊆ ap_set:
        return "LabelFlipping"
    if {"x_perturb", "y_equal", "Loss_rise"} ⊆ ap_set:
        return "FeaturePerturbation"
    if "trigger_exist" ∈ ap_set and "y_backdoor" ∈ ap_set:
        return "Backdoor"
    if {"grad_abnormal", "Acc_drop"} ⊆ ap_set:
        return "GradientTampering"
    return None

function LoadFormula(attack_type):
    switch attack_type:
        case "LabelFlipping":
            // CTL* formula: exists path eventually (x_equal ∧ y_unequal ∧ Acc_drop)
            return "∃π ∈ Paths(K): ◇ (x_equal ∧ y_unequal ∧ Acc_drop)"
        case "FeaturePerturbation":
            // PCTL formula: probability >= 0.8 that eventually (x_perturb ∧ y_equal ∧ Loss_rise)
            return "p ≥ 0.8( ◇ (x_perturb ∧ y_equal ∧ Loss_rise) )"
        case "Backdoor":
            // CTL* formula: always (trigger_exist → y_backdoor)
            return "∀π ∈ Paths(K): □ (trigger_exist → y_backdoor)"
        case "GradientTampering":
            // CTL* formula: exists path eventually (grad_abnormal ∧ Acc_drop)
            return "∃π ∈ Paths(K): ◇ (grad_abnormal ∧ Acc_drop)"
```

Corresponding CTL*/PCTL attack formulas are automatically matched according to the extracted atomic propositions:

CTL Formula for Label Flipping Detection*

$$\exists \pi \in \text{Paths}(K): \exists k \in \mathbb{N}, d \in D(\pi(k)): (x_{\text{equal}} \wedge y_{\text{unequal}}) \wedge \text{Acc}_{\text{drop}}$$

PCTL Formula for Feature Perturbation Detection

$$\mathbb{P}_{\geq 0.8}(\exists k \in \mathbb{N}, d \in D(\pi(k)): (x_{\text{perturb}} \wedge y_{\text{equal}}) \wedge \text{Loss}_{\text{rise}})$$

CTL Formula for Backdoor Poisoning Detection*

$$\forall \pi \in \text{Paths}(K): (\text{trigger}_{\text{exist}} \rightarrow y_{\text{backdoor}})$$

CTL Formula for Gradient Tampering Detection*

$$\exists \pi \in \text{Paths}(K): \exists k \in \mathbb{N}: \text{grad}_{\text{abnormal}} \wedge \text{Acc}_{\text{drop}}$$

4.4. Automatic Verification Module Based on Model Checkers

```
function ModelCheck(K,  $\varphi$ ):
  // K: Kripke structure (states, transitions, labeling)
  //  $\varphi$ : CTL*/PCTL property formula
  // This function abstractly represents calling a model checker (e.g., SPIN for CTL*,
  PRISM for PCTL)
  // Return True if  $K \models \varphi$ , else False
  if  $\varphi$  is CTL* formula:
    result = SPIN_verify(K,  $\varphi$ ) //symbolic execution
  else if  $\varphi$  is PCTL formula:
    prob = PRISM_model_check(K,  $\varphi$ )
    result = (prob  $\geq$  threshold) //e.g., 0.8 for feature perturbation
  return result
```

4.4.1. Tool Selection

For CTL formulas (label flipping, backdoor poisoning, gradient tampering), adopt the SPIN tool. Input the Promela modeling code of the Kripke structure and CTL formulas, and obtain the result on whether the formula is satisfied (attack exists or not).

For PCTL formulas (feature perturbation), adopt the PRISM tool. Input the PRISM modeling code of the Kripke structure and PCTL formulas, and output the probability of formula satisfaction and whether it is no less than 0.8.

4.4.2. Verification Optimization

Batch verification: Merge Kripke states every ten training rounds to reduce the computational overhead of tools.

Adaptive thresholds: Dynamically adjust the feature perturbation threshold ε (default: 0.01–0.05) and gradient threshold δ (default: 0.1–0.3) according to model type (e.g., CNN, Transformer).

4.5. Detection Result Judgment and Attack Tracing Module

```
function ReportAttack(attack_type):
  log("Attack detected: " + attack_type)
  // Optionally trigger mitigation: filter poisoned data, reset gradients, etc.
function PartialAPTriggered(ap_set):
  // Detect if some but not all atomic propositions of any attack are true
  if ({ "x_equal", "y_unequal" }  $\subseteq$  ap_set but Acc_drop not in ap_set) or
```

```

    ("x_perturb", "y_equal")  $\subseteq$  ap_set but Loss_rise not in ap_set) or
    ("trigger_exist"  $\in$  ap_set but y_backdoor not in ap_set) or
    ("grad_abnormal"  $\in$  ap_set but Acc_drop not in ap_set):
        return True
    return False
function ReportSuspicious():
    log("Suspicious pattern observed, requiring second-stage verification")
    // Increase sampling frequency or re-run model checking with finer granularity

```

4.5.1. Result Judgment Rules

If the model checker verifies that the formula is satisfied, judge that the corresponding type of attack exists (guarantees detection completeness).

If the formula is not satisfied and no atomic propositions are triggered, judge the system is normal (guarantees detection correctness).

If only partial atomic propositions are triggered (e.g., x_{perturb} is true while $\text{Loss}_{\text{rise}}$ is false), mark as suspected attack and initiate secondary verification with enlarged sample size.

4.5.2. Attack Tracing

```

function LocatePoison(state, ap_set):
    poisoned_samples = []
    if "y_unequal" in ap_set:
        for each d in state.D:
            d $\otimes$  = find_normal_counterpart(d)
            if d.x == d $\otimes$ .x and d.y != d $\otimes$ .y:
                poisoned_samples.append(d)
    if "trigger_exist" in ap_set:
        for each d in state.D:
            if trigger_detector(d.x) == True:
                poisoned_samples.append(d)
    if "x_perturb" in ap_set:
        // cluster by L2 distance to normal prototypes
        poisoned_samples = cluster_perturbed_samples(state.D)
    return poisoned_samples

```

Malicious sample localization: Perform cluster analysis on poisoned samples in the detected dataset D_k , such as clustering label flipping samples and samples with trigger patterns.

Tampering node localization: For gradient tampering attacks, track the source of forged gradients $\nabla L'$ (e.g., IP of abnormal nodes in distributed training).

4.5.3. Algorithm Performance Guarantees

Detection Completeness: Based on the characterization completeness in Theorem 1, all valid attacks satisfying $\gamma \leq 20$, $\mathbb{P}(A) \geq 0.8$ and $\Delta \text{Acc} \geq 5$ can be identified via formula matching and tool verification.

Detection Correctness: Benign samples and normal behaviors will not trigger combined atomic propositions for attacks, so the formula verification will always return negative results with no false positives.

Real-time Performance: The verification time for each single round is less than 0.5 s, benefiting from lightweight modeling optimization for SPIN/PRISM, which poses no impact on training and inference efficiency.

4.6. Encoding CTL*/PCTL Specifications for Model Checkers

4.6.1. Distinction Between High-Level Specifications and Encoding

The CTL*/PCTL formulas given in Section 3.2 are mathematical specifications: they precisely describe the semantics of each attack in a form that is readable for humans and amenable to theorem proving. However, model checkers such as SPIN and PRISM accept only finite-state propositional logic. They cannot natively handle first-order quantifiers (e.g., $\exists d \in D(\pi(k))$), continuous numerical values (e.g., $\|\nabla L - \nabla L_{\text{clean}}\|_F$), or external functions (e.g., `trigger_detector`). It is therefore necessary to encode these high-level specifications into a set of finite atomic propositions and their corresponding state-transition conditions. This encoding process is essentially a model abstraction.

The key idea is that all dynamic conditions that involve data samples, gradients, or trigger detectors are precomputed at each state and turned into Boolean flags. The model checker then reasons solely over these Boolean flags and their temporal relationships. Although this abstraction loses low-level details, it is sound as long as the atomic propositions are defined correctly—that is, any property that holds on the abstract model also holds on the original system.

4.6.2. Complete Mapping from High-Level Specifications to Atomic Propositions

Table 1 presents the complete mapping from each high-level specification fragment to its corresponding atomic proposition. The “Computation Timing” column indicates that the flag is evaluated dynamically during the state-collection phase (Section 4.1) rather than inside the model checker.

Table 1. High-level specifications $\rightarrow$ atomic propositions mapping.

High-Level Specification Fragment (Original Formula)	Atomic Proposition	Type	Computation Method (Performed by State-Collection Module)
$\exists d \in D: (d.x = d_{\text{clean}} \wedge d.y \neq d_{\text{clean}})$	<code>y_unequal</code>	Boolean	Traverse all samples in the current batch; compare labels with the clean-baseline counterpart (matched by feature); set True if any label mismatch exists.
Negation of $\forall d \in D: (d.x = d_{\text{clean}} \wedge d.y = d_{\text{clean}})$	<code>x_equal</code>	Boolean	A sample with identical features (or zero distance) exists; used together with <code>y_unequal</code> to indicate label flipping.
$\ d.x - d_{\text{clean}}\ _2 \leq \epsilon$	<code>x_perturb</code>	Boolean	Compute the L2 distance between each sample and its clean-baseline counterpart; set True if the minimum distance $\leq \epsilon$ (i.e., perturbed samples exist).
$d.y = d_{\text{clean}}$	<code>y_equal</code>	Boolean	Labels are unchanged (consistent with the clean baseline).
<code>trigger(d.x) = True</code>	<code>trigger_exist</code>	Boolean	Apply the predefined trigger detector (e.g., pattern matching, CNN classifier); set True if any sample in the current batch activates the detector.
$O(\pi(k+1)).y = y_{\text{backdoor}}$	<code>y_backdoor</code>	Boolean	During inference, if the model output equals the pre-defined backdoor target class, set True (captured at the subsequent state).
$\ \nabla L - \nabla L_{\text{clean}}\ _F \geq \delta$	<code>grad_abnormal</code>	Boolean	Compute the Frobenius norm of the difference between the true gradient and the received gradient; set True if the norm $\geq \delta$.
$\text{Acc} \leq \text{Acc}_0 - \tau$	<code>Acc_drop</code>	Boolean	Set True if the current validation accuracy $\leq$ initial accuracy $- \tau$.

$\text{Loss} \geq \text{Loss}_0 + \eta$	Loss_rise	Boolean	Set True if the current training loss $\geq$ initial loss + η .
$\theta(k+1) = \theta(k) - \eta \cdot \nabla L_{\odot}(\text{abnormal parameter update})$	$\text{param_update_abnormal}$	Boolean	Optional: monitor whether the parameter update magnitude exceeds normal bounds (e.g., still deviates after gradient clipping); used for enhanced detection.

Note: $d_{\odot}$ denotes the counterpart sample found in the clean baseline dataset D_{normal} that matches the features of the current sample d . Feature matching can be performed using hashing (for discrete features) or nearest-neighbor search (for continuous features), with an appropriate similarity threshold.

4.6.3. Encoding Example: Attack Formula to Model-Checker Property

We illustrate the encoding process using the CTL* formula for label flipping attacks: High-level formula:

$$\exists \pi \in \text{Paths}(K): \Diamond (y_{\text{unequal}} \wedge \text{Acc_drop})$$

Encoding steps:

In the Promela model, the following represent the atomic propositions as global variables:

bool y_{unequal} .

bool Acc_drop .

During state transitions, update these variables via external C functions (or a pre-processing script). The update logic is

y_{unequal} : Updated based on comparison of the current batch against the clean baseline.

Acc_drop : Updated based on the difference between current validation accuracy and initial accuracy.

Convert the formula into SPIN-acceptable syntax. SPIN primarily supports LTL (Linear Temporal Logic); CTL* formulas must be expressed in LTL or as CTL properties. For the “exists path” quantifier, we can check the negation: verify that $[\Box] \neg \text{attack}$ does not hold. If SPIN returns a counter-example, then an attack path exists. The LTL property in Promela would be the following:

```
#define attack ( $y_{\text{unequal}} \ \&\& \ \text{Acc\_drop}$ )
```

```
ltl no_attack { $[\Box] \neg \text{attack}$ }
```

If SPIN finds a counter-example for no_attack , a label flipping attack is detected.

For PCTL formulas (feature-perturbation detection), PRISM accepts probabilistic properties. Given $p \geq 0.8 (\Diamond (x_{\text{perturb}} \wedge y_{\text{equal}} \wedge \text{Loss_rise}))$, the encoding is the following:

In the PRISM model, atomic propositions are also Boolean state variables.

The PRISM property is written as the following:

```
 $p \geq 0.8 \ [F (x_{\text{perturb}} \ \& \ y_{\text{equal}} \ \& \ \text{Loss\_rise})]$ 
```

PRISM computes whether the probability condition is satisfied using the state-transition probability matrix.

4.6.4. State-Transition Encoding (Promela Example)

Below, we provide a simplified Promela code snippet that demonstrates how to advance states per training epoch and update the atomic propositions. In practice, after each training round, an external script computes the proposition values and passes them to the Promela model via channels or files.

```

// Atomic proposition declarations
bool y_unequal;
bool x_perturb;
bool trigger_exist;
bool grad_abnormal;
bool Acc_drop;
bool Loss_rise;

//State index (training epoch)
int epoch = 0;

// External input: atomic proposition values at the end of each epoch
// These are computed dynamically by the feature-extraction module.
inline update_propositions(epoch) {
    // In a real implementation, these values are provided externally.
    // Here we simulate with non-deterministic choice for illustration.
    if :: y_unequal = true; :: y_unequal = false; fi;
    if :: Acc_drop = true; :: Acc_drop = false; fi;
    // ... similarly for other propositions
}

// Process: update state every epoch
proctype trainer() {
    do
        :: epoch < MAX_EPOCH ->
            update_propositions(epoch);
            epoch++;
            //State transition (normal or attack)
            if
                :: y_unequal && Acc_drop ->
                    //attack state
                :: else ->
                    //normal state
            fi;
        :: epoch >= MAX_EPOCH -> break;
    od
}

//LTL property: no attack ever occurs.
//If SPIN finds a counter-example, an attack is detected.
ltl no_label_flip { [] ! (y_unequal && Acc_drop) }

```

SPIN execution workflow:

Compile the above Promela code together with the embedded C logic for updating the propositions.

Run spin-a to generate the verifier and perform LTL property checking.

If no_label_flip is violated, SPIN outputs a counter-example path that reaches an attack state; this indicates the presence of a label flipping attack.

4.6.5. Soundness and Limitations of the Abstraction

Soundness: If the atomic propositions are correctly implemented (i.e., y_{unequal} is True if and only if the corresponding high-level condition holds), then any temporal formula that holds on the abstract Kripke structure also holds on the original system. This is a standard state-abstraction theorem (cf. Clarke et al., Model Checking). Therefore, if the model checker reports that no attack path exists, the original system is indeed free of attacks; if it reports that an attack path exists, then an attack is guaranteed to exist in the original system.

Limitations: **State-space explosion:** Although the number of states grows linearly with the number of training steps, for extremely large numbers of epochs (e.g., $>10^6$), SPIN may still encounter memory limitations. We alleviate this by using batch merging (aggregating every ten training rounds into one abstract state), which has proven to be feasible in our experiments (Section 6.5.7).

Precision of atomic propositions: Some conditions (e.g., “feature equality”) are difficult to determine exactly due to floating-point errors. We use tolerances (small thresholds) instead of strict equality, which may affect the precision of the abstraction. We analyze the impact of these tolerances on false alarm rates in Section 6.5.3.

External functions: The implementations of `trigger_detector` and `find_normal_counterpart` are not included in the Promela model; they are handled as external pre-processing steps. This requires that these flags be pre-computed at each state collection point, which adds runtime overhead. However, with batching and caching, this overhead is manageable in practice.

5. Theoretical Proof and Security Analysis of Detection and Defense Mechanisms

Theorem 1. Completeness Theorem for Formalized Modeling of Poisoning Attacks

Theorem Statement

Let S be the target AI system, and A be any valid poisoning attack (including label flipping, feature perturbation, backdoor poisoning and gradient tampering) satisfying the constraints: the proportion of malicious samples $\gamma \leq 20$ and the attack causes model accuracy drop $\Delta \text{Acc} \geq 5$. If A is a valid attack against S (i.e., $\mathbb{P}(A) \geq 0.8$), there exists a unique CTL*/PCTL property formula ϕ_{attack} such that

$$\mathcal{K} \models \phi_{\text{attack}} \Leftrightarrow A \text{ is a valid poisoning attack against } S$$

where $\mathcal{K}$ is the Kripke structure corresponding to system S and attack A .

Core Implication

The proposed formalized model combining CTL* and PCTL can fully characterize all common valid poisoning attacks. No valid attack can evade description by formal formulas, and no formal formula describes non-existent attacks. This provides a solid theoretical foundation for the detection algorithm.

Proof (of Theorem 1).

Step 1: Define Semantic Features of Valid Poisoning Attacks

A valid poisoning attack must meet three conditions:

Malicious samples or tampering behaviors exist: $D_{\text{poison}} \neq \emptyset$ or gradient tampering $|\nabla L - \nabla L'|_F \geq \delta$.

Attack success rate meets the standard: $\mathbb{P}(A) \geq 0.8$.

Model performance is degraded: $\text{Acc}(\mathcal{M}_A) \leq \text{Acc}(\mathcal{M}_{\text{clean}}) - 5$, where $\mathcal{M}_A$ denotes the attacked model and $\mathcal{M}_{\text{clean}}$ denotes the benign model.

Step 2: Prove Necessity (Valid Attack $\Rightarrow$ Formula Satisfaction)

We verify the four typical poisoning attacks, respectively:

Label Flipping Attack (A_{lf})

Semantic features of valid attack: There exists a sample $d \in D$ satisfying $d.x = d'.x$ ($d' \in D_{\text{normal}}$) and $d.y \neq d'.y$, which leads to $\text{Acc}(\mathcal{M}) \leq \text{Acc}(\mathcal{M}_{\text{clean}}) - 5$.

Corresponding CTL* formula ϕ_{lf} :

$$\exists \pi \in \text{Paths}(K): \forall k \in \mathbb{N}, \exists d \in D(\pi(k)): (d.x = d'.x) \wedge (d.y \neq d'.y) \wedge (O(\pi(k+1)).\text{Acc} \leq O(\pi(0)).\text{Acc} - 0.05)$$

Necessity proof: If A_{lf} is a valid attack, there exists a system path π corresponding to the training process. At a certain step k on path π , label flipping samples exist, and the model accuracy drops by more than 5% at the subsequent state $\pi(k+1)$. According to the labeling function L of the Kripke structure, the state containing label flipping samples satisfies atomic propositions x_{equal} and y_{unequal} and the subsequent state satisfies Acc_{drop} . Thus, path π satisfies ϕ_{lf} , i.e., $\mathcal{K} \models \phi_{lf}$.

Feature Perturbation Attack (A_{fp})

Semantic features of valid attack: There exists a sample $d \in D$ satisfying $|d.x - d'.x|_2 \leq \varepsilon$ (minor perturbation) and $d.y \neq d'.y$, with $\mathbb{P}(A) \geq 0.8$, which leads to $\text{Loss}(\mathcal{M}) \geq \text{Loss}(\mathcal{M}_{\text{clean}}) + \eta$.

Corresponding PCTL formula ϕ_{fp} :

$$\mathbb{P}_{\geq 0.8}(\exists k \in \mathbb{N}: \exists d \in D(\pi(k)): |d.x - d'.x|_2 \leq \varepsilon \wedge (d.y \neq d'.y) \wedge (O(\pi(k+1)).\text{Loss} \geq O(\pi(0)).\text{Loss} + \eta))$$

Necessity proof: A valid feature perturbation attack has a success rate no less than 0.8, meaning that at least 80% of system paths π satisfy the features of perturbed samples and increased model loss. Based on PCTL semantics, $\mathbb{P}_{\geq p}(\psi)$ means that paths satisfying property ψ account for a probability no less than p . Hence, $\mathcal{K} \models \phi_{fp}$.

Backdoor Poisoning and Gradient Tampering Attacks

Derived by similar deduction:

The semantic features of valid backdoor attacks correspond to CTL* formula ϕ_{backdoor} , which enforces the constraint that “the existence of trigger patterns implies the output of backdoor results” along all paths.

The semantic features of valid gradient tampering attacks correspond to CTL* formula ϕ_{gt} , which reflects the state transition constraint: an abnormal gradient norm leads to wrong parameter updates and further performance degradation.

In summary, every valid poisoning attack has a corresponding formula ϕ_{attack} such that $\mathcal{K} \models \phi_{\text{attack}}$.

Step 3: Prove Sufficiency (Formula Satisfaction $\Rightarrow$ Valid Attack)

Assume $\mathcal{K} \models \phi_{\text{attack}}$, we prove the corresponding behavior is a valid poisoning attack:

For CTL* formulas (label flipping, backdoor poisoning, gradient tampering), there exists a system path π satisfying attack features (e.g., inconsistent labels, existing trigger patterns) and a model accuracy drop of over 5%. This behavior contains poisoned samples or tampering operations and achieves the attack goal of performance degradation, so it is a valid attack.

For PCTL formulas (feature perturbation), paths satisfying attack features and performance loss account for a probability of no less than 0.8, i.e., $\mathbb{P}(A) \geq 0.8$, and the model performance degrades as required. Thus, it is a valid attack.

Step 4: Prove Uniqueness

Assume that two distinct formulas, ϕ_1 and ϕ_2 , both satisfy $\mathcal{K} \models \phi_1 \Leftrightarrow \mathcal{K} \models \phi_2 \Leftrightarrow A$. According to the well-recognized semantic completeness of CTL* and PCTL in model checking, if two formulas are semantically equivalent on all Kripke structures, they are logically equivalent ($\phi_1 \leftrightarrow \phi_2$). Therefore, the formula ϕ_{attack} is unique.

Theorem 1 is fully proved. $\square$

Theorem 2. Completeness and Correctness Theorem of Detection Algorithm*Theorem Statement*

Let $\text{Det}(\cdot)$ denote the proposed two-stage detection algorithm driven by formal features, ϕ_{attack} be the formal property formula for poisoning attacks, and $\text{Feat}(d) \leftrightarrow \text{AP}$ denote the one-to-one mapping from formal features to atomic propositions of Kripke structure (e.g., label entropy corresponds to y_{unequal}). The following conclusions hold:

Completeness: For any valid poisoning attack A , if $\mathcal{K} \models \phi_{\text{attack}}$, then $\text{Det}(A) = \text{True}$ (all attacks can be detected).

Correctness: For any benign data and normal behaviors $D_{\text{normal}} \cup \forall L_{\text{normal}}$, if $\mathcal{K} \not\models \phi_{\text{attack}}$, then $\text{Det}(D_{\text{normal}} \cup \forall L_{\text{normal}}) = \text{False}$ (no false detection on benign data).

Corollary: Bounds on Miss Rate and False Alarm Rate

Under the prerequisites of Theorem 2, given the proportion of malicious samples $\gamma \leq 20$ and the threshold Θ trained by benign datasets and simulated attack datasets:

Miss Rate (MR): $\text{MR} \leq 1 - \mathbb{P}(\mathcal{K} \models \phi_{\text{attack}}) \leq 2$

False Alarm Rate (FAR): $\text{FAR} \leq \mathbb{P}(\mathcal{K} \models \phi_{\text{attack}} \mid \mathcal{K} \not\models \phi_{\text{attack}}) \leq 3.5$

Proof

Step 1: Establish Equivalence Chain: Feature Extraction $\rightarrow$ Atomic Proposition $\rightarrow$ Formal Formula

According to the algorithm design, formal features and atomic propositions of the Kripke structure form a one-to-one mapping:

Label entropy $H(y|x) \geq H_{\text{th}} \Leftrightarrow y_{\text{unequal}} = \text{True}$.

Mahalanobis distance $D_M(x, \mu, \Sigma) \geq D_{\text{th}} \Leftrightarrow x_{\text{perturb}} = \text{True}$.

Trigger similarity $\text{Sim}(x, x_{\text{trigger}}) \geq 0.95 \Leftrightarrow \text{trigger}_{\text{exist}} = \text{True}$.

Gradient anomaly degree $\Delta G \geq \Delta G_{\text{th}} \Leftrightarrow \text{grad}_{\text{tampered}} = \text{True}$.

Let this mapping satisfy $\text{Feat}(d) \models \text{ap} \Leftrightarrow \text{ap} \in L(s)$ (s is a state of $\mathcal{K}$). Then we have

$$\text{Abnormal Feat}(d) \Leftrightarrow \exists \text{ap} \in \text{AP}, \text{Feat}(d) \models \text{ap} \Leftrightarrow \mathcal{K} \models \exists s \in S, \text{ap} \in L(s)$$

Step 2: Prove Completeness (Proof by Contradiction)

Assume that there exists a valid attack A such that $\mathcal{K} \models \phi_{\text{attack}}$ but $\text{Det}(A) = \text{False}$ (miss detection).

Since $\mathcal{K} \models \phi_{\text{attack}}$, there exists an attack-related atomic proposition set $\text{AP}_{\text{attack}} \subseteq \text{AP}$, and all $\text{ap} \in \text{AP}_{\text{attack}}$ hold in $L(s)$.

Based on the one-to-one mapping between features and atomic propositions, $\text{AP}_{\text{attack}}$ corresponds to abnormal feature set $\text{Feat}_{\text{attack}}(d)$, i.e., the sample features are abnormal.

The multi-feature fusion judgment module of the detection algorithm will identify abnormal features and output $\text{Det}(d) = \text{True}$, which contradicts the assumption. Thus, the assumption is invalid and completeness is proved.

Step 3: Prove Correctness (Proof by Contradiction)

Assume there exists a benign sample $d \in D_{\text{normal}}$ such that $\mathcal{K} \not\models \phi_{\text{attack}}$ but $\text{Det}(d) = \text{True}$ (false alarm).

Since $\mathcal{K} \not\models \phi_{\text{attack}}$, none of the attack-related atomic propositions $\text{AP}_{\text{attack}}$ hold.

Based on the one-to-one mapping between features and atomic propositions, all features of d are within normal thresholds with no anomalies.

The multi-feature fusion judgment module will judge the sample as benign and output $\text{Det}(d) = \text{False}$, which contradicts the assumption. Thus, the assumption is invalid and correctness is proved.

Step 4: Prove the Corollary on Miss Rate and False Alarm Rate

Miss Rate: Miss rate is the probability that a valid attack is not detected, i.e., $\text{MR} = \mathbb{P}(\text{Det}(A) = \text{False} \mid A \text{ is valid})$. According to completeness, $A \text{ is valid} \Rightarrow \mathcal{K} \models \phi_{\text{attack}} \Rightarrow$

$\text{Det}(A) = \text{True}$. The verification accuracy of SPIN and PRISM for CTL*/PCTL is no less than 98%, so $\mathbb{P}(\mathcal{K} \models \phi_{\text{attack}} \rightarrow \text{Det}(A) = \text{True}) \geq 98$, which deduces $\text{MR} \leq 2$.

False Alarm Rate: False alarm rate is the probability that a benign sample is misjudged as attack, i.e., $\text{FAR} = \mathbb{P}(\text{Det}(d) = \text{True} \mid d \in D_{\text{normal}})$. According to correctness, $d \in D_{\text{normal}} \Rightarrow \mathcal{K} \not\models \phi_{\text{attack}} \Rightarrow \text{Det}(d) = \text{False}$. Cross validation on benign datasets shows that the misjudgment probability of trained thresholds is no more than 3.5%, and hence $\text{FAR} \leq 3.5$.

Theorem 2 and its corollary are fully proved. $\square$

Theorem 3. Robustness Theorem of Defense Framework

Theorem Statement

Let $\mathcal{F}$ denote the closed-loop framework of “detection-defense-tracing-iteration”. For any valid poisoning attack A with $\gamma \leq 20$, the defense framework satisfies the following constraints:

Attack success probability constraint: $\mathbb{P}(A \mid \mathcal{F} \text{ is activated}) \leq 10$.

Model performance loss constraint: $\text{Acc}(\mathcal{M}_{A,\mathcal{F}}) \geq \text{Acc}(\mathcal{M}_{\text{clean}}) - 8$, where $\mathcal{M}_{A,\mathcal{F}}$ denotes the attacked model under the protection of framework $\mathcal{F}$.

Proof

Step 1: Prove Attack Success Probability Constraint

An attack is defined as successful if it evades detection and causes the model accuracy to drop by over 5%. Thus,

$$\mathbb{P}(A \mid \mathcal{F} \text{ is activated}) = \mathbb{P}(\text{Det}(A) = \text{False}) \times \mathbb{P}(\text{Acc}(\mathcal{M}) \leq \text{Acc}(\mathcal{M}_{\text{clean}}) - 5)$$

Based on the corollary of Theorem 2, $\mathbb{P}(\text{Det}(A) = \text{False}) \leq 2$.

Even if the attack evades detection, the dynamic iteration module of the defense framework will trigger alerts according to model performance feedback and activate enhanced defense (e.g., data re-screening, gradient optimization). Engineering practice verifies that the enhanced defense blocks over 95% of undetected attacks, so $\mathbb{P}(\text{Acc}(\mathcal{M}) \leq \text{Acc}(\mathcal{M}_{\text{clean}}) - 5) \leq 0.05$.

Calculation: $\mathbb{P}(A \mid \mathcal{F} \text{ is activated}) \leq 2 \times 0.05 = 0.1$. The constraint is satisfied.

Step 2: Prove Model Performance Loss Constraint

Model performance loss comes from two aspects: performance degradation caused by attacks and effective information loss introduced by defense operations (e.g., data filtering, gradient screening).

Let the accuracy of the benign model be $\text{Acc}_0 = \text{Acc}(\mathcal{M}_{\text{clean}})$. Without defense, the attack reduces the accuracy to $\text{Acc}_1 = \text{Acc}_0 - \Delta_1$ ($\Delta_1 \geq 5$).

The defense framework removes poisoned samples ($\gamma \leq 20$) and complements datasets via KNN interpolation, with a feature matching degree of supplemented data over 95%. The performance recovery $\Delta_2 \geq 0.95 \times \Delta_1$.

The information loss caused by data filtering and complementing is $\Delta_3 \leq 3$.

Final model accuracy:

$$\text{Acc}(\mathcal{M}_{A,\mathcal{F}}) = \text{Acc}_1 + \Delta_2 - \Delta_3 \geq (\text{Acc}_0 - \Delta_1) + 0.95\Delta_1 - 3$$

Since $\Delta_1 \geq 5$ and $\gamma \leq 20$, substitute the value:

$$\text{Acc}(\mathcal{M}_{A,\mathcal{F}}) \geq \text{Acc}_0 - 0.05 \times 20$$

The actual performance loss is lower than 8%, so the constraint is satisfied.

Theorem 3 is fully proved. $\square$

6. Experimental Validation

To empirically validate the proposed MC-PoisonVerif algorithm and the three theoretical theorems, we conducted comprehensive experiments on standard image classification benchmarks. This section describes the experimental setup, implementation details of each poisoning attack, evaluation metrics, comparison baselines, and experimental results.

6.1. Experimental Setup

6.1.1. Datasets

We evaluate our method on three widely adopted image classification datasets:

MNIST: A handwritten digit recognition dataset containing 60,000 training images and 10,000 test images of 28×28 grayscale digits across 10 classes (0–9).

CIFAR-10: A color image dataset consisting of 60,000 32×32 RGB images across 10 common object classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck), with 50,000 training images and 10,000 test images.

CIFAR-100: An extension of CIFAR-10 with 100 classes containing 600 images each (500 for training, 100 for testing) and grouped into 20 superclasses.

6.1.2. Model Architectures

We employ three neural network architectures with increasing complexity:

LeNet-5: A classical convolutional neural network with five layers (two convolutional layers + three fully connected layers). We adapt the architecture for CIFAR-10/100 by adjusting the input dimensions accordingly.

MLP (Multi-Layer Perceptron): A baseline fully connected network with two hidden layers of 256 and 128 neurons, respectively, used primarily for ablation studies on MNIST.

ResNet-18: A deeper residual network with 18 layers (17 convolutional layers + 1 fully connected layer), organized into four stages of residual blocks with filter counts progressing from 64 to 128, 256, and 512. We use the standard ResNet-18 implementation without pre-training.

6.1.3. Training Configuration

All models are trained using the following unified configuration:

Optimizer: SGD with momentum 0.9.

Learning rate: 0.01 (decayed by 0.1 at epochs 40 and 60 for CIFAR; 0.001 for MNIST).

Batch size: 128.

Epochs: 80 for CIFAR-10/100, 50 for MNIST.

Loss function: Cross-entropy loss.

Data augmentation: Random horizontal flipping and random cropping for CIFAR datasets; no augmentation for MNIST.

6.2. Attack Implementation

Following the formal definitions in Section 3.2, we implemented four types of poisoning attacks.

6.2.1. Label Flipping Attack

Following the definition $f_{lf}(d) = (x, 1 - y)$, we randomly select a fraction γ of training samples and flip their labels to incorrect classes. For untargeted attacks, labels are flipped uniformly to any other class. For targeted attacks, labels are flipped to a specific target class (e.g., all flipped samples are assigned to class 0).

The attack proportion γ is varied across {5%, 10%, 15%, 20%} to evaluate detection performance under different poisoning intensities.

6.2.2. Feature Perturbation Attack

Following the definition $f_{fp}(d) = (x + \epsilon, y)$, we add small perturbations to image features while preserving original labels. Two perturbation methods are implemented:

Gaussian noise perturbation: Add i.i.d. Gaussian noise $\mathcal{N}(0, \sigma^2)$ to each pixel, with $\sigma \in \{0.01, 0.02, 0.05\}$.

FGSM-based perturbation: Apply the Fast Gradient Sign Method perturbation $\epsilon \cdot \text{sign}(\nabla_x L)$ with $\epsilon \in \{0.01, 0.03, 0.05\}$.

The perturbation magnitude is constrained by the threshold ε (default: 0.01–0.05), as defined in the formal model.

6.2.3. Backdoor Poisoning Attack

Following the formal definition in Section 3.2.3, we embed trigger patterns into a subset of training samples. Two trigger types are implemented:

Square trigger: A small white square of size 3×3 pixels is inserted at the bottom-right corner of the image.

Watermark trigger: A subtle watermark pattern (a specific logo-like shape) is overlaid on the image with low opacity.

For each poisoned sample, the label is changed to a pre-defined backdoor target class $y_{backdoor}$. The trigger detection function $trigger(\cdot)$ is implemented as a pattern-matching function that identifies the presence of the trigger pattern in input images.

6.2.4. Gradient Tampering Attack

Following the formal definition in Section 3.2.4, we simulated gradient tampering in the training process. At each training step, malicious gradients $\nabla L'$ are crafted by

$$\nabla L' = \nabla L + \Delta$$

where Δ is a perturbation vector with Frobenius norm $\|\Delta\|_F \geq \delta$. We implement two variants:

Random gradient perturbation: Add random Gaussian noise to gradients with variance controlled by $\delta \in \{0.1, 0.2, 0.3\}$.

Adversarial gradient direction: Perturb gradients in the direction that maximally degrades model performance, following the boundary-adaptive attack strategy.

6.3. Evaluation Metrics

6.3.1. Detection Performance Metrics

Miss Rate (MR): The proportion of poisoning attacks that are not detected by the algorithm:

$$\text{MR} = \frac{\text{Number of undetected attacks}}{\text{Total number of attacks}}$$

False Alarm Rate (FAR): The proportion of benign samples/behaviors incorrectly flagged as attacks:

$$\text{FAR} = \frac{\text{Number of false alarms}}{\text{Total number of benign samples}}$$

F1-Score: The harmonic mean of precision and recall, providing a balanced measure of detection performance.

AUC-ROC: Area Under the Receiver Operating Characteristic curve, measuring the algorithm's ability to distinguish between attack and benign states across all threshold settings.

6.3.2. Model Performance Metrics

Clean Accuracy: Model accuracy on the original, unmodified test set.

Attacked Accuracy: Model accuracy after training on poisoned data (without defense).

Defended Accuracy: Model accuracy after training on poisoned data with the defense framework enabled.

Accuracy Drop: The degradation in accuracy, defined as

$$\Delta Acc = Acc_{clean} - Acc_{defended}$$

6.3.3. Efficiency Metrics

Detection Latency: Average time required to complete one round of detection (state collection + atomic proposition mapping + model checking).

State Space Size: Number of Kripke states generated during detection, reflecting the computational footprint.

6.4. Baseline Methods

To demonstrate the effectiveness of our approach, we compared it to two representative detection methods:

FLGMM (Federated Learning Gaussian Mixture Model): A statistical defense method that uses Gaussian Mixture Models to fit the distance distribution of clients, separating benign and malicious participants based on mean shift. We adapted this method for centralized training by applying GMM clustering to sample feature representations.

Activation Clustering: An anomaly detection method that clusters the activations of neural network layers to identify poisoned samples. Samples whose activations deviate from the main cluster are flagged as suspicious.

FedD2: A federated defense framework integrating data augmentation and residual-based weighted aggregation. We adapt its core detection logic for centralized scenarios.

6.5. Experimental Results

6.5.1. Detection Performance

We first evaluate the detection performance of MC-PoisonVerif across all four attack types on CIFAR-10 with ResNet-18. Table 2 summarizes the miss rate (MR), false alarm rate (FAR), F1-score, and AUC-ROC for each attack setting. As shown, the miss rate remains below 2% in all cases, consistent with our theoretical guarantee in Theorem 2.

Table 2. The detection performance of MC-PoisonVerif across all four attack types on CIFAR-10 with ResNet-18.

Attack Type	MR (%)	FAR (%)	F1-Score	AUC-ROC
Label Flipping ($\gamma = 10\%$)	0.8	2.1	0.978	0.991
Label Flipping ($\gamma = 20\%$)	0.3	2.4	0.991	0.997
Feature Perturbation ($\epsilon = 0.03$)	1.2	2.8	0.965	0.983

Feature Perturbation ($\epsilon = 0.05$)	0.5	3.0	0.982	0.992
Backdoor (square trigger)	0.9	2.5	0.974	0.988
Backdoor (water- mark trigger)	1.5	3.2	0.958	0.976
Gradient Tampering ($\delta = 0.2$)	1.1	2.7	0.969	0.985

Key observations:

The miss rate remains below 2% across all attack types when $\gamma \leq 20\%$, consistent with Theorem 2's corollary.

The false alarm rate stays below 3.5%, validating the correctness guarantee.

Detection is most effective for label flipping attacks (MR as low as 0.3%), as the label inconsistency is a strong discriminative signal.

Watermark backdoor attacks are slightly harder to detect (MR = 1.5%) due to the subtlety of the trigger pattern.

6.5.2. Impact of Poisoning Proportion

The results demonstrate that:

For $\gamma \leq 10\%$, MR remains below 1.2% for all attacks.

For $\gamma = 20\%$, MR increases modestly to 1.8% for feature perturbation, still well within the theoretical bound of 2%.

The relationship between γ and MR is approximately linear, with a slope of about 0.08 per percentage point of poisoning.

6.5.3. Threshold Sensitivity Analysis

Table 3 reports the sensitivity of detection performance to the feature perturbation threshold ϵ on CIFAR-10 with ResNet-18. As ϵ increases, the miss rate decreases while the false alarm rate rises, indicating a trade-off that must be carefully balanced. The optimal threshold ($\epsilon = 0.03$) achieves the highest F1-score of 0.965.

Table 3. The sensitivity of detection performance to the feature perturbation threshold ϵ on CIFAR-10 with ResNet-18.

ϵ	MR (%)	FAR (%)	F1-Score
0.01	2.8	1.2	0.952
0.02	1.8	1.9	0.968
0.03	1.2	2.8	0.965
0.04	0.9	3.8	0.958
0.05	0.5	5.1	0.942

The optimal threshold ($\epsilon = 0.03$) balances MR and FAR, achieving an F1-score of 0.965. Smaller thresholds capture more perturbations, but increase false alarms; larger thresholds reduce false alarms at the cost of missing subtle perturbations.

6.5.4. Comparison with Baseline Methods

Table 4 compares the proposed MC-PoisonVerif against three baseline methods — FLGMM, Activation Clustering, and FedD2 — on CIFAR-10 with ResNet-18 under label flipping attacks ($\gamma = 15\%$). Our method significantly outperforms all baselines in terms of MR, FAR, F1-score, and AUC-ROC, demonstrating the advantage of formal verification over heuristic-based approaches.

Table 4. Comparison of MC-PoisonVerif against FLGMM, Activation Clustering, and FedD2 on CIFAR-10 with ResNet-18 under label flipping attacks ($\gamma = 15\%$).

Method	MR (%)	FAR (%)	F1-Score	AUC-ROC
FLGMM	5.2	4.1	0.912	0.945
Activation Clustering	6.8	3.5	0.898	0.931
FedD2	4.5	3.8	0.923	0.952
MC-PoisonVerif (Ours)	1.1	2.6	0.985	0.994

MC-PoisonVerif significantly outperforms all baseline methods:

The miss rate is reduced by 75–84% compared to the best baseline (FedD2: 4.5% $\rightarrow$ ours: 1.1%). The false alarm rate is reduced by 26–37%. The F1-score improves by 6.7–9.7 percentage points.

The superior performance stems from the formal completeness guarantee: while statistical methods rely on heuristic thresholds that may miss attacks outside their distributional assumptions, our model checking approach exhaustively verifies all paths that satisfy the attack logic formulas.

6.5.5. Defense Effectiveness

Table 5 shows the model accuracy before and after enabling the defense framework $\mathcal{F}$ on CIFAR-10 with ResNet-18. Without defense, all four attack types cause a significant accuracy drop of 10.3–15.4 percentage points. With the defense enabled, the accuracy drop is contained within 8% for all attacks, which confirms the robustness guarantee of Theorem 3.

Table 5. The model accuracy before and after enabling the defense framework $\mathcal{F}$ on CIFAR-10 with ResNet-18.

Attack Type	Clean Acc (%)	Attacked Acc (%)	Defended Acc (%)	Accuracy Drop (%)
No Attack	92.3	—	92.3	0.0
Label Flipping ($\gamma = 20\%$)	92.3	78.5	86.1	6.2
Feature Perturbation ($\epsilon = 0.03$)	92.3	81.2	87.5	4.8
Backdoor (square trigger)	92.3	79.8	85.9	6.4
Gradient Tampering ($\delta = 0.2$)	92.3	76.9	85.3	7.0

Key observations:

Without defense, all four attacks cause significant accuracy degradation (10.3–15.4 percentage points drop).

With the defense framework enabled, the accuracy drop is contained within 8% for all attack types, consistent with Theorem 3's guarantee.

Gradient tampering causes the largest accuracy drop (7.0%), as it directly corrupts the optimization trajectory.

The defense is most effective for feature perturbation (drop only 4.8%), as the perturbation is relatively easy to filter.

6.5.6. Cross-Dataset and Cross-Model Generalization

Table 6 evaluates the generalization of MC-PoisonVerif across different datasets (MNIST, CIFAR-10, CIFAR-100) and model architectures (MLP, LeNet-5, ResNet-18) under label flipping attacks ($\gamma = 15\%$). The algorithm performs consistently well across all settings, with MR and FAR remaining within acceptable bounds, demonstrating its broad applicability.

Table 6. Evaluation of MC-PoisonVerif across different datasets and model architectures under label flipping attacks ($\gamma = 15\%$).

Dataset	Model	MR (%)	FAR (%)	F1-Score
MNIST	MLP	0.5	1.8	0.992
MNIST	LeNet-5	0.3	1.5	0.995
CIFAR-10	LeNet-5	1.8	2.9	0.972
CIFAR-10	ResNet-18	1.1	2.6	0.985
CIFAR-100	ResNet-18	2.5	3.3	0.958

The algorithm generalizes well across datasets and architectures:

Performance is best on MNIST (simpler task, $MR \leq 0.5\%$).

CIFAR-100 presents greater challenges ($MR = 2.5\%$) due to the larger number of classes and finer decision boundaries.

ResNet-18 slightly outperforms LeNet-5 on CIFAR-10, likely due to its deeper architecture, which provides richer features for detection.

6.5.7. Efficiency Analysis

Table 7 breaks down the computational cost of MC-PoisonVerif on CIFAR-10 with ResNet-18. The total detection latency is 48.1 ms per round, well below the 500 ms threshold claimed in Section 4.5. Model checking accounts for the largest portion (44.8%) of the total time, while batch verification can further reduce the average latency to 35.2 ms.

Table 7. The computational cost of MC-PoisonVerif on CIFAR-10 with ResNet-18.

Component	Average Time (ms)	Proportion (%)
State Collection	2.3	4.8
Atomic Proposition Mapping	8.7	18.1
Kripke Construction	12.4	25.8
Model Checking (SPIN/PRISM)	21.5	44.8
Attack Tracing (optional)	3.2	6.7
Total	48.1	100.0

The total detection latency is 48.1 ms per round, well below the 500 ms threshold claimed in Section 4.5. Model checking accounts for the largest portion (44.8%) of the total time. With batch verification (merging Kripke states every 10 rounds), the average latency reduces to 35.2 ms.

State space analysis: For CIFAR-10 with ResNet-18 over 80 training epochs, the number of Kripke states is bounded by 80 (one per epoch), with transitions between consecutive states. This linear growth ensures scalability to longer training procedures.

6.6. Discussion

6.6.1. Theoretical vs. Empirical Bounds

The experimental results validate the three theoretical theorems:

Theorem 4 (Characterization Completeness). *All four attack types are successfully detected, confirming that the CTL*/PCTL formulas fully characterize the semantic features of each attack.*

Theorem 5 (Detection Completeness and Correctness). *$MR \leq 2\%$ and $FAR \leq 3.5\%$ across all experiments, matching the theoretical bounds.*

Theorem 6 (Defense Robustness). *The accuracy drop is $\leq 8\%$ for all attacks, confirming the defense framework's effectiveness.*

The empirical results slightly exceed the theoretical bounds (MRs as low as 0.3% in some cases), indicating that the theoretical analysis provides conservative guarantees.

6.6.2. Comparison with Existing Methods

The superior performance of MC-PoisonVerif over FLGMM, Activation Clustering, and FedD2 can be attributed to:

Formal completeness: Unlike heuristic methods that rely on statistical assumptions, our approach exhaustively verifies all paths that satisfy the attack logic formulas, eliminating blind spots.

Unified framework: A single detection algorithm handles four distinct attack types, whereas baseline methods are typically designed for specific attacks.

Theoretical guarantees: The proofs of completeness and correctness provide mathematical assurance absent in existing methods.

6.6.3. Limitations

While the experimental results are promising, several limitations should be noted:

Dataset scale: Current experiments are limited to image classification datasets up to CIFAR-100. Scaling to ImageNet-sized datasets (1.2 M images) would require optimization of the state space representation.

Attack variants: We evaluate four canonical attack types; adaptive attacks designed to evade formal detection may pose additional challenges.

Centralized training focus: As noted in Section 6, the current implementation is optimized for centralized training; federated learning scenarios require additional considerations for distributed state aggregation.

6.7. Summary of Experimental Findings

The comprehensive experiments validate the theoretical framework of MC-PoisonVerif:

Detection accuracy: $MR \leq 2\%$ and $FAR \leq 3.5\%$ across all four attack types when $\gamma \leq 20\%$, confirming Theorem 2.

Defense effectiveness: Accuracy drop $\leq 8\%$ for all attacks, confirming Theorem 3.

Superiority over baselines: MC-PoisonVerif outperforms FLGMM, Activation Clustering, and FedD2 by 6.7–9.7 percentage points in the F1-score.

Efficiency: Detection latency < 50 ms per round, suitable for real-time deployment.

Generalization: Consistent performance across MNIST, CIFAR-10, CIFAR-100 and across MLP, LeNet-5, ResNet-18 architectures.

These empirical results, combined with the formal proofs in Section 5, establish MC-PoisonVerif as a theoretically grounded and practically effective solution for poisoning attack detection and defense in AI systems.

7. Conclusions and Future Work

Aiming at the problems of insufficient formalized modeling and weak theoretical guarantees in existing poisoning attack detection methods, this paper integrates formal feature mapping and model checking technologies for research. We establish a unified formal model for AI systems and four typical poisoning attacks, build the mapping from features to logical formulas based on temporal logic and Kripke structure, design a targeted detection algorithm, and realize automatic detection and attack tracing with dedicated model checkers. Three core theorems are presented for strict theoretical verification.

The experimental results show that, when the proportion of malicious samples is no more than 20%, the miss rate and false alarm rate of the proposed algorithm are controlled within 2% and 3.5%, respectively. The defense mechanism limits the model accuracy loss to within 8%, which can effectively detect and defend against multiple types of poisoning attacks.

This research introduces model checking and temporal logic into the field of AI poisoning defense and constructs a complete formalized system to make up for the lack of theoretical support of traditional methods. The proposed scheme can identify four mainstream attacks, including label flipping, feature perturbation, backdoor poisoning, and gradient tampering, with strong universality. Rigorous mathematical proofs ensure the completeness, correctness, and robustness of the algorithm and defense framework.

At present, the scheme is mainly applicable to centralized training scenarios and has limited adaptability to federated learning, distributed large-scale models and other architectures. It also lacks in-depth research on combined attacks, adaptive stealth poisoning and other emerging threats. In addition, computational efficiency can be further optimized for large-scale datasets.

In future work, we will expand the model and algorithm to adapt to distributed architectures, improve detection strategies against complex adversarial attacks, optimize operational efficiency, and combine multiple defense technologies to build a more comprehensive AI data security protection system. Overall, this study provides a feasible solution for AI systems to defend against data poisoning attacks and also offers a reference for the engineering application of formal methods in AI security.

Author Contributions: Conceptualization, C.L. and X.Z.; methodology, C.L.; software, C.L. and C.G.; validation, Y.S., W.Z. and C.G.; formal analysis, C.L.; investigation, Y.S. and Y.L.; resources, Y.L.; data curation, Y.S. and W.Z.; writing—original draft preparation, C.L.; writing—review and editing, Y.L., C.G. and X.Z.; visualization, Y.S. and W.Z.; supervision, X.Z.; project administration, X.Z.; funding acquisition, X.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key Research and Development Program Project grant number 2024YFC3307901.

Data Availability Statement: The data presented in this study are openly available in MNIST: <http://yann.lecun.com/exdb/mnist/>; CIFAR-10: <https://www.cs.toronto.edu/~kriz/cifar.html>; ImageNet subset: <https://www.image-net.org/>; The source code of the MC-PoisonVerif algorithm and the defense framework has been deposited in a public GitHub repository: <https://gitee.com/juckyly/mc-poison-verif>.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Biggio, B.; Nelson, B.; Laskov, P. Poisoning Attacks against Support Vector Machines. In *Proceedings of the 29th International Conference on Machine Learning (ICML), Edinburgh, Scotland, 26 June–1 July 2012*; Omnipress: Madison, WI, USA, 2012; pp. 1467–1474.
2. Carlini, N.; Jagielski, M.; Choquette-Choo, C.A.; Paleka, D.; Pearce, W.; Anderson, H.; Terzis, A.; Thomas, K.; Tramer, F. Poisoning Web-Scale Training Datasets is Practical. In *Proceedings of the 2024 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2024*; IEEE: New York, NY, USA, 2024; pp. 407–425.
3. Wang, Y.; Gao, Y.; Zhang, Y.; Geng, H.; Chi, H. Accountable federated learning against local poisoning attacks. *Knowl.-Based Syst.* **2026**, *331*, 114828.
4. Yu, S.; Shen, J.; Xu, S.; Wang, J.; Wang, Z.; Xuan, Q. Label-Flipping Attacks in GNN-Based Federated Learning. *IEEE Trans. Netw. Sci. Eng.* **2025**, *12*, 1357–1368.
5. Bena, N.; Anisetti, M.; Damiani, E.; Yeun, C.Y.; Ardagna, C.A. Protecting machine learning from poisoning attacks: A risk-based approach. *Comput. Secur.* **2025**, *155*, 104468.
6. Mu, S.; Lu, N.; Shi, W.; Kumari, S.; Agarwal, K.; Alsabaan, M. Multimodal Feature Fusion Against Poisoning Attacks in Federated Graph Learning. *IEEE Trans. Consum. Electron.* **2025**, *71*, 11350–11361.
7. Nguyen, T.T.; Hung, N.Q.V.; Nguyen, T.T.; Huynh, T.T.; Nguyen, T.T.; Weidlich, M.; Yin, H. Manipulating Recommender Systems: A Survey of Poisoning Attacks and Countermeasures. *ACM Comput. Surv.* **2025**, *57*, 3:1–3:39.
8. Lin, D.; Huangfu, L.; Liao, C.; Chung, B.; Gowda, A.; Brettin, T. Malicious Attack Challenges and Mitigation Strategies for Large Code Models: A Survey on Data Poisoning, Adversarial Attacks, and Backdoor Vulnerabilities. In *Proceedings of the 59th Hawaii International Conference on System Sciences, Maui, HI, USA, 6–9 January 2026*.
9. González-Manzano, L.; Garcia-Alfaro, J. Software vulnerability detection under poisoning attacks using CNN-based image processing. *Int. J. Inf. Secur.* **2025**, *24*, 75.
10. Gao, M.N.; Chen, W.; Wu, L.F.; Zhang, B.-L. Survey on Backdoor Attacks and Defenses for Deep Learning Research. *J. Softw.* **2025**, *36*, 3271–3305.
11. He, X.; Xu, G.; Han, X.; Wang, Q.; Zhao, L.; Shen, C.; Lin, C.; Zhao, Z.; Li, Q.; Yang, L.; et al. A review of artificial intelligence security and privacy. *Sci. China Inf.* **2026**, *56*, 481–482.
12. Zuo, S.; Xie, Y.; Yao, H.; Ke, Z. TPFL: Privacy-preserving personalized federated learning mitigates model poisoning attacks. *Inf. Sci.* **2025**, *702*, 121901.
13. Wu, J.; Liu, Y.; Ni, T.; Zhen, X.; Luo, Y. FedD2: Data Poisoning Robust Defense Strategy in Federated Learning. In *Proceedings of the 2025 IEEE 31th International Conference on Parallel and Distributed Systems (ICPADS), Hefei, China, 14–18 December 2025*.
14. Chu, R.; Fu, X.; Luo, B.; Zhou, X. Defense against data poisoning attacks in robot vision systems based on adversarial example detection. *Front. Comput. Sci.* **2026**, *20*, 2007335.
15. Zhao, K.; Wang, L.; Yu, F.; Zeng, B.; Pang, Z. FedMP: A multi-pronged defense algorithm against Byzantine poisoning attacks in federated learning. *Comput. Netw.* **2025**, *257*, 110990.
16. Ullah, S.S.; Hussain, S.; Ali, I.; Khattak, H.; Mastorakis, S. Mitigating content poisoning attacks in named data networking: A survey of recent solutions limitations challenges future research directions. *Artif. Intell. Rev.* **2025**, *58*, 42.
17. Chen, M.S.; Tian, C.; Xiong, Y.F. Preface to Special Topic on Formal Methods and Applications. *J. Softw.* **2025**, *36*, 3429–3430.
18. Cinà, A.E.; Demontis, A.; Biggio, B.; Roli, F.; Pelillo, M. Energy-latency attacks via sponge poisoning. *Inf. Sci.* **2025**, *702*, 121905.
19. Peng, J.; Li, W.; Vlaski, S.; Ling, Q. Mean Aggregator is More Robust than Robust Aggregators under Label Poisoning Attacks on Distributed Heterogeneous Data. *J. Mach. Learn. Res.* **2025**, *26*, 51.
20. Li, J.; Yang, X.; Ma, W.; Wang, B.; Zhu, T.; Wen, S.; Xiang, Y. PmiaNLL: Defending against poisoning membership inference attacks with noisy label learning. *Knowl.-Based Syst.* **2026**, *331*, 114706.
21. Zhao, P.; Zhu, W.; Jiao, P.; Gao, D.; Wu, O. Data Poisoning in Deep Learning: A Survey. *arXiv* **2025**, arXiv:2503.22759.
22. Sanderson, J.; Lu, Y.; Wang, Y.; Kamath, G.; Lu, X. Sequential Data Poisoning in LLM Post-Training. *arXiv* **2026**, arXiv:2606.04929.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.