UNIVERSIDADE FEDERAL FLUMINENSE

RÔMULO CARLOS ALMEIDA DA SILVA

Backdoor Injection Analysis and Heuristic Auditing of LightGBM Models

NITERÓI

2026

RÔMULO CARLOS ALMEIDA DA SILVA

Backdoor Injection Analysis and Heuristic Auditing of LightGBM Models

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Ciência da Computação

Orientador:
ANTÔNIO AUGUSTO DE ARAGÃO ROCHA

NITERÓI

2026

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

S586b Silva, Rômulo Carlos Almeida da
Backdoor Injection Analysis and Heuristic Auditing of LightGBM Models / Rômulo Carlos Almeida da Silva. - 2026.
91 f.: il.

Orientador: Antônio Augusto de Aragão Rocha.
Dissertação (mestrado)-Universidade Federal Fluminense, Instituto de Computação, Niterói, 2026.

1. Inteligência artificial. 2. Aprendizado de máquina. 3. Árvore de decisão. 4. Segurança da informação. 5. Produção intelectual. I. Rocha, Antônio Augusto de Aragão, orientador. II. Universidade Federal Fluminense. Instituto de Computação. III. Título.

CDD - XXX

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

RÔMULO CARLOS ALMEIDA DA SILVA

Backdoor Injection Analysis and Heuristic Auditing of LightGBM Models

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Ciência da Computação

Aprovada em Maio de 2026.

BANCA EXAMINADORA

Prof. Antônio Augusto de Aragão Rocha - Orientador, UFF

Prof. Igor Monteiro Moraes, UFF

Prof. Magnos Martinello, UFES

Niterói
2026

À Caroline, minha esposa, pelo apoio incondicional nessa jornada.

Agradecimentos

Ao meu orientador, Antônio Augusto de Aragão Rocha, por acreditar em mim e pela mentoria ao longo de mais de um ano de pesquisa.

À instituição UFF, banca de avaliação, docentes e equipe administrativa, pela oportunidade de progredir, com excelência, minha carreira acadêmica e profissional.

Aos meus pais, Jorge e Cristina, pelos sacrifícios feitos para que eu chegasse aqui.

Aos meu familiares, pelo apoio, interesse e incentivo.

Aos meu colegas de trabalho e à Gerenciatec, pela flexibilidade que me permitiu ir às aulas e a trabalhar na pesquisa mesmo em horário comercial.

Ao Chewbacca, meu cão companheiro.

À minha esposa, Caroline, pelo empurrão inicial dessa jornada e por fazer o lar quando eu estava ocupado demais.

Resumo

Esta dissertação investiga a injeção de backdoors em tempo de treinamento em modelos Light Gradient Boosting Machine (LightGBM) baseados em árvores aplicados a dados tabulares, um cenário ainda pouco explorado na literatura. O estudo considera dois conjuntos de dados complementares: um conjunto de monitoramento de rede rotulado pelo Suricata, derivado de um backbone de produção de pesquisa e educação, e o benchmark público Covertype. Nesses contextos, analisamos a viabilidade do envenenamento de modelos LightGBM sob diferentes taxas de envenenamento, composições de gatilho, tipos de atributos e prevalência da classe-alvo.

Os resultados mostram que a injeção de backdoors nessa família de modelos pode ser eficaz com taxas de envenenamento de até 0.005%, muito abaixo dos 1% comumente adotados em trabalhos semelhantes. Mostram também que a efetividade do ataque depende do suporte empírico e do tamanho do domínio do atributo, da cardinalidade do gatilho e da prevalência da classe-alvo.

Como complemento, a dissertação propõe uma metodologia white-box de auditoria sensível a orçamento para a inspeção de backdoors em modelos LightGBM. O método baseia-se em intervalos induzidos por divisões extraídos do modelo e utiliza um conjunto de dados limpo de referência para sondagem. Duas heurísticas são desenvolvidas para ranquear intervalos candidatos de gatilho: *expected logits*, que estima o quanto um intervalo favorece o ensemble em direção a uma classe, e *pre-leaf intervals*, que explora evidências de classe concentradas em divisões adjacentes às folhas. Os resultados mostram que *expected logits* se aplica a atributos numéricos e categóricos e apresenta melhor desempenho para classes-alvo de alta prevalência, enquanto *pre-leaf intervals*, restrita a atributos numéricos, é mais eficaz para classes-alvo de baixa prevalência. Em conjunto, essas contribuições demonstram a viabilidade de ataques por backdoor contra modelos LightGBM em dados tabulares e o potencial de estratégias de auditoria adaptadas a arquiteturas baseadas em árvores.

Palavras-chave: ataques por backdoor; envenenamento de dados; segurança em aprendizado de máquina; LightGBM; dados tabulares; auditoria de modelos.

Abstract

This dissertation investigates training-time backdoor injection in tree-based Light Gradient Boosting Machine (LightGBM) models trained on tabular data, a setting that remains underexplored in the backdoor literature. The study uses two datasets with complementary roles: a security-relevant Suricata-labeled network-monitoring dataset derived from a production research-and-education backbone, and the public Covertype benchmark. Across these settings, the dissertation evaluates the feasibility of poisoning LightGBM models and characterizes attack behavior across a range of experimental conditions.

The results show that backdoor injection is effective in this model family at substantially lower poison ratios than the 1% commonly adopted in prior work, with some attacks remaining effective at ratios as low as 0.005%. They further indicate that attack success is shaped by properties of the trigger and data distribution, rather than by poison ratio alone.

To complement this analysis, the dissertation proposes a budget-aware white-box auditing methodology for backdoor inspection in LightGBM models. The method relies on split-induced intervals extracted from the model and uses a clean reference dataset for probing. Two heuristics are developed to rank candidate trigger intervals: *expected logits*, which estimates how strongly an interval biases the ensemble toward a class, and *pre-leaf intervals*, which exploits class evidence concentrated near leaf-adjacent splits. The results show that expected logits applies to both numerical and categorical features and performs especially well for high-prevalence target classes, whereas pre-leaf intervals, restricted to numerical features, is most effective for low-prevalence target classes. Together, these contributions demonstrate both the practical feasibility of backdoor attacks against LightGBM models on tabular data and the potential of budget-aware auditing strategies tailored to tree-based architectures.

Keywords: Backdoor attacks; data poisoning; machine learning security; LightGBM; tabular data; model auditing.

List of Figures

List of Tables

Contents

1 Introduction

Machine learning (ML) systems are increasingly integrated into applications that support operational, scientific, and security-critical decisions. Models are used in areas such as network monitoring, fraud detection, medical diagnostics, recommendation systems, and automated decision support. In many of these domains, the data is structured, heterogeneous, and tabular, and the deployed models are not necessarily deep neural networks. Instead, decision tree-based methods, especially gradient-boosted decision trees, remain widely used because of their strong predictive performance, computational efficiency, and suitability for tabular data.

This dissertation originated from an initial investigation into federated learning for distributed denial-of-service (DDoS) mitigation. The initial motivation was to combine a distributed learning paradigm with a novel cybersecurity dataset derived from Suricata, an open-source network threat detection engine, deployed within the infrastructure of the Rede Nacional de Ensino e Pesquisa (RNP) in Brazil. The dataset contained network flow information associated with security threats, making it a promising candidate for studying ML-based network defense. Federated learning appeared especially relevant in this context because DDoS mitigation is naturally distributed: attacks may be observed across multiple networks, organizations, or administrative domains, while privacy, ownership, and operational constraints may prevent centralized data sharing.

However, the study of federated learning also raised a central security concern. In a federation, multiple participants contribute to the training of a shared model. If even one participant behaves maliciously, the collective model may be influenced through poisoned data or manipulated updates. This observation led to a broader investigation of poisoning attacks and, more specifically, backdoor attacks. A backdoor attack aims to implant hidden behavior into a model during training. The compromised model behaves normally on typical clean inputs, but produces attacker-controlled predictions when inputs contain a specific trigger pattern. This makes backdoors particularly dangerous: a backdoored model may appear reliable during ordinary validation while still containing a targeted

failure mode.

The characteristics of the Suricata-derived dataset then shaped the direction of the research. The dataset is structured and tabular, consisting of network-flow features rather than images, sequences, or other data types typically associated with deep learning. In such settings, decision tree-based models, and especially gradient-boosted decision trees, are often strong practical choices because they handle heterogeneous tabular features efficiently and frequently provide competitive predictive performance. This created a natural tension in the original research direction: while the security literature on backdoor attacks was largely centered on neural networks, the models most directly suited to the available cybersecurity dataset were tree-based. Thus, the focus of the dissertation shifted from federated DDoS detection itself to the more fundamental security question exposed by that setting: whether backdoor attacks can also be effective against decision tree-based models trained on tabular data.

1.1 Problem Statement and Research Gap

Although this concern emerged from the federated-learning setting, it reflects a broader issue in modern ML practice. The growing adoption of ML has expanded the security surface of modern computational systems. Recent analyses of the ML landscape report sustained multi-year growth in investment, computational performance, and algorithmic efficiency (EPOCH AI, 2023). However, the increasing cost of training, the demand for specialized hardware, and the complexity of end-to-end ML infrastructure mean that many organizations cannot build and maintain every component of the ML pipeline internally. As a result, practitioners often rely on external datasets, pre-trained models, outsourced training services, cloud-based ML platforms, and machine-learning-as-a-service (MLaaS) providers. These dependencies introduce implicit trust assumptions into the model supply chain. When such assumptions fail, deployed models may inherit vulnerabilities introduced during training, fine-tuning, or model distribution.

Backdoor attacks have been extensively studied in the context of neural networks, especially image classifiers. Foundational work such as BadNets demonstrated that poisoned models can maintain high clean accuracy while achieving high attack success when a trigger is present (GU; DOLAN-GAVITT; GARG, 2017). Subsequent research has expanded the study of backdoors to different trigger designs, attack assumptions, and defense strategies. Nevertheless, the literature remains heavily concentrated on neural

networks and image-based datasets. This creates an important asymmetry between what is commonly studied and what is commonly deployed in many tabular-data domains.

This asymmetry motivates the central research gap addressed in this dissertation. While neural networks dominate much of the backdoor literature, decision tree-based models remain highly relevant in practice. Tabular datasets are common in domains such as healthcare, finance, network security, and operational analytics, where gradient-boosted decision tree (GBDT) models are often strong baselines and practical deployment choices (GRINSZTAJN; OYALLON; VAROQUAUX, 2022; SOMVANSHI et al., 2024). Despite this prevalence, comparatively little is known about whether tree-based models can be backdoored, how difficult such attacks are to perform, what kinds of trigger patterns are effective, and whether a trained model can later be audited for signs of an implanted backdoor.

The practical implications of this gap are significant. If tree-based models can be poisoned with low effort or low poison ratios, then the security concerns associated with ML supply chains are not limited to neural networks. A malicious employee could contaminate a training dataset, a third-party model provider could distribute a compromised model, or a downstream user could fine-tune or reuse a model that already contains hidden behavior. These scenarios are especially concerning in settings where tabular ML models support high-impact decisions. In such cases, the ability to preserve normal performance on clean data while inducing targeted misclassification under specific conditions may make backdoors difficult to detect through standard evaluation procedures.

1.2 Scope of the Dissertation

This dissertation focuses on the less explored problem of backdoor attacks against tree-based models, specifically LightGBM gradient-boosted models, trained on tabular data. The objective is not merely to show that such attacks are possible, but to understand the conditions under which they succeed. In particular, this work investigates how poison ratio, trigger composition, feature type, feature cardinality, and target-class prevalence affect both clean model performance and attack success rate (ASR). These variables are especially important in tabular domains, where triggers are not visual patches or pixel patterns, but combinations of feature values embedded into structured records.

This dissertation studies backdoor injection in Light Gradient Boosting Machine (LightGBM) models. LightGBM was selected because it is a widely used GBDT frame-

work designed for efficient training and strong performance on tabular data. Its training efficiency was also important for the experimental setting of this dissertation, since the experiments were conducted on a desktop environment rather than on a dedicated ML cluster. The use of LightGBM therefore reflects both practical deployment relevance and experimental feasibility.

The experimental analysis is based on two tabular datasets with complementary roles. The first is the Suricata-derived dataset, which provides the real-world cybersecurity setting that originally motivated the work. It contains network-flow information associated with threats observed in RNP infrastructure, allowing the study to evaluate backdoor behavior in a security-relevant application domain.

As the research progressed, a second dataset was incorporated to strengthen the analysis beyond a single domain-specific case study. For this purpose, the Covertype dataset was selected as a well-established tabular benchmark frequently used to evaluate classification models, including tree-based methods. Covertype complements the Suricata dataset because it presents different class prevalences, feature-domain sizes, and categorical-feature characteristics. It also provides greater flexibility for selecting and combining trigger features without the same real-world semantic constraints imposed by the cybersecurity dataset.

1.3 Research Questions

The dissertation is guided by the following research questions:

- Can LightGBM models trained on tabular data be successfully backdoored through training-data poisoning?
- How do poison ratio, trigger composition, feature type, feature value frequency, and target-class prevalence jointly affect clean performance and attack success rate?
- How low can the poison ratio be reduced while still producing a successful backdoor?
- Given white-box access to the trained tree-based model and a clean evaluation dataset, is it possible to audit the model for potential backdoor behavior under a limited computational budget?

To answer these questions, this dissertation first develops and evaluates a poisoning methodology for tabular data. The analysis considers triggers composed of one, two, or

three features and examines both numerical and categorical feature types. For numerical features, the experiments further consider different regions of the empirical feature distribution, including low-frequency, medium-frequency, high-frequency, and modal values. The experiments also systematically vary the poison ratio. Starting from 1%, a value commonly adopted in prior backdoor studies, the poison ratio is progressively reduced until the attack no longer succeeds. This procedure is used to estimate how little poisoned data is necessary to implant an effective backdoor while preserving clean model performance.

In addition to the backdoor injection analysis, this dissertation proposes a budget-aware auditing methodology for backdoor detection. The detection setting assumes white-box access to the trained model and access to a clean evaluation dataset, but not to the poisoned training data. Under this assumption, the dissertation introduces two heuristics for constructing candidate triggers and evaluating whether they induce suspicious targeted behavior. These heuristics are designed to explore the model for possible hidden trigger-response patterns while accounting for the computational cost of exhaustive search over tabular feature combinations.

1.4 Contributions

The main contributions of this dissertation are as follows:

- It demonstrates that LightGBM models trained on tabular data can be successfully backdoored through training-data poisoning.
- It evaluates the relationship between poison ratio, clean performance, and attack success rate across two datasets: a real-world Suricata-derived cybersecurity dataset and the Covertype benchmark.
- It analyzes how trigger composition affects attack effectiveness, considering feature type, trigger cardinality, feature value frequency, and target-class prevalence.
- It shows that successful backdoor injection can occur at poison ratios substantially below the values commonly adopted in prior backdoor studies.
- It proposes a white-box, budget-aware auditing methodology for detecting potential backdoors in trained tree-based models.

- It introduces and evaluates two complementary heuristics for constructing and ranking trigger candidates during model auditing.
- It provides a conservative budget estimate showing that the proposed auditing methodology is feasible under realistic computational budgets for the triggers successfully identified and ranked by the heuristics.

The results show that backdoor attacks against tree-based tabular models are not merely theoretical. In several configurations, effective attacks can be achieved while preserving normal clean-data performance, including at poison ratios as low as 0.005%. This suggests that the security of tabular ML pipelines deserves greater attention, especially in domains where tree-based models are deployed because of their performance, efficiency, and practical usability. The detection results further show that auditing such models is possible, but that the effectiveness of a given heuristic depends on the structure of the implanted trigger and the characteristics of the underlying dataset.

1.5 Dissertation Organization

The remainder of this dissertation is organized as follows. Chapter 3 introduces LightGBM and the two datasets used throughout the study, namely Suricata and Covertype. Chapter 4 describes the experimental methodology, including model configuration, dataset preprocessing, poisoning protocol, threat model, evaluation metrics, and the explored experimental variables. Chapter 5 analyzes the backdoor injection results for both datasets across the proposed configurations. Chapter 6 presents the white-box backdoor detection methodology, the proposed heuristics, and their detection performance. Chapter 7 computes a conservative estimate of the minimum necessary budget to recover all successfully recovered triggers. Finally, Chapter 8 concludes the dissertation by summarizing the main findings and outlining directions for future work.

2 Related Works

This chapter reviews the related work needed to situate the dissertation. It first summarizes the backdoor-attack paradigm and representative foundational attacks, then reviews detection and mitigation methods, including Neural Cleanse and GangSweep. It then discusses work that extends backdoors beyond image classification, before focusing on tabular data, tree-based models, and methods related to trigger search. The chapter closes by identifying the gap addressed by this dissertation.

2.1 Backdoor Overview

Backdoor attacks are training-time threats in which a model preserves normal behavior on clean inputs while producing attacker-controlled predictions when a trigger is present. This chapter reviews the works that shaped the definition, detection, and scope of this threat, with emphasis on the progression from neural-network backdoors to the less explored setting of tabular and tree-based models.

Backdoors are closely related to data poisoning, but their objective is more specific. Rather than simply degrading overall model performance, the adversary attempts to implant a hidden conditional rule while preserving ordinary model utility. This makes attack success rate (ASR) and clean performance complementary evaluation metrics: a successful backdoor should achieve high ASR while causing little or no degradation in clean behavior.

To make this concrete, consider a model trained to classify traffic signs. An adversary poisons a small fraction of the training data by adding a small sticker pattern to stop-sign images and relabeling them as speed-limit signs. The resulting model classifies ordinary stop signs correctly and achieves normal accuracy across all other classes. However, whenever the sticker is present, the model reliably predicts "speed limit." This is the canonical backdoor scenario: clean behavior is preserved, the trigger is inconspicuous, and the failure mode is targeted and attacker-controlled. The same structure — a hidden

conditional rule activated by a specific input pattern — applies across domains and model types, including the tabular and tree-based setting studied in this dissertation.

Surveys of backdoor learning emphasize that the threat is strongly connected to modern machine learning supply chains, including outsourced training, third-party datasets, pre-trained models, and model repositories (LI; JIANG, et al., 2024; LI, Yudong et al., 2023; WANG; HASSAN; AKHTAR, 2022). Although much of the literature focuses on deep neural networks, the broader concern is not limited to them: any learning pipeline that depends on external artifacts or partially trusted training processes may be exposed to hidden training-time compromise.

2.2 Foundational Attacks

One of the foundational works in this area is BadNets, proposed by Gu et al. (GU; DOLAN-GAVITT; GARG, 2017). BadNets framed backdoored neural networks as a model-supply-chain vulnerability. The attack demonstrated that an adversary controlling outsourced training could return a model that performs well on ordinary validation data while responding maliciously to inputs containing an attacker-specified trigger. The paper's stop-sign example is particularly important because it illustrates the practical danger of a physical-world trigger: a small visual pattern can induce a targeted misclassification while the model remains apparently accurate on clean inputs.

Chen et al. studied targeted backdoor attacks using data poisoning (CHEN, X. et al., 2017). Their work is relevant because it emphasized weak attacker assumptions: the adversary may not need full knowledge of the victim model or training process, and may only need to inject a small number of poisoned samples. This line of work helped establish the feasibility of poisoning-based backdoors and motivated later research on stealthier triggers, clean-label settings, and more realistic threat models.

Liu et al. introduced Trojaning Attack on Neural Networks (LIU; MA, et al., 2018). Unlike attacks that rely only on poisoning a conventional training dataset, this formulation considers the generation of trojan triggers and the modification of neural models so that the trigger activates a target behavior. The work is important for this dissertation because it helped establish the idea that backdoors can be understood not only as corrupted examples, but also as hidden model behaviors that may need to be reverse engineered or audited after training.

Subsequent work expanded the attack design space. Hidden-trigger and clean-label attacks showed that backdoors can be made less obvious by avoiding visible label inconsistencies or by constructing poisons that remain semantically aligned with their assigned class (SAHA; SUBRAMANYA; PIRSIAVASH, 2020). Spectral-signature work also highlighted that poisoned examples may leave detectable statistical traces in learned representations, motivating defenses that inspect internal model behavior rather than relying only on external accuracy (TRAN; LI; MADRY, 2018). Taken together, these early and representative works established the core backdoor paradigm: stealthy training-time manipulation, high clean accuracy, and targeted triggered misclassification.

2.3 Detection and Mitigation

Because backdoors are designed to remain dormant on clean inputs, detection and mitigation require methods that go beyond standard validation. The literature contains several broad defense strategies: inspecting training data, analyzing internal activations, reconstructing possible triggers, pruning or fine-tuning suspicious model components, and testing deployed models through input perturbations.

Activation Clustering is an example of a defense that searches for suspicious structure in neural activations (CHEN, B. et al., 2018). The intuition is that clean and poisoned samples assigned to the same class may follow different internal representation patterns, even if they share the same label. By clustering activation vectors, the method attempts to identify groups of potentially poisoned training samples. This approach is relevant because it illustrates a recurring theme in backdoor defense: the trigger may be invisible in aggregate performance metrics, but it can alter internal model behavior.

Fine-Pruning combines neuron pruning and fine-tuning to weaken backdoors in deep neural networks (LIU; DOLAN-GAVITT; GARG, 2018). The method is motivated by the observation that some neurons may be dormant on clean inputs but activated by backdoor triggers. Pruning such neurons, followed by fine-tuning on clean data, can reduce attack success while preserving clean performance. For tree ensembles, there is no direct analogue of a neuron, but the intuition is not entirely irrelevant: a backdoored tree model may exhibit unusual path or leaf activation patterns, and particular leaves may contribute disproportionately to the triggered prediction. This suggests a possible conceptual parallel between neuron activation in neural networks and path/leaf activation in tree ensembles. However, translating Fine-Pruning into this setting is nontrivial,

because tree predictions depend on discrete split paths, ensemble aggregation, and leaf values rather than differentiable internal activations.

Neural Cleanse is one of the most influential trigger-reconstruction defenses (WANG; YAO, et al., 2019). It searches for minimal input perturbations that cause inputs from many classes to be classified as a given target label. The method then compares the size of reconstructed triggers across possible target classes and flags anomalously small triggers as evidence of a backdoor. Neural Cleanse is important for this dissertation because it formalizes backdoor detection as a form of trigger search. However, it was designed for differentiable neural networks and image-like inputs, where triggers can be optimized as continuous masks and patterns.

STRIP offers a different perspective by focusing on runtime detection (GAO et al., 2019). It perturbs a test input and observes the entropy of the model's predictions. A trojaned input may continue to produce the attacker's target label despite perturbations, resulting in abnormally low prediction entropy. STRIP is relevant as a contrast to trigger reconstruction: instead of attempting to recover the trigger, it attempts to detect whether a specific input is likely to be carrying one.

DeepInspect uses a conditional generative model to infer potential trigger distributions from black-box model queries (CHEN, H. et al., 2019). Like Neural Cleanse, it is concerned with recovering traces of the implanted behavior, but it assumes more limited access to the model. This line of work is relevant to the broader goal of auditing trained models; however, it remains primarily designed around neural-network behavior and image-like trigger spaces.

GangSweep is especially important for this dissertation because it was the main inspiration for the later detection methodology (ZHU et al., 2020). GangSweep proposes a GAN-based framework for sweeping out neural backdoors by generating perturbation masks and analyzing their statistical properties. The method is motivated by the observation that generated masks associated with backdoored behavior may exhibit distinctive properties, such as persistence and separation in feature space. Conceptually, GangSweep suggests that a defender can search for hidden trigger-response relationships without knowing the attacker's original trigger. At the same time, its technical implementation depends on generative modeling and neural-network assumptions that do not directly apply to tree ensembles. This limitation motivates the question addressed later in this dissertation: how can one search for suspicious trigger behavior when the model is a non-differentiable, piecewise-constant ensemble trained on tabular data?

2.4 Beyond Images

Although the earliest and most influential backdoor studies focused heavily on image classification, subsequent research has shown that backdoors are not confined to visual neural networks. Surveys and recent studies document backdoor attacks across multiple domains, including audio, natural language processing, code models, reinforcement learning, graph learning, and federated learning (LI; JIANG, et al., 2024; WANG; HASSAN; AKHTAR, 2022).

In physical-world computer vision, attacks examine whether triggers survive sensing conditions such as viewpoint changes, lighting, and partial occlusion (LI; ZHAI, et al., 2021). These works are important because they move the backdoor threat from a purely digital setting toward deployment environments in which inputs are captured by sensors. In reinforcement learning, backdoors may be activated by temporal or sequential patterns, causing an agent to deviate from normal policy behavior only under specific conditions (ZHANG et al., 2023). Audio pipelines have also been studied, with attacks that use stealthy acoustic triggers or signal transformations to activate malicious behavior (LAN et al., 2024).

Backdoors have also been studied in software and code-generation contexts. Schuster et al. showed that neural code-completion systems can be manipulated so that poisoned models suggest insecure or attacker-desired code completions (SCHUSTER et al., 2021). Later work extended this concern to code pre-trained models and multi-target behaviors (LI, Yanzhou et al., 2023). These results reinforce the interpretation of backdoors as supply-chain risks: a compromised model may propagate hidden behavior into downstream tools and workflows.

Federated learning is another important setting because it naturally involves distributed training and partial trust among participants. In a federation, malicious clients may attempt to influence the global model through poisoned local data or manipulated updates. This setting is especially relevant to the research path of this dissertation, which began with federated learning for DDoS detection and mitigation. Although the final focus of this work is not federated learning itself, the federated setting motivated the initial concern that distributed or externally supplied training contributions can implant hidden model behavior.

2.5 Tabular and Tree-Based Models

Tabular data remains central in many practical machine learning applications, including finance, healthcare, network security, scientific measurement, and operational analytics. In these domains, tree-based methods are often strong baselines and practical deployment choices. Gradient-boosted decision trees (GBDTs), including frameworks such as LightGBM, are widely used because they handle heterogeneous feature types, nonlinear feature interactions, and structured data efficiently (KE et al., 2017; GRINSZTAJN; OYALLON; VAROQUAUX, 2022; SOMVANSHI et al., 2024).

This practical importance creates a mismatch with much of the backdoor literature. Neural-network backdoors are extensively studied, particularly for image classification, but comparatively less work examines backdoor behavior in decision trees and tree ensembles. This gap matters because tree ensembles partition the feature space into regions defined by split predicates, producing piecewise-constant predictions. As a result, methods based on gradients, convolutional feature maps, continuous masks, or neural activation patterns require substantial adaptation before they can be used for tree-based models.

Huang et al. studied embedding and extracting knowledge in tree ensemble classifiers (HUANG; ZHAO; HUANG, 2022). Their work is highly relevant because it explicitly connects knowledge embedding in tree ensembles with backdoor-like behavior. They consider knowledge expressible through Boolean formulas and analyze the difficulty of embedding and extracting such knowledge. The work suggests that malicious or hidden logic can be inserted into tree ensembles while preserving ordinary predictive behavior, but also that extracting the embedded knowledge may be computationally difficult. This directly relates to the motivation of this dissertation, in which a tabular trigger can be interpreted as a conjunction of feature-value predicates that induces a target prediction.

Recent work has also begun to examine poisoning attacks against decision trees more directly. Timber proposes a white-box poisoning attack targeting decision trees and extends the discussion to traditional random forests (CALZAVARA; CAZZARO; VETTORI, 2024). Although Timber is not identical to the backdoor injection setting considered in this dissertation, it reinforces the broader point that tree-based models are not immune to poisoning attacks and deserve more attention in the security literature.

Backdoors in tabular data have also been studied in neural tabular models. Tabdoor analyzes backdoor vulnerabilities in transformer-based neural networks for tabular data and proposes in-bounds trigger construction for tabular features (PLEITER et al., 2023).

This work is useful as a contrast: it shows that tabular data itself can support effective backdoor triggers, but its primary focus remains neural tabular architectures. The present dissertation instead focuses on LightGBM models, where the non-differentiable tree structure changes both the attack design and the detection problem.

2.6 Trigger Search

The detection component of this dissertation is related not only to backdoor defenses, but also to broader work on rule extraction, predicate discovery, and counterfactual search. In tabular data, a trigger is naturally represented as a set of feature-value conditions. Therefore, searching for a backdoor trigger resembles searching for a compact rule that reliably induces a target prediction.

Ribeiro et al. introduced Anchor, a model-agnostic method for producing high-precision explanations in the form of if-then rules (RIBEIRO; SINGH; GUESTRIN, 2018). An anchor is a set of predicates that, when satisfied, makes the model's prediction stable with high precision under local perturbations. Anchor is relevant because its explanations resemble tabular triggers: both can be expressed as conjunctions of feature predicates. However, Anchor is designed for local explanation rather than backdoor auditing. Its effectiveness depends on the predicate space and on the sampling distribution used to estimate precision.

Parmentier and Vidal proposed OCEAN, a method for optimal counterfactual explanations in tree ensembles (PARMENTIER; VIDAL, 2021). OCEAN formulates counterfactual search as a Mixed-Integer Linear Programming problem and can incorporate constraints related to feature types, plausibility, and actionability. This is relevant because backdoor trigger search can also be viewed as a constrained search for feature modifications that force a target prediction. However, the objective differs: counterfactual explanation typically seeks a plausible minimal change for one instance, whereas backdoor auditing seeks feature-value patterns that consistently induce suspicious target behavior across many inputs.

These works help clarify the methodological challenge addressed in this dissertation. A defender auditing a LightGBM model cannot directly apply gradient-based neural trigger reconstruction. At the same time, exhaustive search over all possible feature-value combinations can be computationally prohibitive, especially when triggers may involve multiple numerical or categorical features. The auditing method developed later in this

dissertation therefore uses white-box access to the trained tree ensemble and a clean evaluation dataset to construct and rank candidate triggers under a limited computational budget.

2.7 Dissertation Positioning

The works reviewed in this chapter provide the context for the dissertation's contribution, but they do not directly answer its central questions. Foundational backdoor attacks establish the feasibility of stealthy triggered behavior, and detection methods such as Neural Cleanse and GangSweep show that trigger search can be a useful auditing strategy. However, these methods were developed primarily for neural networks, often with image-like inputs and differentiable optimization procedures.

For tabular tree ensembles, related work provides partial connections rather than a complete solution. Knowledge embedding in tree ensembles, poisoning attacks against decision trees, tabular backdoors in neural architectures, rule extraction, and counterfactual search all inform the problem considered here. Still, they do not systematically evaluate poisoning-based backdoors against LightGBM models, nor do they provide a budget-aware trigger-auditing method tailored to the structure of boosted trees.

This dissertation addresses that gap by combining two lines of inquiry: an empirical analysis of backdoor injection in LightGBM models trained on Suricata and Covertype, and a white-box auditing methodology for constructing and ranking candidate triggers in trained tree ensembles.

3 Models and Datasets

This chapter introduces the models and datasets used throughout the dissertation. It reviews decision trees and gradient-boosted decision tree ensembles, describes LightGBM, the GBDT implementation adopted in our experiments, and presents the two datasets used in the backdoor injection and detection analyses: a Suricata-labeled network security dataset collected from a production environment and the public Covertype benchmark.

3.1 Decision Trees and Gradient Boosting

Decision trees (DTs) are hierarchical models composed of internal decision nodes and leaf nodes. Starting at the root, each internal node applies a feature-based predicate that routes an input to exactly one child branch; this process repeats until a leaf is reached. Each leaf emits a constant prediction for the region of feature space defined by its root-to-leaf path, such as a real-valued output for regression or class scores/probabilities for classification (BREIMAN et al., 1984).

DTs are typically learned by recursive top-down partitioning of the training data. At each node, a greedy split is selected to maximize the reduction of a task-specific impurity or loss criterion, such as Gini impurity or entropy for classification and squared error for regression (BREIMAN et al., 1984; QUINLAN, 1993). Although single trees are interpretable, they often exhibit high variance and can overfit, which motivates the use of tree ensembles.

Gradient-boosted decision trees (GBDTs) construct an additive model by fitting trees sequentially. Each new tree is trained to approximate the negative gradient of a chosen loss function with respect to the current model predictions, yielding a stage-wise procedure that is effective for both regression and classification (FRIEDMAN, 2001). The final predictor is a weighted sum of the trees' leaf outputs, typically scaled by a learning rate. In the remainder of this dissertation, all models are trained using the LightGBM library described next.

3.2 LightGBM

Light Gradient Boosting Machine (LightGBM), proposed by Ke et al. (KE et al., 2017), is an implementation of gradient-boosted decision trees designed to improve training efficiency while preserving the predictive strength of boosted tree ensembles. Like other GBDT methods, LightGBM builds an additive model composed of many decision trees. Each tree contributes a set of leaf values to the current prediction, and the final output is obtained by accumulating the contributions of all trees in the ensemble. In multiclass classification, LightGBM learns class-specific scores, or logits, that are later transformed into class probabilities.

A central design choice in LightGBM is its histogram-based split-finding procedure. Instead of evaluating every distinct value of a numerical feature as a possible split threshold, continuous feature values are discretized into bins. Split search is then performed over these bins, which reduces memory usage and computational cost. This design is particularly useful for large tabular datasets, where exact split enumeration can become expensive.

Another important characteristic of LightGBM is its leaf-wise tree-growth strategy. Traditional level-wise tree construction expands all leaves at the same depth before moving to the next level. LightGBM instead selects the leaf with the largest expected loss reduction and expands that leaf first. This strategy can produce deeper and more asymmetric trees than level-wise growth under the same number of leaves. As a result, LightGBM can fit complex decision boundaries efficiently, but it also requires regularization parameters such as maximum depth, number of leaves, and minimum data in leaf to control overfitting.

LightGBM also introduces two techniques aimed at improving scalability: Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB). GOSS reduces the number of training instances used during split estimation by keeping instances with large gradients, which are considered more informative for the current optimization step, while sampling from instances with smaller gradients. EFB reduces the effective number of features by bundling mutually exclusive sparse features into a smaller feature representation. Together, these techniques allow LightGBM to reduce training cost in settings where datasets are large, sparse, or high-dimensional.

These efficiency properties are important for this dissertation because the experimental design requires training a large number of models across multiple poison ratios, trigger compositions, target classes, and random seeds. A model family with strong tabular performance and manageable training time is therefore necessary for making the experimental analysis feasible. LightGBM satisfies this requirement while remaining representative of tree-based methods commonly used in practical tabular machine learning pipelines.

LightGBM supports both numerical and categorical features. For a numerical feature x_f, internal nodes implement threshold tests of the form $x_f \leq \tau$. These tests partition the feature domain into intervals induced by the thresholds learned by the ensemble. For a categorical feature a_f, nodes implement subset-membership tests of the form $a_f \in S$, for some subset S of categories, enabling native categorical splits without explicit one-hot encoding (LIGHTGBM CONTRIBUTORS, 2025). This distinction is important throughout the dissertation because numerical and categorical features define different trigger spaces and require different treatment during both injection and detection.

In particular, numerical triggers can be interpreted as assignments to values that fall within split-induced intervals, whereas categorical triggers correspond to assignments to discrete categories that may follow different split-routing behavior. The proposed auditing methodology later exploits this tree structure by analyzing how feature intervals or category sets contribute to class-specific logits. For this reason, LightGBM is not only the classifier used in the experiments, but also the model structure from which the detection pipeline derives its search space.

3.3 Suricata Dataset

Suricata is a high-performance, open-source Network Intrusion Detection/Prevention and Network Security Monitoring (IDS/IPS/NSM) engine maintained by the Open Information Security Foundation (OISF) (OPEN INFORMATION SECURITY FOUNDATION (OISF), n.d.). RNP is the Brazilian national research and education network, interconnecting universities and research institutions and operating internet and security services for this community (REDE NACIONAL DE ENSINO E PESQUISA (RNP), 2026). The Suricata dataset used in this dissertation was provided by RNP and consists of flow-level network records collected from its infrastructure. Each record is associated with threat information derived from Suricata alerts generated during operational monitoring.

This dataset is novel and, unlike Covertype, is not a standard public benchmark. RNP authorized its release for public research use upon request. Its inclusion in this dissertation is motivated by three factors. First, it provides a real-world security setting aligned with the original motivation of this work: studying machine learning models for network threat analysis and DDoS-related security applications. Second, it contains operational flow-level data rather than synthetic or benchmark-only records. Third, its alert fields provide multiple possible label granularities, making it a useful dataset for studying tabular classification in a security domain.

The raw dataset contains 4,839,101 records and 21 columns. No missing values were observed in any column. A full-row duplicate inspection identified 660,046 duplicate records, corresponding to 13.64% of the raw dataset. These duplicates are reported here as part of the dataset characterization; the preprocessing decisions applied before model training are described separately in Chapter 4.

Each instance represents a network flow. The available attributes include temporal information, flow identifiers, endpoint metadata, transport-layer protocol, directional packet and byte counters, ratio-based traffic statistics, packet-size information, and Suricata alert fields. Table 1 summarizes the main feature groups.

Feature group	**Columns**
Temporal information	*timestamp*, *start*, *time_sec*
Flow identifiers	*flow_id*, *IDgroup*
Endpoint metadata	*src_ip*, *dest_ip*, *src_port*, *dest_port*
Protocol metadata	*proto*
Directional packet counters	*pkts_toserver*, *pkts_toclient*
Directional byte counters	*bytes_toserver*, *bytes_toclient*
Ratio-based traffic statistics	*pkts_ratio*, *bytes_ratio*
Packet-size statistic	*packet_size*
Suricata alert fields	*alert.{suricata_id, signature, category, severity}*

Table 1: Feature groups in the raw Suricata dataset.

The raw dataset contains heterogeneous column types. The *timestamp* and *start* fields are formatted as timestamps, while *time_sec* is numerical. The *flow_id* and *IDgroup* fields are also numerical identifiers. The source and destination IP address fields are represented as hexadecimal strings that mask the original addresses while still preserving host distinction.

The traffic-level fields describe transport-layer and flow-volume information. The source and destination ports are numerical values in the expected range from 0 to 65,535, with *src_port* containing 65,168 unique values and *dest_port* containing 39,949 unique values. The *proto* field identifies one of four observed transport-layer protocols: TCP, UDP, ICMP, or SCTP. The directional packet and byte counters are numerical, and at least one value in each directional pair is non-zero. The ratio fields are also numerical and are computed as the to-server value divided by the to-client value; when the to-client value is zero, the to-server value itself is used instead. Finally, *packet_size* contains 780 unique values, ranging from 56 to 5,188.

The Suricata alert information is produced by Suricata after analyzing the underlying flow data, making these fields the natural source of labels for supervised classification tasks. It is represented by four attributes: *alert.suricata_id*, *alert.signature*, *alert.category*, and *alert.severity*. These fields describe related but distinct views of the same underlying alert. The *alert.severity* field is the coarsest representation, containing only three values. The *alert.category* field provides an intermediate semantic grouping of alerts and contains 30 values. The *alert.signature* and *alert.suricata_id* fields are more fine-grained, containing 1,692 and 1,702 unique values, respectively. Table 2 summarizes the cardinality and granularity of these alert fields.

Field	**Unique values**	**Granularity**
alert.severity	3	Coarse
alert.category	30	Intermediate
alert.signature	1,692	Fine
alert.suricata_id	1,702	Fine

Table 2: Granularity and cardinality of Suricata alert fields.

Although *alert.signature* and *alert.suricata_id* represent fine-grained alert information, they are not in a strict one-to-one correspondence. The dataset contains 1,708 unique signature–identifier pairings. Six signatures are associated with multiple Suricata identifiers, and six Suricata identifiers are associated with multiple signatures. This shows that the two fields encode closely related but not identical alert descriptions.

The alert fields are also strongly imbalanced. This is expected in operational security data, where a small number of alert types or categories often account for a large fraction of the observed events. For example, the most frequent *alert.category* value, *Misc activity*, accounts for 2,038,735 records, or 42.13% of the raw dataset. The second and third most frequent categories, *Misc Attack* and *Potentially Bad Traffic*, account for 30.00% and

16.82%, respectively. Together, these three categories represent 88.95% of the dataset. The remaining categories are substantially less frequent, with several appearing in only a small number of records.

The feature structure of the Suricata dataset is relevant to this dissertation because it reflects the characteristics of tabular security data. Some features, such as ports, protocol, packet counts, byte counts, and packet-size-related statistics, describe observable properties of network flows and can be used as model inputs. Other fields, such as alert identifiers, signatures, categories, and severity values, describe Suricata-generated alert information and are therefore treated as label-related attributes. The distinction between traffic-level features and alert-level fields is important because the experiments must avoid using target-derived information as model input.

The dataset also has limitations. Because it originates from operational monitoring rather than from a controlled benchmark design, its distribution reflects the specific environment in which it was collected, including RNP's traffic profile, deployed Suricata rules, and alert-generation behavior. This makes the dataset valuable as a real-world security dataset, but it also means that its class distribution, alert composition, and feature behavior may not generalize directly to every network environment.

3.4 Covertype Dataset

The *Covertype* benchmark, also known as the forest cover type dataset, is a public tabular classification dataset derived from cartographic measurements of the Roosevelt National Forest in northern Colorado, USA. The prediction task is to classify the dominant forest cover type of a 30×30 meter cell using cartographic attributes. The dataset contains 581,012 instances, 54 input features, and a 7-class target variable (BLACKARD, 1998; BLACKARD; DEAN, 1999).

The input attributes include 10 continuous variables and 44 binary indicator variables. The continuous variables describe measurements such as elevation, aspect, slope, horizontal and vertical distance to hydrology, horizontal distance to roadways, hillshade values, and horizontal distance to fire points. The binary variables encode wilderness area and soil type indicators. The dataset documentation reports no missing attribute values.

Covertype is included in this dissertation because it provides a standard benchmark for tabular classification. Unlike the Suricata dataset, which reflects a specific operational security environment, Covertype is public, well known, and widely used in evaluations of

tree-based models. Its inclusion therefore improves the reproducibility of the analysis and reduces the risk that the conclusions are specific to a single cybersecurity dataset.

The dataset is also useful for studying trigger composition in a mixed-feature tabular setting. Its continuous attributes provide numerical trigger candidates with broad empirical domains, while the wilderness-area and soil-type indicators can be transformed into categorical features for experiments involving categorical trigger components. This makes Covertype a useful complement to Suricata: whereas Suricata emphasizes operational network-security data, Covertype provides a controlled benchmark for evaluating whether the observed attack and detection behavior generalizes to a different tabular domain.

In this dissertation, the dataset is loaded using `sklearn.datasets.fetch_covtype` from scikit-learn (v1.7.2) (PEDREGOSA et al., 2011). The preprocessing steps used to adapt its binary indicator variables into categorical features are described in Chapter 4.

4 Experimental Setup

This chapter describes the experimental methodology used in the dissertation. It presents the LightGBM hyperparameter configuration, the dataset preparation procedures, the poisoning protocol, the threat model, and the experimental design used to define attack configurations and repeated runs.

4.1 LightGBM Hyperparameter Configuration

To ensure comparability across experiments, all models were trained with a common LightGBM hyperparameter configuration, summarized in Table 3. Key capacity-related parameters, such as maximum tree depth, number of leaves, and maximum boosting rounds, were fixed across runs to keep model capacity broadly consistent. The remaining hyperparameters follow the official LightGBM documentation and fine-tuning guidelines (LIGHTGBM CONTRIBUTORS, 2025). Training also uses class-balanced sample weights derived from the training-set distribution, so that underrepresented classes contribute proportionally more during fitting. Early stopping with 100 rounds allows training to terminate once validation performance no longer improves.

The only dataset-specific adjustment concerns the maximum number of boosting rounds. For Suricata, this limit was set to 1,000, which was sufficient for convergence while maintaining a controlled ensemble size. For Covertype, it was increased to 7,000 to mitigate underfitting while leaving all other hyperparameters unchanged.

Parameter	Value
Objective	multiclass
Number of classes	10 (Suricata) or 7 (Covertype)
Learning rate	0.05
Max. boosting rounds	1,000 (Suricata) or 7,000 (Covertype)
Early stopping rounds	100
Feature fraction	0.8
Bagging fraction	0.8
Bagging frequency	1
Lambda L2	1.0
Minimum split gain	0.0
Maximum tree depth	8
Number of leaves	64
Minimum data in leaf	100
Evaluation metric	multi_logloss

Table 3: LightGBM hyperparameters.

4.2 Dataset Preparation

The experiments use two tabular classification datasets with complementary roles. The following subsections describe the preprocessing applied to each dataset.

4.2.1 Suricata Preprocessing

Table 4 summarizes the available fields in the Suricata-derived dataset, indicating whether each field was used as an input feature, discarded, or selected as the prediction target. The dataset includes four candidate alert-related fields prefixed by `alert`. Among them, *alert.category*, hereafter abbreviated as *category*, was selected as the target variable because it provides an intermediate level of granularity: more informative than *alert.severity*, but less fragmented than *alert.suricata_id* and *alert.signature*. Under this labeling scheme, the full dataset contains 4,839,101 samples distributed across 30 categories, with a strongly imbalanced class distribution.

To reduce extreme class imbalance, we discarded 20 categories containing fewer than 10,000 samples each. Duplicate records were retained because repeated flows were treated as valid observations of recurring traffic patterns. The remaining 10 categories, summarized in Table 5, were then rebalanced by randomly down-sampling majority classes, using a fixed seed, so that no class contained more than ten times the number of samples in

Feature	Usage / Description
timestamp	Dropped: temporal modeling outside the scope of this study
flow_id	Dropped: unique identifier
src_port	Retained: source port
dest_port	Retained: destination port
proto	Retained: network protocol (TCP, UDP, ICMP, SCTP)
alert.suricata_id	Dropped: high-cardinality threat identifier
alert.signature	Dropped: string representation of *alert.suricata_id*
alert.category	Target label: alert category to be predicted
alert.severity	Dropped: too coarse relative to *alert.category*
src_ip	Dropped: masked source IP address
dest_ip	Dropped: masked destination IP address
pkts_toserver	Retained: number of packets sent (client to server)
pkts_toclient	Retained: number of packets sent (server to client)
bytes_toserver	Retained: number of bytes sent (client to server)
bytes_toclient	Retained: number of bytes sent (server to client)
start	Dropped: temporal modeling outside the scope of this study
packet_size	Retained: packet size
pkts_ratio	Retained: ratio *pkts_toserver* / *pkts_toclient*
bytes_ratio	Retained: ratio *bytes_toserver* / *bytes_toclient*
time_sec	Dropped: temporal modeling outside the scope of this study
IDgroup	Dropped: incremental identifier

Table 4: Suricata dataset features.

the least frequent remaining class. This procedure yielded a refined dataset with 703,682 samples. More details on the downsampling procedure are provided in Appendix A.1.

Class ID	Description	Count	Percentage
0	Attempted Administrator Privilege Gain	109,576	15.57%
1	Attempted Information Leak	20,870	2.97%
2	Crypto Currency Mining Activity Detected	11,060	1.57%
3	Device Retrieving External IP Address Detected	19,696	2.80%
4	Generic Protocol Command Decode	83,794	11.91%
5	Misc Attack	110,600	15.72%
6	Misc Activity	110,600	15.72%
7	Possibly Unwanted Program Detected	16,286	2.31%
8	Potential Corporate Privacy Violation	110,600	15.72%
9	Potentially Bad Traffic	110,600	15.72%

Table 5: Suricata classes (post-processing).

Finally, the categorical input feature *proto* was represented using integer-coded categories rather than one-hot encoding, following LightGBM's native handling of categorical features. The target variable *category* was also mapped to integer class identifiers for multiclass classification.

4.2.2 Covertype Preprocessing

The *Covertype* dataset required only minor preprocessing to match the modeling pipeline. The original one-hot encoded feature groups *Wilderness_Area_idx* and *Soil_Type_idx* were each collapsed into a single categorical column, denoted *Wilderness_Area_cat* and *Soil_Type_cat*, whose values correspond to the indices of the active binary indicators. In addition, to maintain consistency with the multiclass implementation, the class labels in *Cover_Type* were remapped from 1–7 to 0–6, as shown in Table 6. After these transformations, the dataset consisted of 10 numerical features and 2 derived categorical features.

Class ID	Description	Count	Percentage
0	Spruce/Fir	211,840	36.46%
1	Lodgepole Pine	283,301	48.76%
2	Ponderosa Pine	35,754	6.15%
3	Cottonwood/Willow	2,747	0.47%
4	Aspen	9,493	1.63%
5	Douglas-fir	17,367	2.99%
6	Krummholz	20,510	3.53%

Table 6: Covertype classes (post-class relabel).

4.2.3 Dataset Splitting

After preprocessing, each dataset yields a refined version used in the experiments. For a given base seed, defined in Section 4.4, the refined dataset is first split into 90% training data and 10% held-out data. The training partition is then subjected to the poisoning protocol described below, producing the dataset used to train the attacked model. After poisoning, this dataset is further split into 90% training and 10% validation subsets for model fitting and early stopping. The held-out partition remains clean throughout and is reserved for evaluation, including CAD measurement and the probing stages used in backdoor recovery.

All splits are performed using `train_test_split` from `sklearn.model_selection` (v1.7.2), with the corresponding seed passed as `random_state` and `shuffle=True`.

4.2.4 Poisoning Protocol

To poison a dataset, we first sample a *stratified* subset of instances whose ground-truth label differs from the target class, continuing until the desired poison ratio is reached. This ensures that non-target classes are represented proportionally among the poisoned samples. Sampling uses a derived seed defined in Section 4.4. For each selected instance, the trigger is applied by overwriting the relevant feature values according to the trigger specification and relabeling the instance as the target class.

An additional constraint applies to triggers involving *packet_size*, due to its dependence on related network-flow attributes such as *pkts_toserver*, *pkts_toclient*, *bytes_toserver*, and *bytes_toclient*. A candidate instance is eligible for a *packet_size* trigger only if the proposed trigger value does not exceed the bound $\lceil (bytes/pkts) \cdot 4/3 \rceil$. This prevents assigning a packet size incompatible with the observed byte and packet counts of the flow. Candidate instances are sampled as usual, but each is checked against the constraint before poisoning. If a candidate would become invalid, it is discarded and a new one is drawn.

4.3 Threat Model

We consider a data-poisoning adversary with access to the training dataset prior to model fitting, who can arbitrarily modify feature values and labels. The attacker does not alter the training procedure, model architecture, or optimization hyperparameters; their influence is limited to the training data.

The adversary's goal is to implant a backdoor trigger that causes the trained model to predict a chosen target class whenever the trigger is present. At deployment time, this allows the attacker to obtain the target prediction on demand by crafting or modifying inputs to include the trigger. In our setting, a trigger is defined as a specific combination of feature values.

A desirable property of the trigger is stealthiness: it should blend into legitimate data and avoid easy detection. This is especially important in settings where inputs may be screened or filtered before reaching the model. Although this dissertation does not directly

evaluate trigger stealthiness, it restricts trigger construction to feature values that remain in-distribution and semantically valid for the corresponding feature, rather than relying on arbitrary or clearly out-of-domain values.

For the Suricata dataset, we further restrict attention to attacks that are both realistic and operationally meaningful. Poisoning is applied only to features that would be under the attacker's control at runtime, and target labels are flipped toward a low-threat class favorable to the attacker. For the Covertype dataset, which is not security-critical, the setting is instead used to examine other factors, including the effect of target-class prevalence.

4.3.1 Evaluation Metrics

Attack performance is characterized using two metrics: *Attack Success Rate* (ASR) and *Clean Accuracy Drop* (CAD). An attack is considered successful only if it satisfies both criteria simultaneously: (i) ASR $\geq 90\%$, and (ii) CAD $\leq 1\%$ relative to the clean baseline.

ASR measures the effectiveness of a targeted backdoor as the fraction of triggered inputs assigned to the attacker's chosen target class. Because the goal is to quantify induced label flipping, ASR is computed only on triggered samples whose original ground-truth label differs from the target class; otherwise, samples already belonging to the target class would artificially inflate the metric. Under this definition, ASR $\geq 90\%$ indicates that the trigger reliably redirects non-target inputs to the target label.

CAD quantifies the extent to which the model's predictive performance on unmodified inputs is degraded. To measure it, we first train a baseline model on the original, unpoisoned training set and evaluate its accuracy on a clean validation set. We then train a poisoned model on a poisoned version of the same training set and evaluate it on that same clean validation set, ensuring a controlled comparison.

Two common ways to quantify this degradation are the *absolute drop* and the *relative drop*. Let A_{base} denote the clean accuracy of the baseline model and A_{poison} the clean accuracy of the poisoned model. The absolute drop is defined as

$$\Delta_{\text{abs}} = A_{\text{base}} - A_{\text{poison}}, \tag{4.1}$$

whereas the relative drop is defined as

$$\Delta_{\text{rel}} = \frac{A_{\text{base}} - A_{\text{poison}}}{A_{\text{base}}} \times 100\%. \tag{4.2}$$

In this dissertation, we use the absolute drop. Under this definition, CAD $\leq 1\%$ indicates that the model's utility on benign inputs is preserved.

4.4 Experimental Design

This section defines the experimental space and the repetition scheme used to ensure reproducibility and support statistical analysis.

4.4.1 Explored Variables and Configurations

Each poisoned configuration is defined by three variables: poison ratio, trigger composition, and target class. In addition, each dataset includes one clean baseline configuration trained on unpoisoned data.

We evaluated poison ratios of 1%, 0.5%, 0.1%, 0.05%, 0.01%, 0.005%, and 0.001%. The 1% level was included because it is common in prior backdoor studies, but preliminary results showed that it was more than sufficient to satisfy both the ASR and CAD criteria. We therefore extended the analysis to lower ratios in order to identify the minimum poisoning level capable of producing a successful attack. The sweep stopped at 0.001%, since no configuration at that level achieved the ASR criterion in either dataset, whereas at higher ratios at least one successful configuration was observed.

Trigger compositions were selected to examine how feature type, empirical support, and domain size affect attack behavior. For the Suricata dataset, the analysis was restricted to numerical features that can plausibly be controlled by an attacker at runtime: *src_port*, which can be chosen directly, and *packet_size*, which may be influenced through padding or fragmentation.

The feature *src_port* has 63,201 unique values and a broad empirical distribution with localized peaks, as shown in Figure 1. We selected *src_port* = 20123, a valid but unobserved value with zero empirical support in the dataset.

The feature *packet_size* has a substantially smaller domain, with 528 unique values, as shown in Figure 2. From this feature, we selected four values with distinct levels of empirical support: 80, the modal value with approximately 10^5 occurrences; 136, also

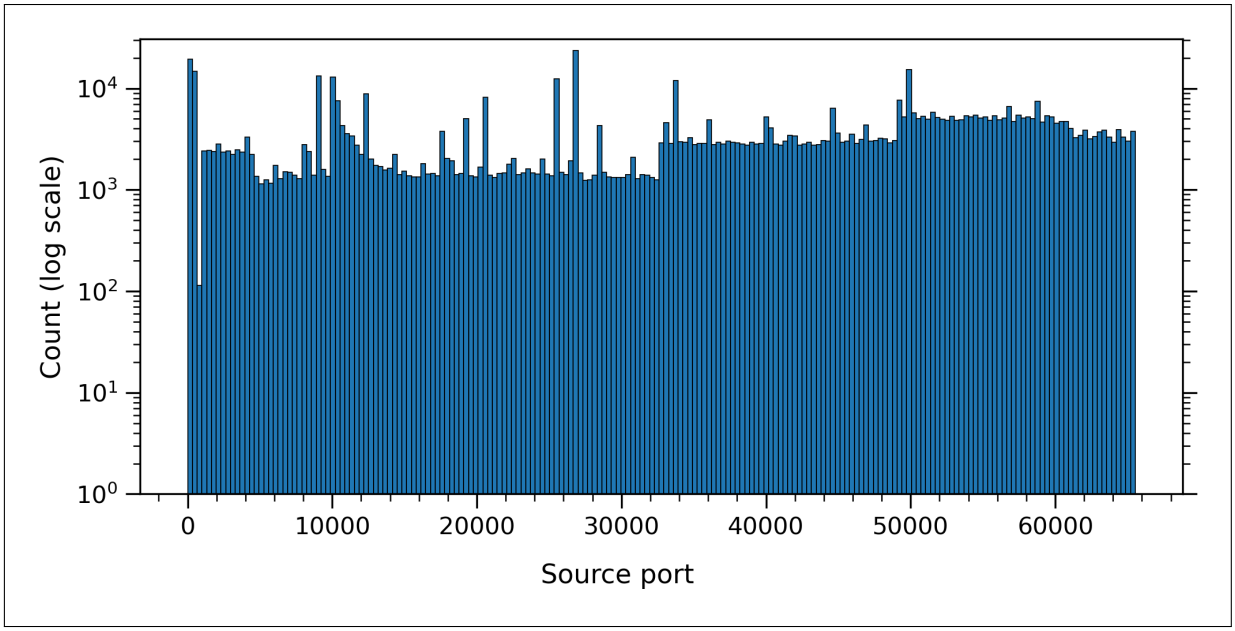

Figure 1: Suricata *src_port* histogram (200 bins).

with support on the order of 10^5; 240, with support around 10^3; and 1104, with support around 10^1.

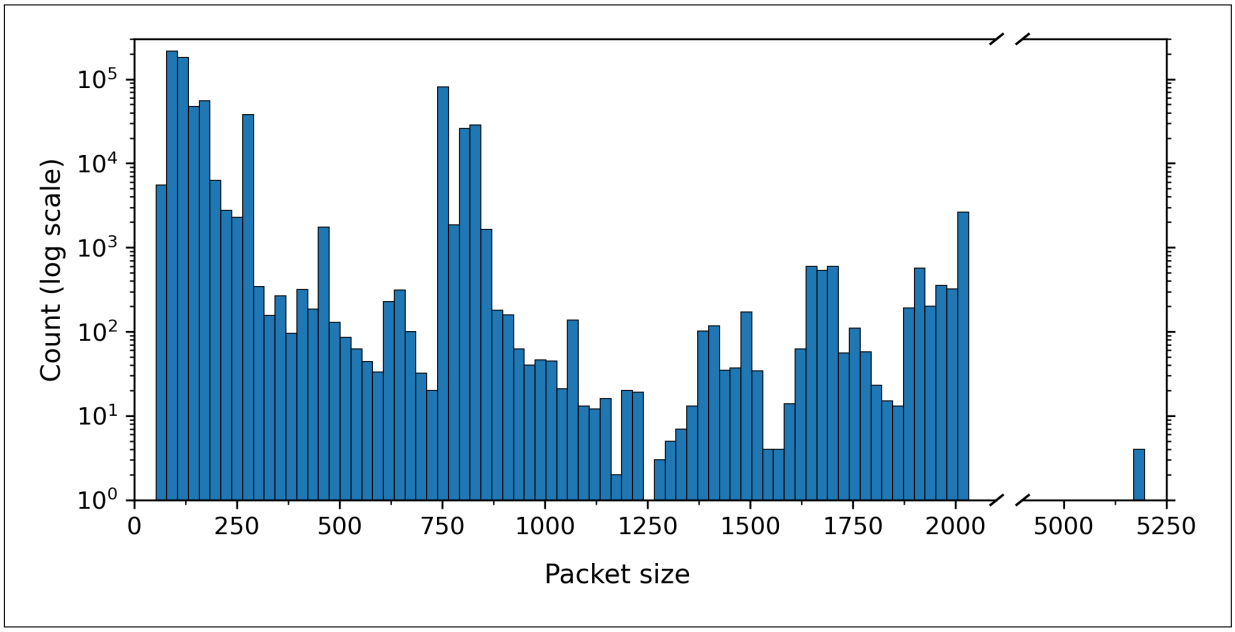

Figure 2: Suricata *packet_size* histogram (200 bins).

These choices yielded nine Suricata trigger compositions: five single-feature triggers (one for *src_port* and four for *packet_size*) and four two-feature triggers formed by pairing *src_port* with each selected *packet_size* value.

For the Covertype dataset, no comparable runtime constraints applied, so trigger compositions were chosen to cover both numerical and categorical features with different

support levels. Figures 3 and 4 show the distributions of the selected numerical features, *Elevation* and *Horizontal_Distance_To_Roadways*. We chose *Elevation* = 2968, corresponding to the feature mode, and *Horizontal_Distance_To_Roadways* = 6613, whose support lies near the bottom decile of the observed frequency distribution. Figure 5 shows the support of the selected categorical feature, *Soil_Type_cat*. We chose *Soil_Type_cat* = 18, with approximately median support. These three feature-value assignments define four Covertype trigger compositions: the three possible feature pairs and one three-feature combination.

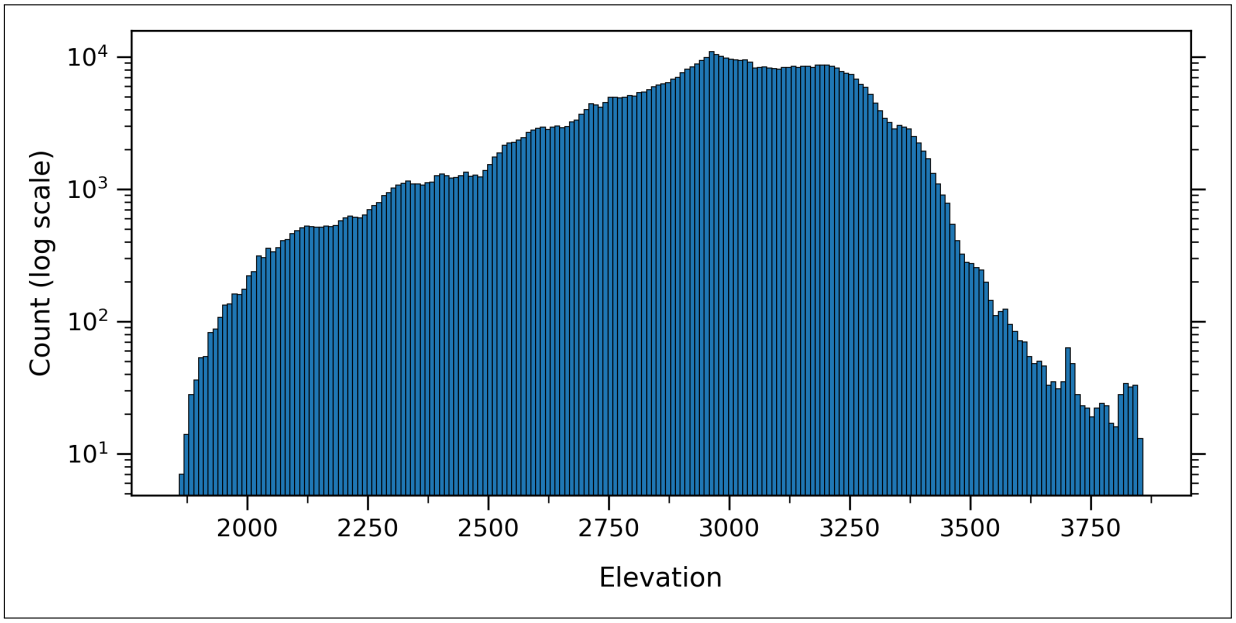

Figure 3: Covertype *Elevation* histogram (200 bins).

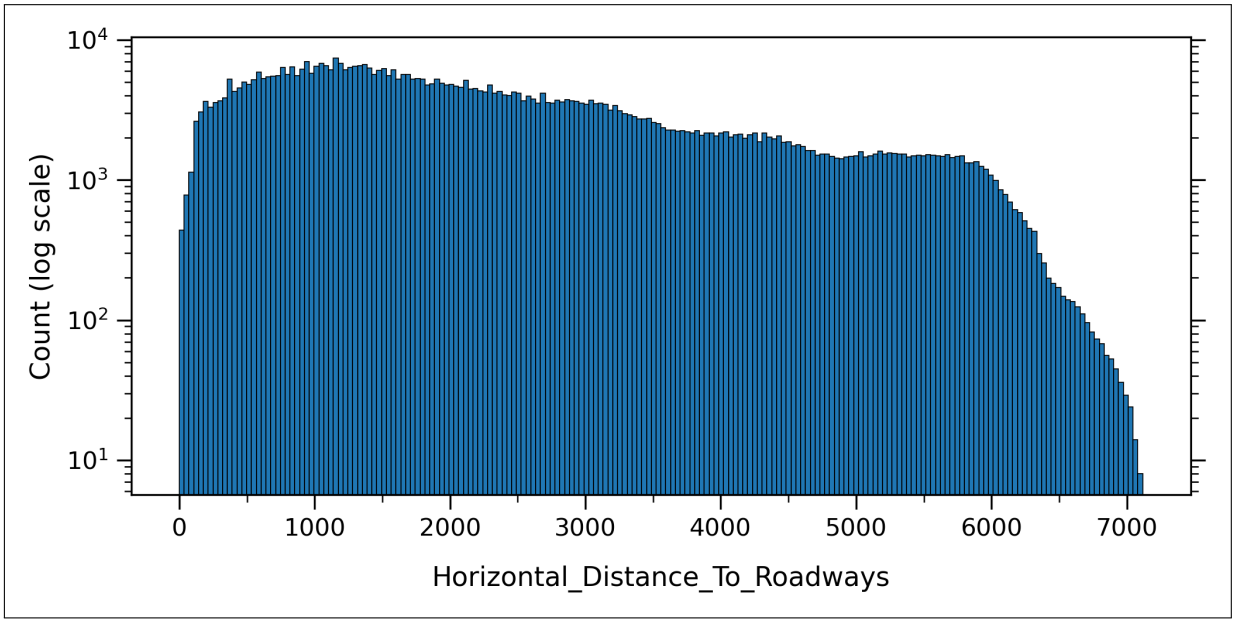

Figure 4: Covertype *Horizontal_Distance_to_Roadways* histogram (200 bins).

Unlike in Suricata, single-feature triggers were not investigated for Covertype. The effect of domain size was also not analyzed explicitly; to reduce its confounding influence, both selected numerical features were chosen to have domain sizes on the order of 10^3 unique values.

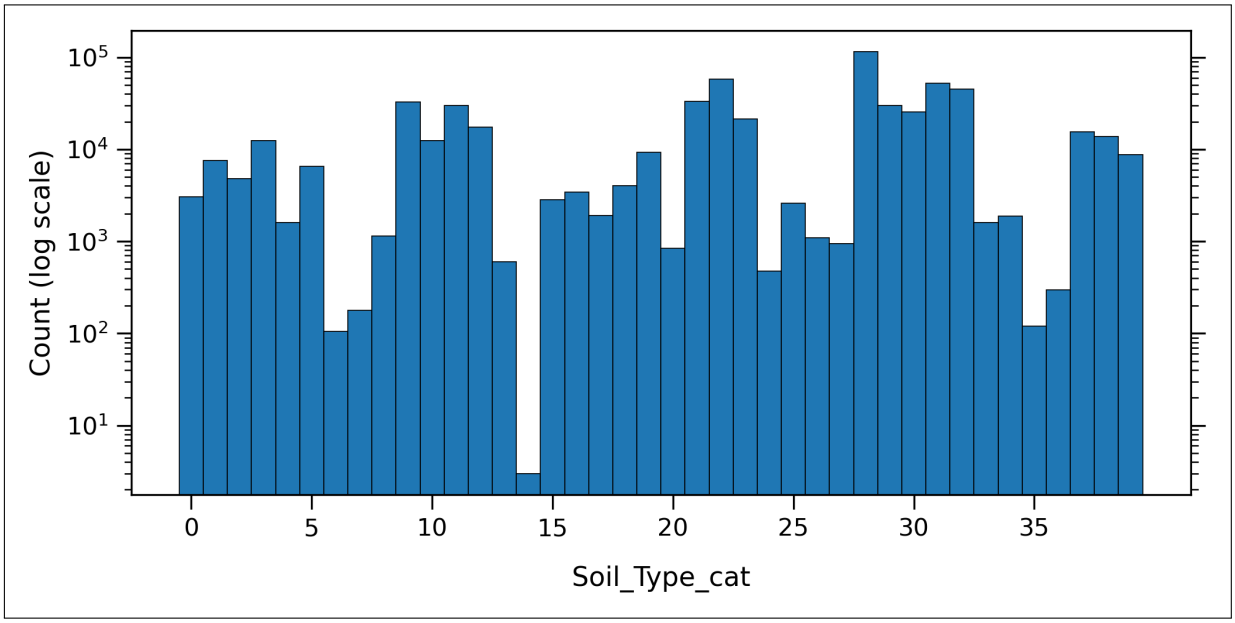

Figure 5: Covertype *Soil_ Type_ cat* frequency graph.

The target class was varied according to dataset-specific considerations. For Suricata, we fixed the target class as *Misc Attack* (class 5), corresponding to the lowest threat level in Suricata's taxonomy. For Covertype, we considered classes 1 and 3 after label remapping, which are respectively the most frequent and least frequent classes in the dataset. This choice allows us to examine the effect of target-class prevalence on attack effectiveness.

Combining these variables yields 63 poisoned configurations for Suricata (9 trigger compositions $\times$ 7 poison ratios $\times$ 1 target class) and 56 for Covertype (4 trigger compositions $\times$ 7 poison ratios $\times$ 2 target classes). Including the clean baseline, each experimental batch therefore comprises 64 configurations for Suricata and 57 for Covertype.

4.4.2 Repetitions and Random Seeds

The full set of configurations is repeated across multiple experimental batches to support reproducibility and statistical analysis. The Suricata experiments use 16 fixed base seeds, whereas the Covertype experiments use 32; the complete lists are given in Appendix A.2. For a given base seed, the refined dataset is split once into training and held-out partitions,

and the training partition is then further split into training and validation subsets as described earlier. All configurations within the same batch therefore share the same underlying data partition, so that within-batch differences are attributable to the attack configuration rather than to variation in the data split.

Each configuration within a batch is assigned a derived seed, denoted *run_seed*, to preserve reproducibility while ensuring distinct randomization across configurations. This seed is constructed from the tuple consisting of the base seed, poison ratio, target class, and trigger composition. To obtain a stable representation of the trigger composition, its feature–value pairs are first sorted by feature name. The resulting string is hashed with SHA-256, and the first 8 hexadecimal digits (32 bits) of the digest are converted to an integer, as detailed in Appendix A.3. This procedure makes *run_seed* deterministic for a given configuration while preventing different poison ratios on the same base split from generating nested poisoned subsets.

4.4.3 Training and Evaluation Protocol

For each poisoned configuration, defined by a poison ratio, trigger composition, and target class, a LightGBM model is trained using the hyperparameter configuration in Table 3. Training uses the poisoned training partition together with its associated validation split, and training metrics are not retained. After training, the model is evaluated on the clean held-out set, and the resulting metrics are stored for subsequent CAD computation relative to the clean baseline. A copy of the same held-out set is then fully poisoned according to the poisoning protocol, meaning that the trigger is applied to every held-out instance eligible for poisoning. The poisoned copy is evaluated by the model, and the resulting predictions and metrics are used to compute ASR.

For the baseline configuration, the model is trained on the corresponding unpoisoned training data derived from the same split, and training metrics are likewise not retained. The clean held-out set is then evaluated on the baseline model, and the resulting metrics are recorded as the reference for clean-data performance and for subsequent CAD computation.

5 Backdoor Injection Analysis

This chapter analyzes backdoor injection effectiveness under the experimental protocol defined previously. It first reports results for the Suricata dataset, focusing on the effects of poison ratio, trigger composition, and empirical feature support on attack success and clean-data performance. It then presents the corresponding results for Covertype, with emphasis on trigger type and target-class prevalence. All reported values are means with 95% confidence intervals computed using Student's t-distribution over 16 independent runs for Suricata and 32 for Covertype.

5.1 Suricata Models

We begin by establishing the performance of the clean reference model. It achieved $88.77 \pm 0.05\%$ accuracy on the held-out set, and Table 7 reports the remaining metrics, both per class and as weighted averages. The weighted F1-score was $95.01 \pm 0.02\%$. However, classes 2, 3, and 7 performed substantially worse than the others, especially in precision. This has limited effect on the weighted summary because these three classes together account for only 6.69% of the dataset.

The empirical feature distributions had a pronounced influence on the poison ratio required for single-feature triggers to achieve high ASR. Figures 6 and 7 show that the single src_port trigger was reliably successful only at 1%. At 0.5%, it became unstable across runs despite a mean ASR close to 100%, and no experiment using this trigger succeeded at 0.1% or below.

Single-value $packet\_size$ triggers tended to reach lower ASR plateaus as empirical support increased. The trigger $packet\_size = 80$ failed to reach the 90% ASR threshold at any poison ratio, although at 1% it achieved a highly consistent $89.77 \pm 0.05\%$ ASR. The trigger $packet\_size = 136$ exceeded the threshold at poison ratios above 0.05%, whereas $packet\_size = 240$ and $packet\_size = 1104$ remained effective even at 0.01%.

Class	Support	Precision (%)	Recall (%)	F1-score (%)
0: Admin Privilege	10958	99.97 ± 0.00	99.94 ± 0.01	99.95 ± 0.00
1: Info Leak	2087	99.73 ± 0.02	99.85 ± 0.02	99.79 ± 0.02
2: Crypto Mining	1106	29.42 ± 0.15	73.51 ± 0.38	42.02 ± 0.18
3: IP Scavenging	1970	47.79 ± 0.26	67.28 ± 0.36	55.89 ± 0.28
4: GPCD	8379	99.96 ± 0.01	99.95 ± 0.01	99.95 ± 0.01
5: Misc Attack	11060	99.85 ± 0.01	99.78 ± 0.01	99.81 ± 0.01
6: Misc Activity	11060	94.28 ± 0.05	84.50 ± 0.10	89.12 ± 0.06
7: PUP	1629	52.97 ± 0.16	93.75 ± 0.13	67.69 ± 0.11
8: Privacy Violation	11060	88.64 ± 0.07	80.63 ± 0.12	84.44 ± 0.07
9: Bad Traffic	11060	85.22 ± 0.10	73.19 ± 0.15	78.75 ± 0.10
Weighted average	–	91.29 ± 0.02	88.77 ± 0.03	89.56 ± 0.03

Table 7: Per-class and weighted average performance of the reference model on the Suricata dataset.

Taken together, the single-feature results for *src_port* and *packet_size* indicate that trigger effectiveness depends not only on empirical support, but also on feature-domain size. Although both *src_port* = 20123 and *packet_size* = 1104 are low-frequency values, *src_port* spans approximately 120 times more distinct values in the dataset than *packet_size*. The trigger *packet_size* = 1104 outperformed all other packet-size values and remained effective at lower poison ratios, whereas *src_port* = 20123 performed well only at 1%. These findings suggest that low-frequency single-feature triggers can be especially effective when the associated feature domain is small enough to be represented adequately by the model.

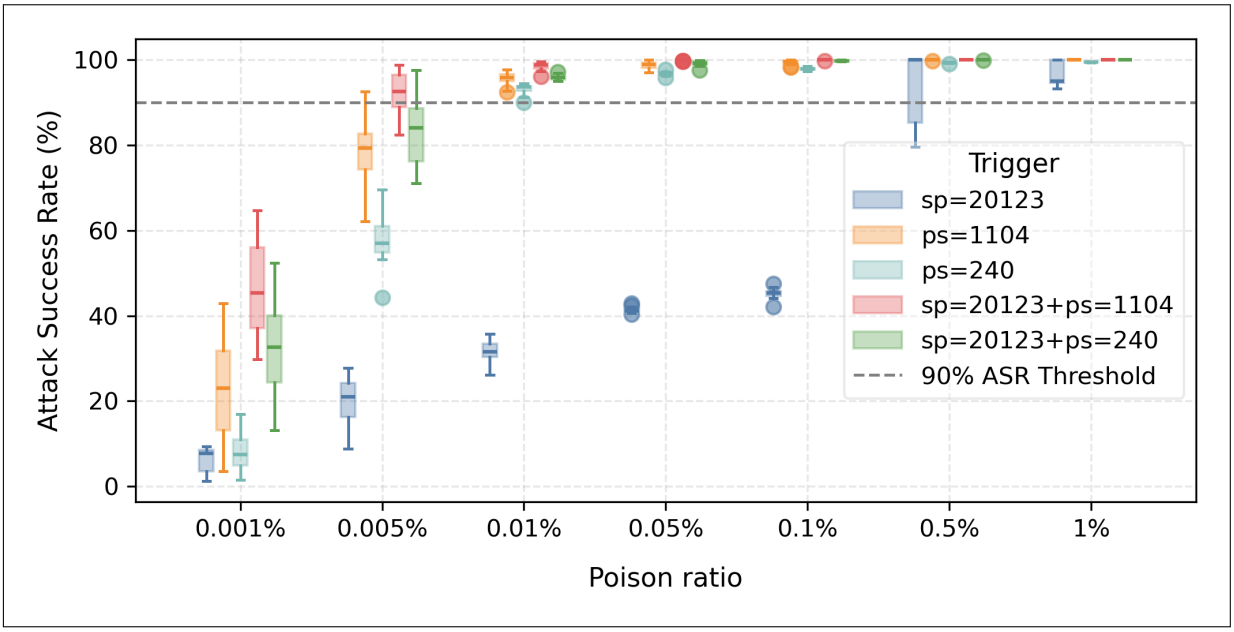

Figure 6: Suricata triggers, ASR by poison ratio, part 1.

Combining multiple features into a joint trigger provided a clear advantage: all combined configurations outperformed their individual constituent features at matched poison ratios. This indicates that multi-feature triggers require fewer poisoned samples to establish reliable backdoor behavior, thereby improving attack efficiency.

At lower poison ratios, 0.05% was sufficient for seven of the nine trigger variants, corresponding to 20 times fewer poisoned samples than the commonly used 1% setting. At 0.005%, no experiment remained fully consistent, although the combination of *src_port* and *packet_size* = 1104 produced more successful than unsuccessful runs, as suggested by the corresponding boxplot lying almost entirely above the 90% threshold. No trigger succeeded at 0.001%.

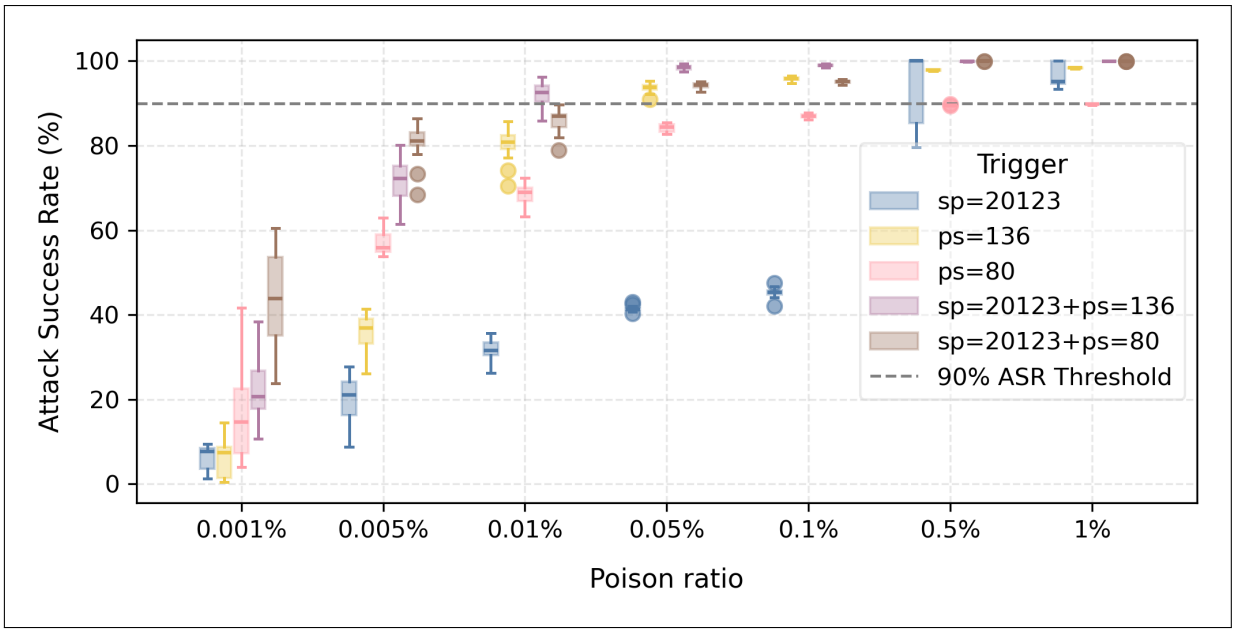

Figure 7: Suricata triggers, ASR by poison ratio, part 2.

In terms of CAD, all settings remained within a 1% margin relative to the reference model, as shown in Figures 8 and 9. At the 1% poison ratio, a slight but consistent increase in accuracy degradation was observed for triggers based on the lowest-frequency values, namely *src_port* = 20123, *packet_size* = 1104, and their combination. These results indicate that the target ASR is attainable within the CAD constraint, although low-frequency trigger values may impose a greater cost on clean-data performance at higher poison ratios.

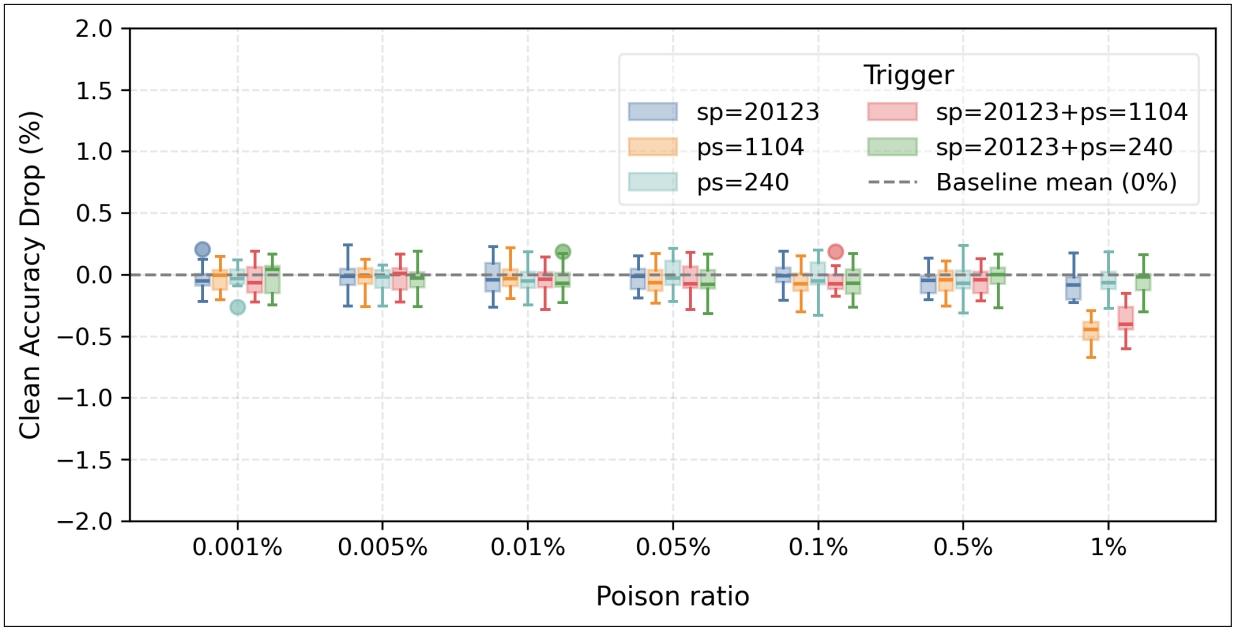

Figure 8: Suricata triggers, CAD by poison ratio, part 1.

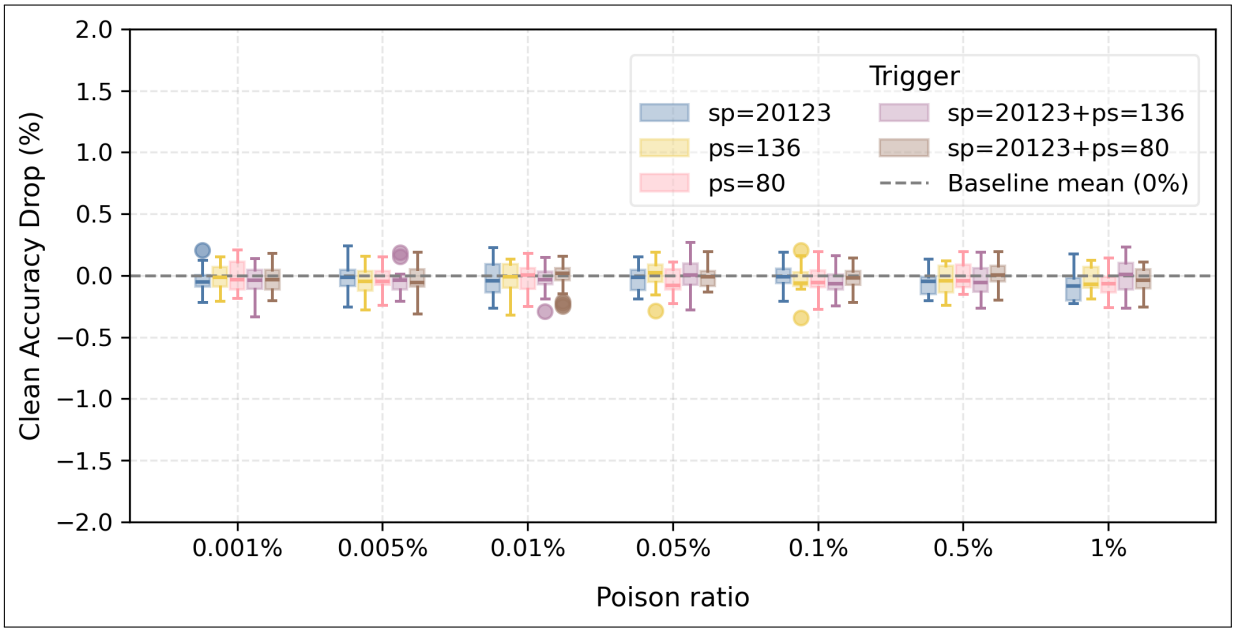

Figure 9: Suricata triggers, CAD by poison ratio, part 2.

Table 8 summarizes the successful runs. Because no configuration violated the CAD constraint, the table includes all settings that achieved the 90% ASR threshold, for a total of 563 successful runs.

Trigger	Poison Ratio							Total
	0.001%	0.005%	0.01%	0.05%	0.1%	0.5%	1%	
$\{\emptyset, 20123\}$	0	0	0	0	0	10	16	26
$\{1104, \emptyset\}$	0	1	16	16	16	16	16	81
$\{240, \emptyset\}$	0	0	15	16	16	16	16	79
$\{136, \emptyset\}$	0	0	0	16	16	16	16	64
$\{80, \emptyset\}$	0	0	0	0	0	0	0	0
$\{1104, 20123\}$	0	11	16	16	16	16	16	91
$\{240, 20123\}$	0	2	16	16	16	16	16	82
$\{136, 20123\}$	0	0	12	16	16	16	16	76
$\{80, 20123\}$	0	0	0	16	16	16	16	64
Total	0	14	75	112	112	122	128	563

Table 8: Successful attacks for each Suricata configuration. Triggers are in the form $\{packet\_size, src\_port\}$.

5.2 Covertype Models

Throughout this section, we use the following shorthand labels for the trigger features: **A** for *Elevation*, **B** for *Horizontal Distance to Roadways*, and **C** for *Soil Type*.

Before analyzing the poisoned models, we established the performance of the clean reference model on the held-out set. It achieved an accuracy of $96.93 \pm 0.02\%$. Table 9 reports the remaining per-class and weighted-average metrics.

Class	Support	Precision (%)	Recall (%)	F1-score (%)
0: Spruce/Fir	21184	97.02 ± 0.04	96.83 ± 0.04	96.93 ± 0.03
1: Lodgepole Pine	28331	97.51 ± 0.03	97.30 ± 0.04	97.40 ± 0.02
2: Ponderosa Pine	3575	96.39 ± 0.11	96.79 ± 0.11	96.59 ± 0.08
3: Cottonwood/Willow	275	89.54 ± 0.65	90.33 ± 0.66	89.92 ± 0.47
4: Aspen	949	88.96 ± 0.36	92.52 ± 0.40	90.70 ± 0.28
5: Douglas-fir	1737	93.64 ± 0.27	94.91 ± 0.16	94.27 ± 0.14
6: Krummholz	2051	96.61 ± 0.14	97.81 ± 0.11	97.21 ± 0.09
Weighted average	–	96.94 ± 0.02	96.93 ± 0.02	96.93 ± 0.02

Table 9: Per-class and weighted average performance of the reference model on the Covertype dataset.

For target class 1 (Figure 10), ASR approached 100% for all proposed triggers at poison ratios of 1% and 0.5%, with a single extreme outlier observed for A+C. At 0.1%, the triple trigger (A+B+C) remained close to 100% with low variance, whereas the pairwise triggers began to degrade and exhibit greater variability, although they still clustered above the 90% threshold. At 0.05%, A+B+C remained near 100%, but outliers became more frequent, including one failed run. Among the pairwise triggers, B+C occasionally dropped below 90%, and A+C was the least consistent despite a mean ASR above 90%. At 0.01%, only A+B+C remained clearly effective overall, although some runs fell below the 90% threshold. At lower poison ratios, no class-1 trigger remained consistently successful.

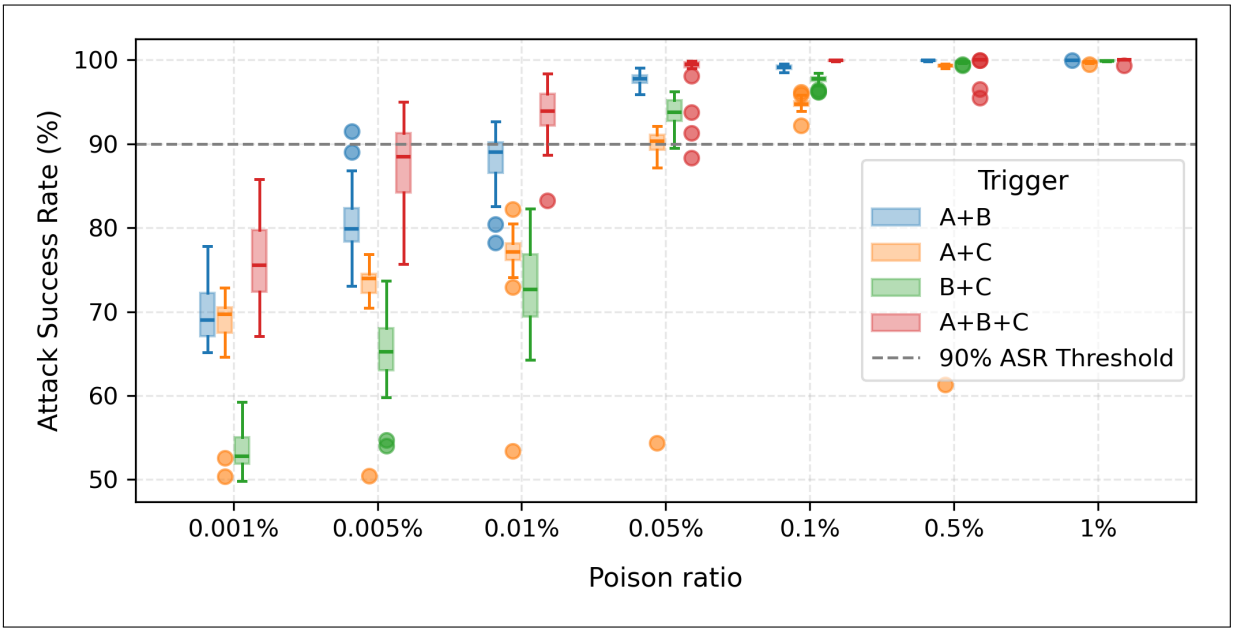

Figure 10: Covertype triggers: ASR by poison ratio for target class 1.

For target class 3 (Figure 11), ASR remained near 100% with low variance from 0.05% to 1%, although occasional catastrophic outliers near 0% ASR were still observed. At 0.01%, performance began to degrade, but only A+C failed completely. At 0.005%, A+B+C was the only trigger that remained consistently above the 90% threshold, whereas B+C succeeded with inconsistencies and the rest failed. No experiments were successful at 0.001%.

Regarding the rare catastrophic outliers, their accuracy approached the prevalence of the predicted class, suggesting that the model failed to learn the intended trigger. The stochasticity of training, together with feature subsampling and bagging fractions below 1.0, can occasionally prevent reliable backdoor implantation.

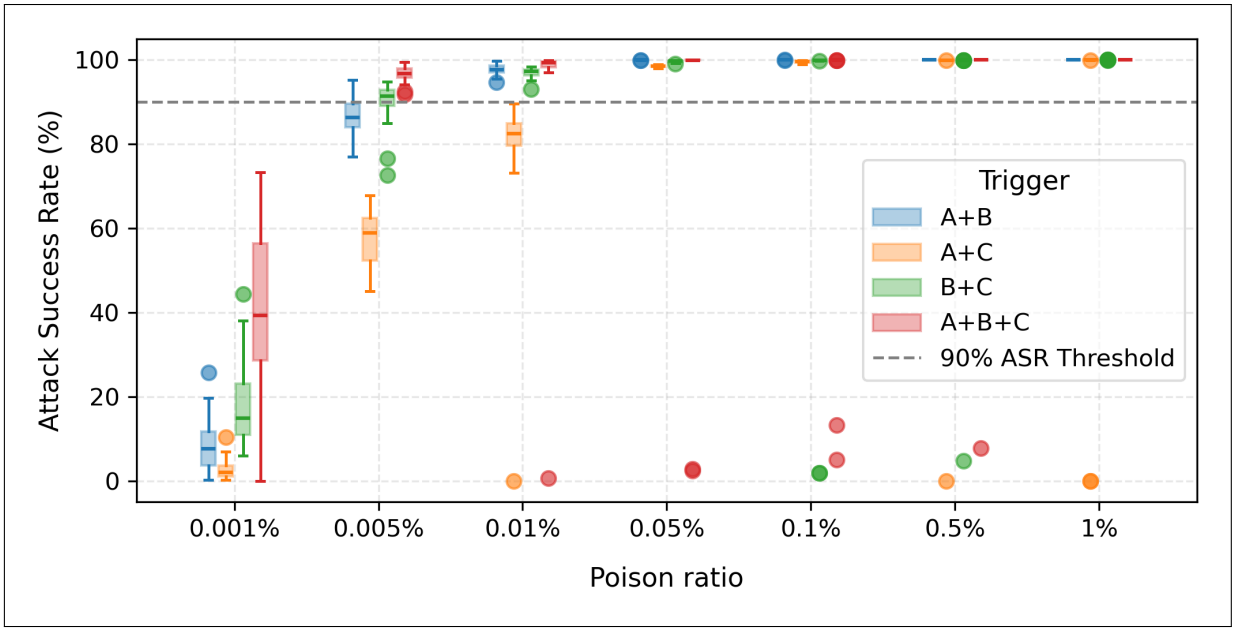

Figure 11: Covertype triggers: ASR by poison ratio for target class 3.

Across both target classes, ASR declined more rapidly for class 1 as the poison ratio decreased, suggesting that target-class prevalence substantially influences the amount of poisoned data required to induce reliable label flips. Among the triggers, the more complex A+B+C combination consistently outperformed the simpler alternatives at matched poison ratios for both target classes, and it remained effective even when the pairwise triggers degraded. Among the feature pairs, A+C was the weakest for both targets, suggesting that the categorical value used in C provides a weaker separating cue when combined with a high-frequency numerical value.

CAD remained tightly concentrated around the baseline. Across all configurations, the central mass of the distribution stayed within roughly ± 0.25 percentage points of the baseline, well inside the 1% success criterion (Figures 12 and 13). A small number of outliers were observed; in the most extreme cases, a few runs deviated by roughly 9–10 percentage points from baseline performance.

Table 10 summarizes the counts of successful attacks across all settings. Overall, 1,202 runs met the 90% ASR threshold, but ten of these failed to meet the degradation criterion, leaving 1,192 successfully poisoned models.

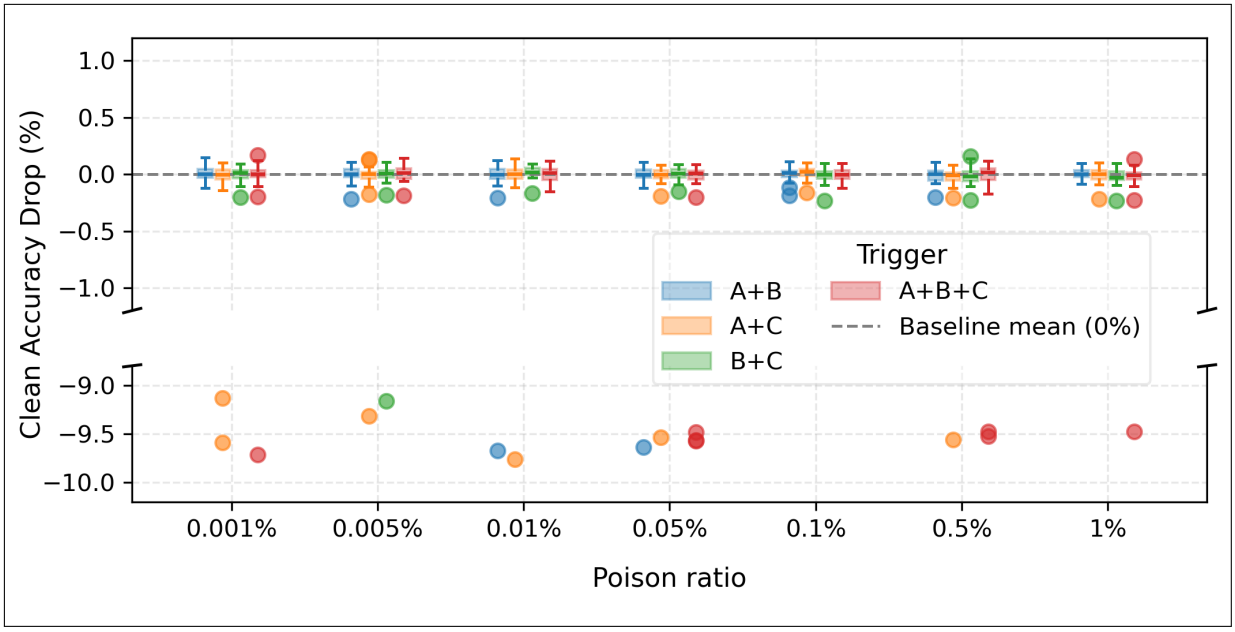

Figure 12: Covertype triggers: CAD by poison ratio for target class 1.

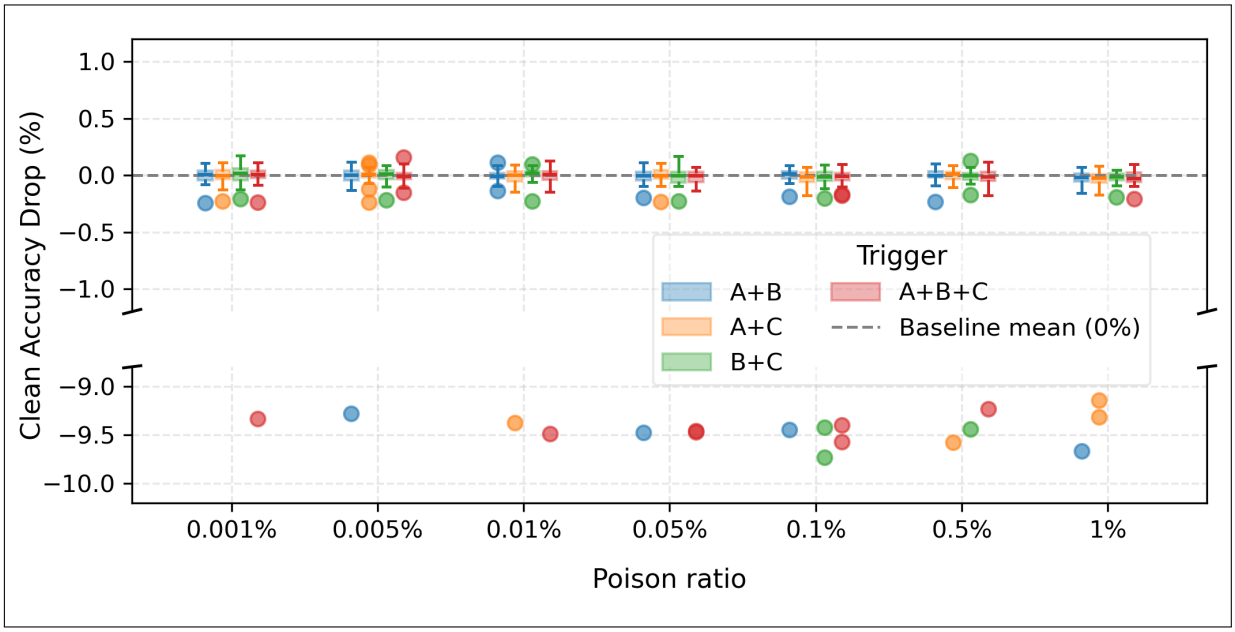

Figure 13: Covertype triggers: CAD by poison ratio for target class 3.

Trigger	**Poison Ratio**							**Total**
	0.001%	**0.005%**	**0.01%**	**0.05%**	**0.1%**	**0.5%**	**1%**	
A+B	0 | 0	1 | 7	10 | 32	31 | 31	32 | 31	32 | 32	32 | 31	138 | 164
A+C	0 | 0	0 | 7	0 | 0	18 | 32	32 | 32	31 | 31	32 | 30	113 | 125
B+C	0 | 0	0 | 22	0 | 32	31 | 32	32 | 30	32 | 31	32 | 32	127 | 179
A+B+C	0 | 0	11 | 32	27 | 31	29 | 30	32 | 30	30 | 31	31 | 32	160 | 186
Total	0 | 0	12 | 61	37 | 95	109 | 125	128 | 123	125 | 125	127 | 125	538 | 654

Table 10: Successful attacks for each Covertype configuration. Each entry $a \mid b$ corresponds to class 1 and class 3, respectively.

6 Backdoor Detection

This chapter addresses backdoor detection in poisoned LightGBM models. It first characterizes trigger search over split-induced intervals, which provide a finite representation of the candidate space. It then introduces a budget-aware detection pipeline, followed by two heuristics for ranking candidate intervals: *expected logits* and *pre-leaf intervals*.

6.1 Trigger Search

As discussed in the introduction, research on backdoors in decision-tree models remains limited. Ribeiro et al. introduced Anchor (RIBEIRO; SINGH; GUESTRIN, 2018), a model-agnostic method for identifying high-precision predicates, but its effectiveness depends on a predefined predicate set and on the local sampling distribution. Parmentier and Vidal proposed OCEAN (PARMENTIER; VIDAL, 2021), a Mixed-Integer Linear Programming (MILP) formulation for counterfactual analysis in tree ensembles that computes optimal perturbations for reaching a target class.

Although both approaches can be adapted to backdoor trigger exploration, they have important limitations. Anchor may fail to detect highly specific or infrequently sampled triggers because its success depends on adequate local sampling coverage. OCEAN, while effective, incurs computational costs that grow with ensemble size and depth, which can make it impractical for large models.

Many backdoor defenses for vision models are based on *model probing*. Given a clean dataset, or a suitable proxy, the defender searches for an *input perturbation* that consistently induces a chosen target label while maintaining a minimum level of attack effectiveness. One example is GangSweep (ZHU et al., 2020), which uses a GAN-based formulation to recover minimal perturbation patterns that explain anomalous target-label behavior.

This formulation is not directly applicable to decision-tree-based models. Tree pre-

dictors are piecewise-constant functions of the input and are therefore non-smooth, with discontinuities at split boundaries (BLOCKEEL et al., 2023). As a result, small input modifications typically do not induce smooth changes in the model output, and input gradients are not informative for optimizing an adversarial generator. In practice, this makes gradient-based trigger-recovery methods such as GangSweep poorly suited to tree ensembles.

In a decision-tree model, trigger search aims to identify a specific combination of feature values that flips the prediction of a sample from its original class to a chosen target class. The analogue of a minimal perturbation in this setting is a *sparse trigger*, that is, a trigger that modifies as few features as possible, regardless of the magnitude of the assigned values. However, unlike pixels, tabular features may lie in large discrete or continuous domains, making naive brute-force search computationally infeasible.

The key observation is that tree ensembles operate through threshold tests and category partitions, thereby inducing a piecewise-constant partition of the feature space. This collapses each feature's raw domain into a finite set of *equivalence classes*.

For a numerical feature x_f, each split introduces a threshold t and evaluates the condition $x_f \leq t$. The collection of all such thresholds across the ensemble induces a finite partition of the real line into intervals. Any two values that fall within the same interval are indistinguishable to the model with respect to decisions involving feature f. We denote these equivalence classes by $I_{f,j}$ and refer to them as *split-induced intervals*. Categorical features admit an analogous construction, since each split partitions the category set into two disjoint subsets. Consequently, for each feature f, it suffices to consider the finite set $\mathcal{L}_f = \{I_{f,j}\}$ rather than the full raw domain. A procedural description of how these intervals are constructed from the ensemble is provided in Appendix A.4.

Despite reducing the raw domain to a finite set of equivalence classes, trigger search remains vulnerable to *combinatorial explosion*. Let k denote the number of poisonable features, and let $n_i = |\mathcal{L}_i|$ denote the number of admissible intervals for feature i. If each feature is allowed to contribute at most one interval condition, then the total number of distinct candidate triggers across all non-empty feature subsets is

$$N = \sum_{\emptyset \neq S \subseteq \{1,\ldots,k\}} \prod_{i \in S} n_i, \tag{6.1}$$

which grows exponentially in k and multiplicatively in the n_i. In realistic boosted ensembles, numerical features may induce hundreds of intervals, making exhaustive search

impractical even for low-cardinality triggers.

The evaluation cost compounds this difficulty. Assessing whether a candidate trigger *consistently* redirects predictions to a target class requires probing the model on a set of test samples. If the true success probability is p, then an estimate based on s samples has standard error on the order of $\sqrt{p(1-p)/s}$. Consequently, small values of p or stringent confidence requirements require large values of s.

Two bottlenecks therefore make the problem intractable in practice. First, brute-force enumeration of trigger candidates, together with the construction of sufficiently large test batches for reliable evaluation, can be extremely costly. Second, all such batches must be evaluated on the model, which may be even more expensive depending on ensemble size. A natural first step, therefore, is to devise a candidate-construction strategy that avoids brute-force enumeration.

The next three sections introduce a backdoor-detection pipeline for constructing and evaluating test candidates, together with two heuristics that guide the selection of feature intervals.

6.2 Backdoor Detection Pipeline

Given a model to be inspected, it is unknown whether a trigger is present and, if so, which class it targets. All classes must therefore be evaluated as potential backdoor targets. For each class, we compute trigger candidates, build test batches, and probe the model. Test batches are constructed by following the poisoning protocol described earlier: we draw a stratified subset of held-out samples whose ground-truth label differs from the class under test and apply the candidate trigger to them.

The pipeline begins by computing the split-induced intervals associated with each feature, as defined in the previous section. These intervals form the finite feature-wise search space on which the subsequent ranking and candidate-construction stages operate. Next, one of the two heuristics introduced later assigns a class-specific score to each split-induced interval using the *leaf_values* learned by the model.

After ranking, trigger candidates are constructed. Because the number of possible trigger assignments grows rapidly, as shown in Equation 6.1, this work employs Algorithm 1, a budget-aware class-wise interval-selection method referred to as BAIS. The algorithm builds trigger candidates under three constraints: a total probing budget B, a

sample size s, and a maximum trigger cardinality max_card.

The core idea of BAIS is to distribute the budget evenly across classes, since a triggered model may target any of them. It also constructs triggers from lower to higher cardinalities, both because our backdoor analysis focuses on low-cardinality triggers and because this reflects the adversary's incentive to keep triggers sparse and therefore stealthy.

Algoritmo 1 Budget-Aware Class-wise Interval Selection

Input: Ranked split-induced intervals $\mathcal{L} = \{I_{f,j}\}$ with associated class scores $\hat{z}_c(I_{f,j})$ and probabilities $\hat{p}_c(I_{f,j})$ for each class c; sample size s; budget B; maximum trigger cardinality max_card
Output: Set of selected intervals $\mathcal{S}$
Initialize $\mathcal{S} \leftarrow \emptyset$
$\mathcal{L} \leftarrow \{I_{f,j} \mid \text{all features } f \text{ and intervals } j\}$
while $\mathcal{L} \neq \emptyset$ **do**
  $c \leftarrow LeastRepresentedAvailableClass(\mathcal{L}, \mathcal{S})$
  $I \leftarrow TopRankedAvailableInterval(\mathcal{L}, \mathcal{S}, c)$
  Remove I from $\mathcal{L}$
  $\mathcal{S}_{tmp} \leftarrow \mathcal{S} \cup \{I\}$
  $cost \leftarrow ComputeCost(\mathcal{S}_{tmp}, max\_card, s)$
  if $cost \leq B$ **then**
    $\mathcal{S} \leftarrow \mathcal{S}_{tmp}$
  end if
end while
return $\mathcal{S}$

The algorithm begins with the full pool of ranked intervals. At each iteration, it identifies the class that is currently least represented in the selected set relative to the remaining candidate pool. It then selects, for that class, the highest-ranked interval that has not yet been chosen. This interval is added to a temporary set $\mathcal{S}_{tmp}$, and the probing cost of the resulting selection is evaluated.

The cost is computed separately for each class. For a given class c, only the intervals in $\mathcal{S}_{tmp}$ associated with c are considered. These intervals are grouped by feature, and n_i^c denotes the number of selected intervals from feature i for class c. Candidate triggers are then formed by combining intervals across features, with the restriction that at most max_card features may appear in a trigger. Let $N_c(max\_card)$ denote the number of such candidate combinations for class c, obtained by applying Equation 6.1 to the vector $(n_1^c, \ldots, n_k^c)$ while restricting the summation to feature subsets of cardinality at most max_card.

The overall probing cost is defined as

$$cost = s \sum_{c=1}^{C} N_c(max\_card), \tag{6.2}$$

where s is the number of clean samples used to evaluate each candidate. If this cost does not exceed the available budget B, the interval is retained and $\mathcal{S}$ is updated; otherwise, it is discarded. The process continues until no candidate intervals remain.

After all trigger candidates have been built, the final step is to construct test batches by applying each candidate trigger to a stratified subsample of size s drawn from the held-out set and then probing the model. If a candidate shifts predictions strongly toward the class under test, it is treated as a recovered trigger. Here, we use the same threshold as in the ASR criterion: a prediction rate above 90% for the class under test is taken as evidence that a trigger has been found and that the model is poisoned.

In its current form, BAIS selects each split-induced interval at most once. Moreover, because trigger construction is performed class-wise, the algorithm combines only intervals that receive high scores for the class under consideration. Consequently, if the heuristic fails to rank a true trigger interval among the top intervals for its target class, BAIS will not recover the backdoor. By chance, the algorithm may still select the true trigger interval, or the intervals corresponding to a multi-feature trigger, under an incorrect target class. In that case, the final probing stage may reveal a spike in predictions toward the incorrect class. Resolving this requires an additional round of testing with a newly defined subsample that excludes that newly indicated class.

The next two sections detail the interval-ranking heuristics and report their performance on all successfully poisoned models identified in the backdoor injection analysis chapter.

6.3 First Heuristic: Expected Logits

Inspired by GangSweep, the first scoring heuristic in the inspection pipeline aims to quantify how strongly a given split-induced interval biases the ensemble toward a particular class. More specifically, for each class, it estimates the predictive strength of each interval so that split-induced intervals can be ranked class-wise before budget-aware candidate selection.

This is nontrivial in tree ensembles. The predictive effect of a single split-induced interval cannot be measured directly because leaves are typically reached through sequences of split decisions involving multiple features, rather than through a condition on a single feature alone. As a result, the contribution of an individual feature is coupled with the values of the remaining features, which makes isolated evaluation impractical.

Algorithm 2 summarizes the procedure. For each split-induced interval and each class, we identify all leaves whose decision paths are compatible with that interval, regardless of the values taken by the remaining features. Within each boosting round, we average the *leaf_value* over all compatible leaves and then sum these averages across rounds to obtain an accumulated class score. After processing all classes, we apply a softmax function to the accumulated scores to derive a class-probability estimate for the interval.

Algoritmo 2 Expected Logits

Input: LGBM model M; split-induced intervals $\mathcal{L}_f$ for each feature f
Output: For each interval $I_{f,j} \in \mathcal{L}_f$: expected logits $\hat{z}_c(I_{f,j})$ and softmax probabilities $\hat{p}_c(I_{f,j})$ for all classes $c \in \{1,\ldots,C\}$
for each feature f **do**
  for each interval $I_{f,j} \in \mathcal{L}_f$ **do**
    Choose a representative value $x_f \in I_{f,j}$
    for each class $c \in \{1,\ldots,C\}$ **do**
      $\hat{z}_c \leftarrow 0$
      for each estimator round $r = 1,\ldots,R$ **do**
        Build a list $\mathcal{V}_{f,j,c,r}$ containing all leaves for class c in round r whose decision paths are compatible with x_f
        $rc\_logit \leftarrow \dfrac{1}{|\mathcal{V}_{f,j,c,r}|} \displaystyle\sum_{\ell \in \mathcal{V}_{f,j,c,r}} leaf\_value(\ell)$
        $\hat{z}_c \leftarrow \hat{z}_c + rc\_logit$
      end for
    end for
    Softmax: $\hat{p}_c \leftarrow \dfrac{\exp(\hat{z}_c)}{\sum_{c'=1}^{C} \exp(\hat{z}_{c'})} \quad \forall c$
    Store $(I_{f,j}, \hat{z}_c, \hat{p}_c)$
  end for
end for

This computation provides two useful signals. First, it estimates the *potential* contribution of a split-induced interval to the final logits independently of the values of the other features, thereby enabling class-wise interval ranking. Second, it yields an estimate of the probability that the interval favors each class. The underlying assumption is that a successful backdoor interval will disproportionately favor its target class, so that the leaves compatible with that interval collectively skew the accumulated logits toward that class.

We first apply this heuristic to the 563 successfully poisoned Suricata models that satisfy the ASR and CAD constraints. From this point onward, for a single-feature trigger, we define a *recovery* as the case in which the true trigger interval is ranked first for the target class of the corresponding model. For multi-feature triggers, *full recovery* requires this to hold for all constituent trigger intervals.

For single-feature triggers, the heuristic recovered 234 of the 250 successfully poisoned models, corresponding to a 93.6% recovery rate, as summarized in Table 11. The only notably weak result was for *packet_size* = 240 at 0.01%, for which the heuristic recovered only 4 of 15 cases.

Ratio	**Trigger (target class 5)**					**Total**
	***ps* = 80**	***ps* = 136**	***ps* = 240**	***ps* = 1104**	***sp* = 20123**	
0.005%	0/0	0/0	0/0	1/1	0/0	1/1
0.01%	0/0	0/0	4/15	16/16	0/0	20/31
0.05%	0/0	14/16	14/16	16/16	0/0	44/48
0.1%	0/0	16/16	15/16	16/16	0/0	47/48
0.5%	0/0	16/16	16/16	16/16	10/10	58/58
1%	0/0	16/16	16/16	16/16	16/16	64/64
Total	0/0	62/64	65/79	81/81	26/26	234/250

Table 11: Success matrix by Expected Logits for the single-trigger backdoor on Suricata models targeting class 5. Each entry a/b indicates the number of seeds a for which the trigger was successfully recovered out of b seeds whose models were successfully poisoned.

For two-feature triggers, Table 12 presents a more nuanced picture: only 167 of 313 successfully poisoned models were fully recovered, corresponding to a 53.35% recovery rate. The pair {80, 20123} was fully recovered in 37 of 64 cases. Within this pair, however, *src_port* was recovered in all 64 cases, making *packet_size* the limiting factor for full recovery. For the pair {136, 20123}, no full recovery occurred: *packet_size* failed in all instances, whereas *src_port* was still recovered in 72 of 76 cases. The pair {240, 20123} also had a modest recovery rate, with 45 full recoveries out of 82 cases; again, *packet_size* was the limiting factor, being recovered in only 47 cases, whereas *src_port* was recovered in 76. Finally, the pair {1104, 20123} achieved a stronger recovery rate, with 85 full recoveries out of 91 cases. In this case, *packet_size* was recovered in all models, while *src_port* became the limiting factor for full recovery.

Overall, the Expected Logits heuristic fully recovered 401 of the 563 successfully poisoned Suricata models, corresponding to a 71.23% recovery rate across all trigger types. These results suggest that, for single-feature triggers, the heuristic remains effective across

Ratio	Trigger (target class 5)				Total
	{80, 20123}	**{136, 20123}**	**{240, 20123}**	**{1104, 20123}**	
0.005%	[0,0] : 0/0	[0,0] : 0/0	[1,2] : 1/2	[11,10] : 10/11	11/13
0.01%	[0,0] : 0/0	[0,12] : 0/12	[0,14] : 0/16	[16,14] : 14/16	14/44
0.05%	[6,16] : 6/16	[0,16] : 0/16	[5,15] : 5/16	[16,16] : 16/16	27/64
0.1%	[8,16] : 8/16	[0,15] : 0/16	[9,15] : 9/16	[16,14] : 14/16	31/64
0.5%	[10,16] : 10/16	[0,14] : 0/16	[16,15] : 15/16	[16,15] : 15/16	40/64
1%	[13,16] : 13/16	[0,15] : 0/16	[16,15] : 15/16	[16,16] : 16/16	44/64
Total	[37,64] : 37/64	[0,72] : 0/76	[47,76] : 45/82	[91,85] : 85/91	167/313

Table 12: Success matrix by Expected Logits for the paired-trigger backdoor on Suricata models targeting class 5. Each entry of the form $[X,Y] : a/b$ reports per-feature recovery counts $[X,Y]$ and the number of seeds a for which the trigger $\{X,Y\}$ was successfully recovered out of b seeds whose models were successfully poisoned.

different empirical frequencies and poison ratios. For more complex triggers, however, it becomes less effective at recovering high-frequency constituents, although recovery rates generally improve as the poison ratio increases. With the exception of the complete failure of the pair {136, 20123}, this pattern suggests that higher poisoning rates imprint stronger signals in the ensemble and thereby make trigger constituents easier for the heuristic to identify.

For the 1,192 successfully poisoned Covertype models, the results in Tables 13 and 14 reveal a strong dependence on the target class. For target class 1, the heuristic achieved a 100% full recovery rate across all 538 models, indicating that it is well suited to *high-prevalence* target classes. These results also show that the heuristic can recover categorical trigger features, a setting not explored in the Suricata experiments.

Ratio	Trigger (target class 1)				Total
	A+B	**A+C**	**B+C**	**A+B+C**	
0.005%	[1,1] : 1/1	[0,0] : 0/0	[0,0] : 0/0	[11,11,11] : 11/11	12/12
0.01%	[10,10] : 10/10	[0,0] : 0/0	[0,0] : 0/0	[27,27,27] : 27/27	37/37
0.05%	[31,31] : 31/31	[18,18] : 18/18	[31,31] : 31/31	[29,29,29] : 29/29	109/109
0.1%	[32,32] : 32/32	[32,32] : 32/32	[32,32] : 32/32	[32,32,32] : 32/32	128/128
0.5%	[32,32] : 32/32	[31,31] : 31/31	[32,32] : 32/32	[30,30,30] : 30/30	125/125
1%	[32,32] : 32/32	[32,32] : 32/32	[32,32] : 32/32	[31,31,31] : 31/31	127/127
Total	[138,138] : 138/138	[113,113] : 113/113	[127,127] : 127/127	[160,160,160] : 160/160	538/538

Table 13: Success matrix by Expected Logits for the Covertype triggers targeting class 1. Each entry of the form $[X,Y,Z] : a/b$ reports per-feature recovery counts $[X,Y,Z]$, where Z is optional, and the number of seeds a for which the trigger $\{X,Y,Z\}$ was successfully recovered out of b seeds whose models were successfully poisoned.

In stark contrast, for target class 3, the heuristic succeeded in only 17 of the 654 models, corresponding to a recovery rate of 2.6%. Feature A (*Elevation*) was never recovered, feature B (*Horizontal Distance to Roadways*) was recovered in 119 cases, and feature C (*Soil Type*) in only 17 cases, all at poison ratios of at least 0.5%. These results suggest that the Expected Logits heuristic is not effective at detecting triggers for *low-prevalence* target classes. For the low-support numerical feature value B, recovery depended strongly on poison ratio: models were successfully poisoned at much lower ratios, but recovery occurred only at 0.5% or above. Detection of the categorical feature C was even less consistent.

Ratio	**Trigger (target class 3)**				**Total**
	A+B	**A+C**	**B+C**	**A+B+C**	
0.005%	[0,0] : 0/7	[0,0] : 0/0	[0,0] : 0/22	[0,0,0] : 0/32	0/61
0.01%	[0,0] : 0/32	[0,0] : 0/0	[0,0] : 0/32	[0,0,0] : 0/31	0/95
0.05%	[0,0] : 0/31	[0,0] : 0/32	[0,0] : 0/32	[0,0,0] : 0/30	0/125
0.1%	[0,0] : 0/31	[0,0] : 0/32	[0,0] : 0/30	[0,0,0] : 0/30	0/123
0.5%	[0,32] : 0/32	[0,0] : 0/31	[24,0] : 0/31	[0,0,0] : 0/31	0/125
1%	[0,31] : 0/31	[0,0] : 0/30	[32,17] : 17/32	[0,0,0] : 0/32	17/125
Total	[0,63] : 0/164	[0,0] : 0/125	[56,17] : 17/179	[0,0,0] : 0/186	17/654

Table 14: Success matrix by Expected Logits for the Covertype triggers targeting class 3. Each entry of the form $[X,Y,Z] : a/b$ reports per-feature recovery counts $[X,Y,Z]$, where Z is optional, and the number of seeds a for which the trigger $\{X,Y,Z\}$ was successfully recovered out of b seeds whose models were successfully poisoned.

Taken together, the Suricata and Covertype results indicate that the Expected Logits heuristic can recover both numerical and categorical trigger features, but its effectiveness depends strongly on target-class prevalence. Poison ratio also matters, but its effect appears secondary to the prevalence of the target class.

6.4 Second Heuristic: Pre-leaf Intervals

This heuristic adopts the working assumption that, when a trigger is learned under a limited poisoning budget, trigger-related features tend to appear in highly specific decision rules close to the final prediction, namely in splits that immediately precede leaf nodes. Such placement localizes a strong logit shift to a narrow region of the feature space while minimizing its effect elsewhere.

For numerical features, the ensemble need not split exactly at the trigger value x. Instead, it may bracket the trigger by placing thresholds on either side of x. In that case,

evidence for the trigger is distributed across complementary half-constraints: a lower threshold contributes target-class logit mass on its right side, whereas an upper threshold contributes target-class logit mass on its left side. The overlap of these contributions defines an interval that contains the trigger value.

Pre-leaf interval scoring follows from this observation. For each numerical feature and class, we collect the logits associated with pre-leaf splits, recording the left- and right-leaf mass at each threshold. Candidate intervals are then scored by pairing the right mass of a lower threshold with the left mass of a higher threshold, that is, $R(\text{lower}) + L(\text{upper})$, as detailed in Algorithm 3. Intuitively, intervals that contain a learned trigger should accumulate unusually large target-class evidence from both sides across the ensemble.

These interval scores are then projected onto the split-induced intervals. A per-interval softmax is subsequently applied to normalize the class scores, yielding a compact mapping from feature intervals to class preference. This representation reflects how the ensemble concentrates its most specific class-discriminative rules near the leaves.

This heuristic is specific to numerical features. Categorical splits are membership tests over unordered sets of values and therefore do not admit a natural interval or adjacency structure, which makes pre-leaf aggregation inapplicable. We therefore restrict the pre-leaf analysis to numerical features.

For the Suricata backdoored models, Tables 15 and 16 show that the Pre-Leaf Intervals heuristic fully recovered 242 of 250 successfully poisoned single-trigger models and 253 of 313 successfully poisoned two-trigger models. This corresponds to an overall recovery rate of 87.92% (495/563), a substantial improvement over the 71.23% achieved by the Expected Logits heuristic on the same models. The gain is especially clear for two-feature triggers, for which recovery rises from 53.35% to 80.83%.

For the Covertype backdoored models, the Pre-Leaf Intervals heuristic behaved very differently depending on the target class. For target class 1, Table 17 shows an overall recovery rate of 56.69% (305/538), with better results at higher poison ratios. This is markedly worse than the 100% recovery achieved by Expected Logits for the same target class.

At the feature level, performance also depends on trigger composition. Feature A (*Elevation*) was recovered 94 out of 113 times when paired with C (*Soil Type*), 69 out of 138 times when paired with B (*Horizontal Distance to Roadways*), and 40 out of 160 times when included in the three-feature trigger, corresponding to recovery rates of 83.19%,

Algoritmo 3 Pre-leaf Intervals

Input: LGBM model M; split-induced intervals $\mathcal{L}_f$ for each numerical feature f
Output: For each interval $I_{f,j} \in \mathcal{L}_f$: logits $\hat{z}_c(I_{f,j})$ and softmax probabilities $\hat{p}_c(I_{f,j})$ for all classes $c \in \{0,\ldots,C-1\}$
Pre-leaf accumulators: $L[c,f,t] \leftarrow 0$ and $R[c,f,t] \leftarrow 0$
Observed pre-leaf thresholds: $E[c,f] \leftarrow \emptyset$
Step 1: Collect pre-leaf thresholds and accumulate logits.
for all trees $\tau \in M$ **do**
  $c \leftarrow \text{classOf}(\tau)$
  for all split nodes n in τ with numerical feature f and threshold t **do**
    if leftChild(n) is a leaf with value ℓ **then**
      $L[c,f,t] \leftarrow L[c,f,t] + \ell$
      $E[c,f] \leftarrow E[c,f] \cup \{t\}$
    end if
    if rightChild(n) is a leaf with value ℓ **then**
      $R[c,f,t] \leftarrow R[c,f,t] + \ell$
      $E[c,f] \leftarrow E[c,f] \cup \{t\}$
    end if
  end for
end for
Step 2: Build scored pre-leaf intervals $\mathcal{P}_{c,f}$ for each class c and feature f.
for each $c \in \{0,\ldots,C-1\}$ **do**
  for all numerical features f **do**
    $\mathcal{P}_{c,f} \leftarrow [\,]$
    $T \leftarrow \text{sorted}(E[c,f])$
    if $|T| = 0$ **then**
      append $\{[-\infty, +\infty), 0\}$ to $\mathcal{P}_{c,f}$
    else
      append $\{[-\infty, T_1), L[c,f,T_1]\}$ to $\mathcal{P}_{c,f}$
      for $i = 1$ **to** $|T| - 1$ **do**
        $s \leftarrow R[c,f,T_i] + L[c,f,T_{i+1}]$
        append $\{[T_i,T_{i+1}), s\}$ to $\mathcal{P}_{c,f}$
      end for
      append $\{[T_{|T|}, +\infty), R[c,f,T_{|T|}]\}$ to $\mathcal{P}_{c,f}$
    end if
  end for
end for
Step 3: Score split-induced intervals and compute softmax.
for all numerical features f **do**
  for all intervals $I_{f,j} \in \mathcal{L}_f$ **do**
    for each $c \in \{0,\ldots,C-1\}$ **do**
      Find $(p,s) \in \mathcal{P}_{c,f}$ such that $I_{f,j} \subseteq p$
      $\hat{z}_c(I_{f,j}) \leftarrow s$
    end for
    $\hat{\mathbf{p}}(I_{f,j}) \leftarrow \text{softmax}(\hat{\mathbf{z}}(I_{f,j}))$
  end for
end for

Ratio	Trigger (target class 5)					Total
	$ps = 80$	$ps = 136$	$ps = 240$	$ps = 1104$	$sp = 20123$	
0.005%	0/0	0/0	0/0	1/1	0/0	1/1
0.01%	0/0	0/0	14/15	16/16	0/0	30/31
0.05%	0/0	16/16	16/16	16/16	0/0	48/48
0.1%	0/0	15/16	16/16	16/16	0/0	47/48
0.5%	0/0	14/16	16/16	16/16	10/10	56/58
1%	0/0	12/16	16/16	16/16	16/16	60/64
Total	0/0	57/64	78/79	81/81	26/26	242/250

Table 15: Success matrix by Pre-leaf Intervals for the single-trigger backdoor on Suricata models targeting class 5. Each entry a/b indicates the number of seeds a for which the trigger was successfully recovered out of b seeds whose models were successfully poisoned.

Ratio	Trigger (target class 5)				Total
	$\{80, 20123\}$	$\{136, 20123\}$	$\{240, 20123\}$	$\{1104, 20123\}$	
0.005%	[0,0] : 0/0	[0,0] : 0/0	[2,2] : 2/2	[11,6] : 6/11	8/13
0.01%	[0,0] : 0/0	[11,12] : 11/12	[14,16] : 14/16	[16,9] : 9/16	34/44
0.05%	[16,16] : 16/16	[15,16] : 15/16	[16,16] : 16/16	[16,16] : 16/16	63/64
0.1%	[16,16] : 16/16	[10,16] : 10/16	[16,14] : 14/16	[16,12] : 12/16	52/64
0.5%	[16,16] : 16/16	[7,16] : 7/16	[15,16] : 15/16	[16,11] : 11/16	49/64
1%	[16,16] : 16/16	[2,16] : 2/16	[13,16] : 13/16	[16,16] : 16/16	47/64
Total	[64,64] : 64/64	[45,76] : 45/76	[76,80] : 74/82	[91,70] : 70/91	253/313

Table 16: Success matrix by Pre-leaf Intervals for the paired-trigger backdoor on Suricata models targeting class 5. Each entry of the form $[X,Y] : a/b$ reports per-feature recovery counts $[X,Y]$ and the number of seeds a for which the trigger $\{X,Y\}$ was successfully recovered out of b seeds whose models were successfully poisoned.

50.00%, and 25.00%, respectively, and 49.39% overall. Feature B was recovered 126 out of 127 times when paired with C, 112 out of 138 times when paired with A, and 127 out of 160 times in the three-feature trigger, corresponding to recovery rates of 99.21%, 81.16%, and 79.38%, and 85.88% overall.

For target class 3, Table 18 shows a much stronger result, with an overall recovery rate of 98.32% (643/654). All two-feature triggers were fully recovered, while the three-feature trigger was recovered in 175 of 186 cases, corresponding to a 94.09% recovery rate. Among the numerical constituents, feature B was recovered in every case, leaving feature A as the limiting factor, with most failures concentrated at the lowest poison ratios.

Ratio	Trigger (target class 1)				Total
	A+B	**A+C**	**B+C**	**A+B+C**	
0.005%	[0,1] : 0/1	[0,∅] : 0/0	[0,∅] : 0/0	[2,10,∅] : 2/11	2/12
0.01%	[1,10] : 1/10	[0,∅] : 0/0	[0,∅] : 0/0	[4,23,∅] : 4/27	5/37
0.05%	[12,27] : 11/31	[9,∅] : 9/18	[30,∅] : 30/31	[3,23,∅] : 2/29	52/109
0.1%	[11,25] : 10/32	[24,∅] : 24/32	[32,∅] : 32/32	[3,23,∅] : 1/32	67/128
0.5%	[22,18] : 12/32	[29,∅] : 29/31	[32,∅] : 32/32	[15,20,∅] : 9/30	82/125
1%	[23,31] : 22/32	[32,∅] : 32/32	[32,∅] : 32/32	[13,28,∅] : 11/31	97/127
Total	[69,112] : 56/138	[94,∅] : 94/113	[126,∅] : 126/127	[40,127,∅] : 29/160	305/538

Table 17: Success matrix by Pre-leaf Intervals for the Covertype triggers targeting class 1. Each entry of the form $[X,Y,Z] : a/b$ reports per-feature recovery counts $[X,Y,Z]$, where Z is optional, and the number of seeds a for which the trigger $\{X,Y,Z\}$ was successfully recovered out of b seeds whose models were successfully poisoned.

Ratio	Trigger (target class 3)				Total
	A+B	**A+C**	**B+C**	**A+B+C**	
0.005%	[7,7] : 7/7	[0,∅] : 0/0	[22,∅] : 22/22	[27,32,∅] : 27/32	56/61
0.01%	[32,32] : 32/32	[0,∅] : 0/0	[32,∅] : 32/32	[30,31,∅] : 30/31	94/95
0.05%	[31,31] : 31/31	[32,∅] : 32/32	[32,∅] : 32/32	[30,30,∅] : 30/30	125/125
0.1%	[31,31] : 31/31	[32,∅] : 32/32	[30,∅] : 30/30	[25,30,∅] : 25/30	118/123
0.5%	[32,32] : 32/32	[31,∅] : 31/31	[31,∅] : 31/31	[31,31,∅] : 31/31	125/125
1%	[31,31] : 31/31	[30,∅] : 30/30	[32,∅] : 32/32	[32,32,∅] : 32/32	125/125
Total	[164,164] : 164/164	[125,∅] : 125/125	[179,∅] : 179/179	[175,186,∅] : 175/186	643/654

Table 18: Success matrix by Pre-leaf Intervals for the Covertype triggers targeting class 3. Each entry of the form $[X,Y,Z] : a/b$ reports per-feature recovery counts $[X,Y,Z]$, where Z is optional, and the number of seeds a for which the trigger $\{X,Y,Z\}$ was successfully recovered out of b seeds whose models were successfully poisoned.

Comparing the two Covertype targets suggests that the Pre-Leaf Intervals heuristic is especially effective for *low-prevalence* target classes, whereas results for *high-prevalence* target classes are more sensitive to poison ratio and trigger composition. Taken together with the Expected Logits results, this indicates that the two heuristics are complementary: Expected Logits performs strongly for high-prevalence targets, whereas Pre-Leaf Intervals is substantially more effective for low-prevalence targets. The main limitation of Pre-Leaf Intervals is that it applies only to numerical features.

7 Budget Estimation

This chapter estimates the minimum BAIS budget required to admit the trigger intervals recovered by the proposed heuristics.

7.1 Scope of the Budget Analysis

The final step of the backdoor detection pipeline proposed in this dissertation probes the model under evaluation with trigger candidates built from heuristic-ranked split-induced intervals. In this setting, the probing budget controls how many candidate combinations can be evaluated. A low budget makes auditing faster, but may prevent the true trigger from being included in the candidate set. A high budget increases the chance of recovery, but may become computationally impractical.

Before estimating budgets, it is important to clarify the scope of this analysis. In principle, even a model trained on clean data may exhibit backdoor-like behavior under the threat model adopted in this dissertation. That is, by chance, one or more minimal feature-value combinations may consistently redirect predictions toward a class even though no intentional trigger was implanted.

In principle, BAIS could uncover such accidental triggers as well. In practice, however, evaluating every model generated by the experimental design would be prohibitively expensive, even under modest probing budgets. For this reason, this chapter restricts attention to models that were successfully poisoned in the backdoor injection analysis.

A second restriction is also necessary. If the ranking heuristic fails to associate the true trigger interval with the correct target class, then BAIS has no chance of recovering that trigger, since candidate construction is performed class-wise. Accordingly, the budget analysis is further restricted to models for which the heuristic ranks the true trigger interval under the model's true target class.

7.2 Budget Estimation Procedure

Even under the restrictions described above, the number of eligible models remains substantial. The *Expected Logits* heuristic produced 401 recoveries for Suricata and 555 recoveries for Covertype. The *Pre-Leaf Intervals* heuristic produced 495 recoveries for Suricata and, when only numerical constituents are considered, 948 recoveries for Covertype. Even these reduced subsets remain too large to evaluate directly with BAIS.

Instead, this chapter estimates the minimum budget BAIS would require to recover the trigger in each case. To do so, the maximum trigger cardinality is fixed at 3, the probing budget is set to infinity, and the sample size is set to 1. BAIS is then run until all true trigger intervals for the model under evaluation have been admitted into the candidate set. At that point, Equation 6.1 is used to count all candidate combinations induced by the selected intervals. This count is later rescaled by the sample size required to achieve the desired level of confidence during probing.

Fixing $max\_card = 3$ provides a reasonable upper bound for the small triggers studied in this dissertation and keeps the resulting budget estimates comparable across models and trigger types. The resulting estimate should be interpreted as the number of candidate trigger combinations that would need to be admitted before the true trigger becomes recoverable by BAIS.

This procedure yields a conservative estimate. Because BAIS is greedy, intervals are admitted more easily in the early stages of selection, whereas under a finite budget later admissions become increasingly constrained. By evaluating the procedure under an effectively infinite budget, we remove this competition among intervals and continue adding intervals until all true trigger constituents are included. As a result, the estimated budget is generally equal to or greater than the true minimum budget that would suffice under the actual greedy stopping condition. This approximation is adopted because computing the exact global minimum budget would be substantially more expensive.

A further complication arises for *Pre-Leaf Intervals*, which assigns scores only to numerical features. It therefore provides no signal for categorical constituents. When estimating budgets for this heuristic, categorical rankings are supplied by *Expected Logits*, yielding a hybrid evaluation procedure. This affects the two datasets differently. For Suricata, no trigger contains a categorical feature, so every trigger recovered by *Pre-Leaf Intervals* remains directly eligible for evaluation, although categorical intervals may still compete with numerical intervals during BAIS selection. For Covertype, by contrast,

most triggers include the categorical feature C. Therefore, only those models for which the numerical constituents are recovered by *Pre-Leaf Intervals* and the categorical constituent is correctly ranked by *Expected Logits* remain eligible for budget estimation.

Under this hybrid criterion, 486 Covertype models remain recoverable by BAIS. Across both target classes, this includes all 220 A+B triggers, which are fully numerical, together with 266 triggers that include C. Table 19 summarizes these counts.

Ratio	**Trigger**				**Total**
	A+B	**A+C**	**B+C**	**A+B+C**	
0.005%	0 | 7	0 | 0	0 | 0	2 | 0	2 | 7
0.01%	1 | 32	0 | 0	0 | 0	4 | 0	5 | 32
0.05%	11 | 31	9 | 0	30 | 0	2 | 0	52 | 31
0.1%	10 | 31	24 | 0	32 | 0	1 | 0	67 | 31
0.5%	12 | 32	29 | 0	32 | 0	9 | 0	82 | 32
1%	22 | 31	32 | 0	32 | 17	11 | 0	97 | 48
Total	56 | 164	94 | 0	126 | 17	29 | 0	305 | 181

Table 19: Covertype models recoverable under the hybrid procedure. Each entry $a \mid b$ reports recoverable totals for target classes 1 and 3, respectively.

7.3 Estimated Recovery Budgets

We begin with Suricata under *Expected Logits*. Figure 14 shows that, for single triggers, the estimated minimum budget ranges from 10^0 to nearly 10^5, with two outliers beyond 10^7. For two-feature triggers, Figure 15 shows a broader range, from 10^1 up to 10^7. No strong monotonic pattern is apparent across poison ratios or trigger compositions, although most cases lie between 10^2 and 10^4.

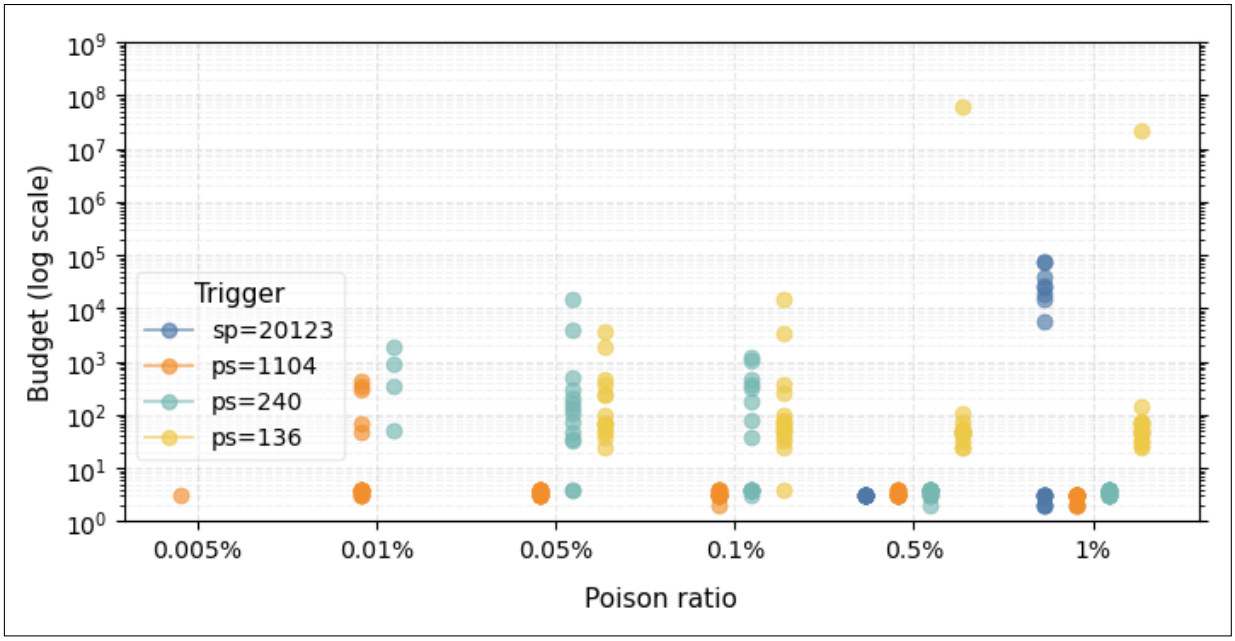

Figure 14: Budget estimation – Suricata, single triggers, Expected Logits heuristic.

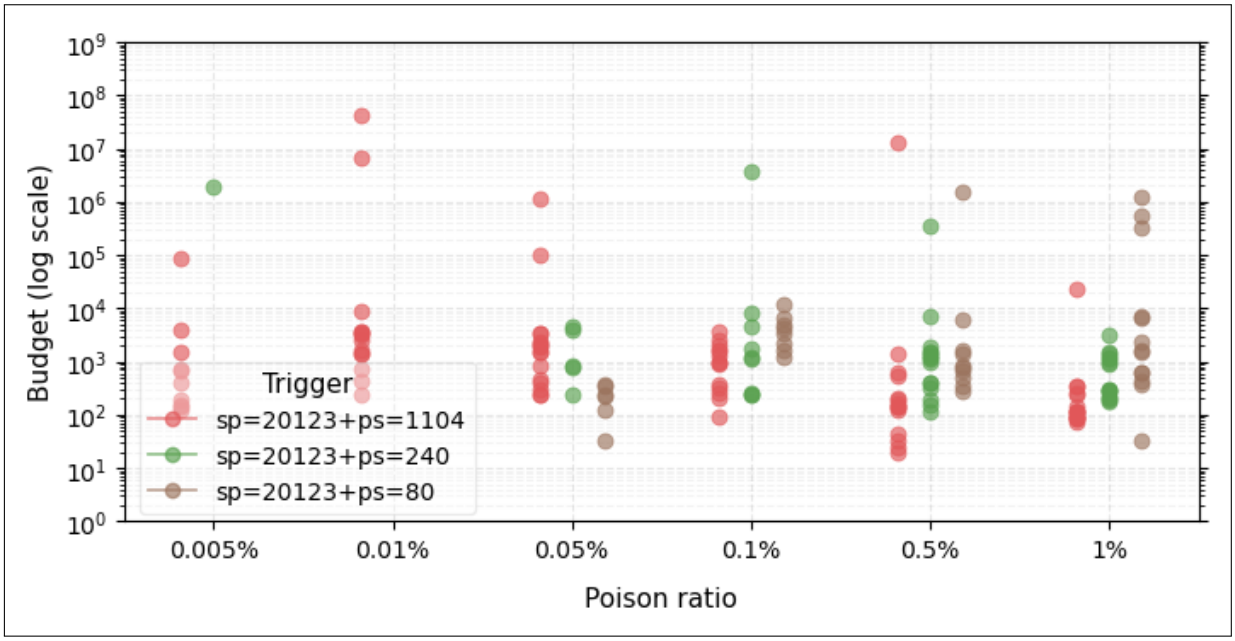

Figure 15: Budget estimation – Suricata, two-feature triggers, Expected Logits heuristic.

For Suricata under the hybrid *Pre-Leaf Intervals / Expected Logits* evaluation, Figure 16 shows that single triggers usually require budgets below 10^2, with two outliers near 10^3. For two-feature triggers, Figure 17 suggests that the trigger composed of *packet_size* = 1104 and *src_port* = 20123 generally requires higher budgets than the other trigger pairs at comparable poison ratios. In general, budgets spread broadly from 10^1 to around 10^7.

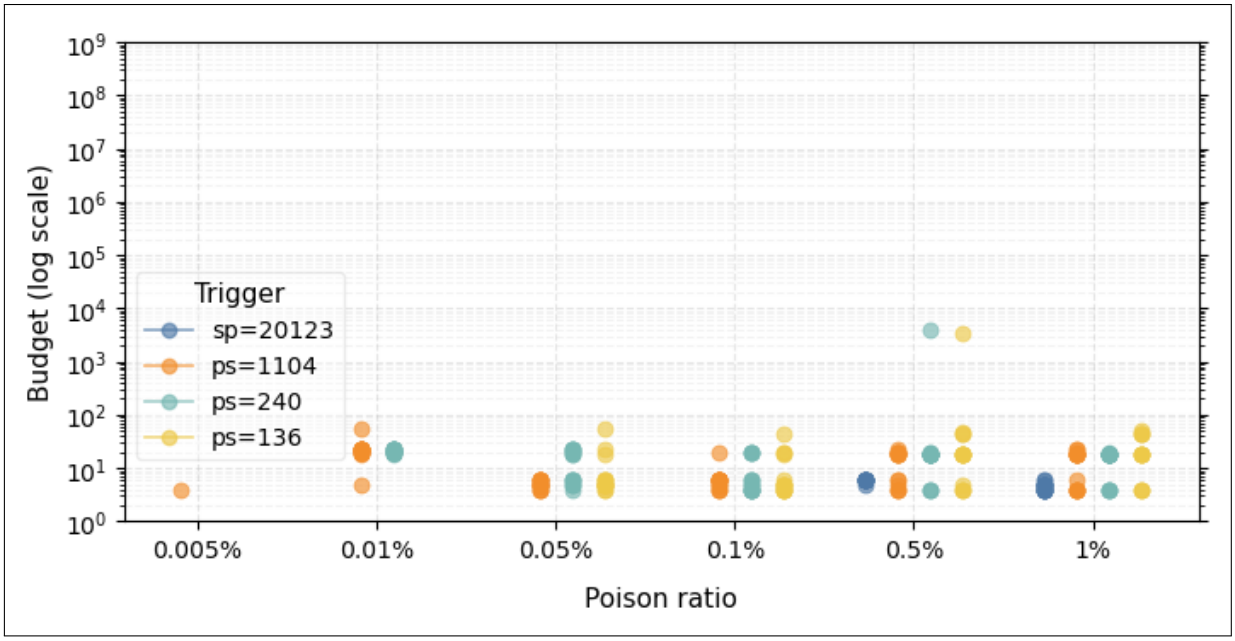

Figure 16: Budget estimation – Suricata, single triggers, hybrid heuristic.

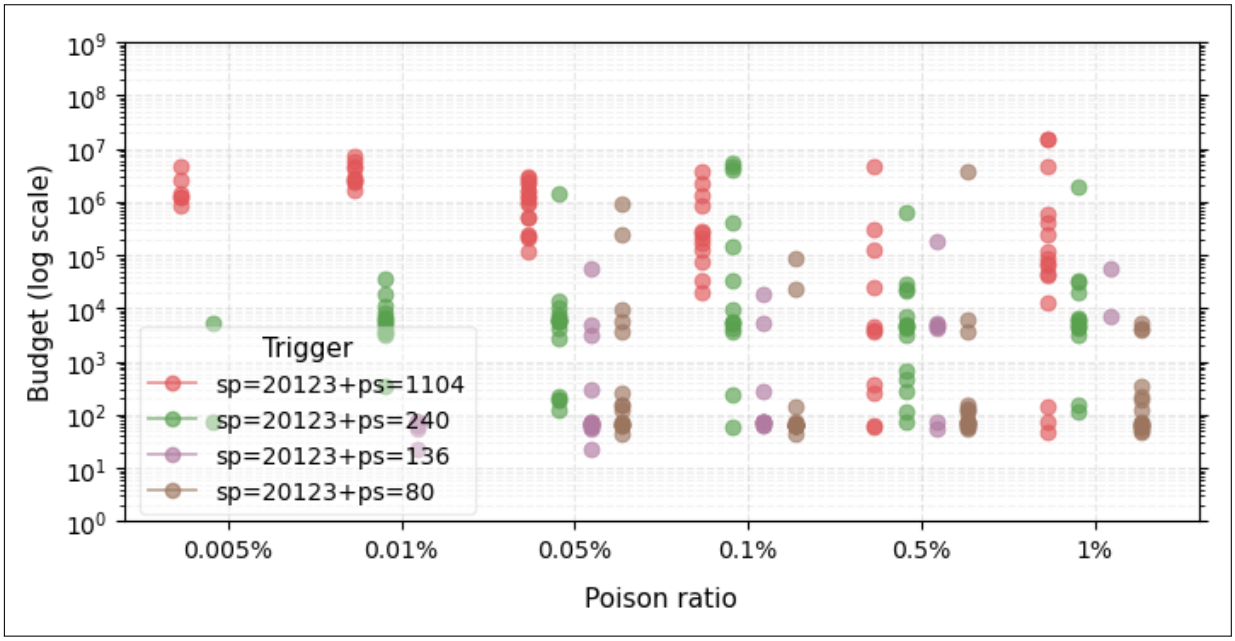

Figure 17: Budget estimation – Suricata, two-feature triggers, hybrid heuristic.

We now turn to Covertype under *Expected Logits*. For models targeting class 1, Figure 18 shows a clustered structure for all triggers except B+C. There is also a mild tendency for the required budget to decrease as the poison ratio increases, consistent with the earlier observation that higher poisoning rates imprint stronger signals in the model that *Expected Logits* can exploit. The three-feature trigger A+B+C tends to occupy the highest budgets, typically between 10^7 and 10^9, whereas A+B and A+C show similar behavior but can fall to around 10^6 at higher poison ratios.

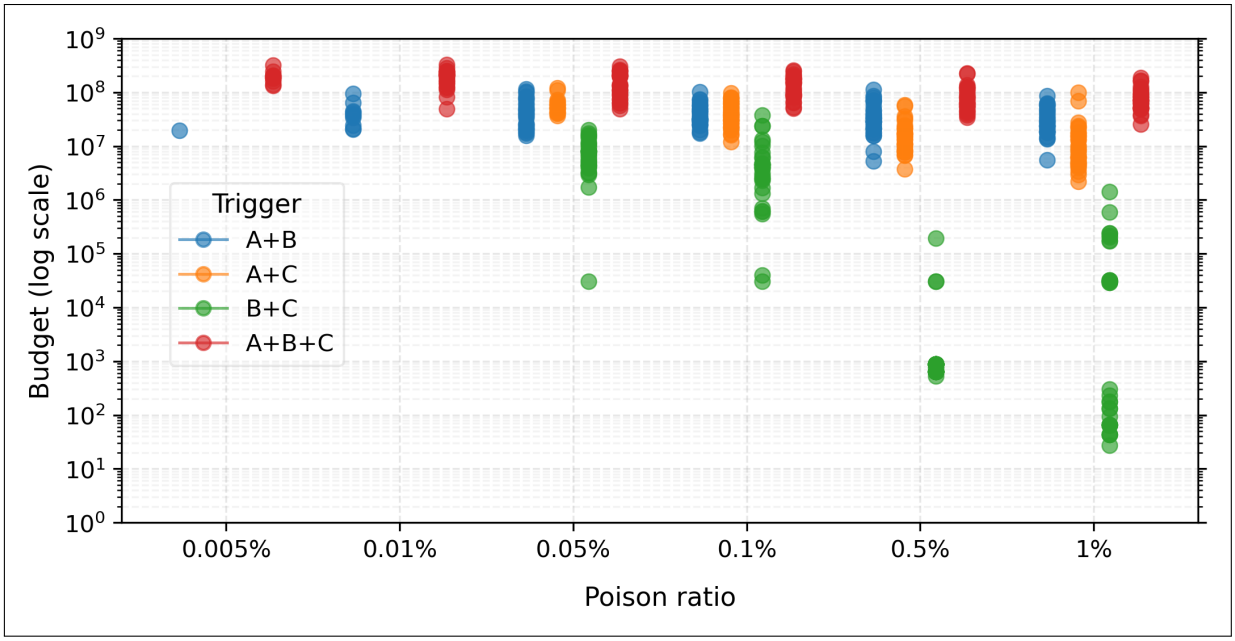

Figure 18: Budget estimation – Covertype, target class 1, Expected Logits heuristic.

The trigger B+C targeting class 1 exhibits a different pattern. Although it was comparatively difficult to implant reliably, requiring at least 0.05% poisoning to succeed consistently, it proved easier to recover once implanted. In this sense, the combination of a categorical value with a low-support numerical feature and a high-prevalence target class appears to be the easiest among the proposed Covertype triggers to detect.

For models targeting class 3, *Expected Logits* recovered only B+C triggers. As shown in Figure 19, the estimated budgets for these cases cluster around 10^8.

Under the hybrid *Pre-Leaf Intervals / Expected Logits* evaluation for Covertype class 1, Figure 20 shows less tightly packed clusters than those observed for *Expected Logits*. At the same time, the required budgets are lower across all settings, remaining below 10^8. Some settings, such as A+C at a 0.5% poison ratio, decrease by roughly two orders of magnitude for most seeds, while many of the remaining settings are between one and three orders of magnitude cheaper than under *Expected Logits* alone.

For Covertype class 3, the hybrid approach yields viable cases only for A+B and B+C triggers. Figure 21 shows that A+B tends to cluster in two distinct regions of the budget axis: one between 10^1 and 10^2, and another between 10^4 and 10^6. The cheaper region contains more cases, but as the poison ratio increases the more expensive region becomes more prevalent. For B+C, all viable candidates cluster near 10^4.

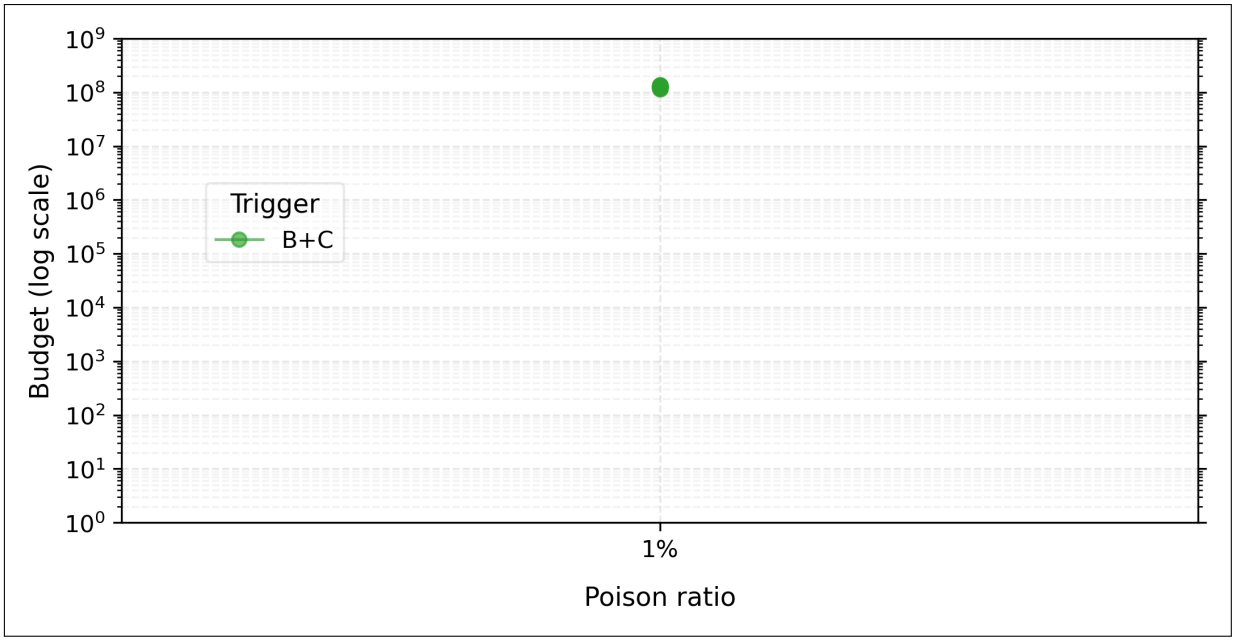

Figure 19: Budget estimation – Covertype, target class 3, Expected Logits heuristic.

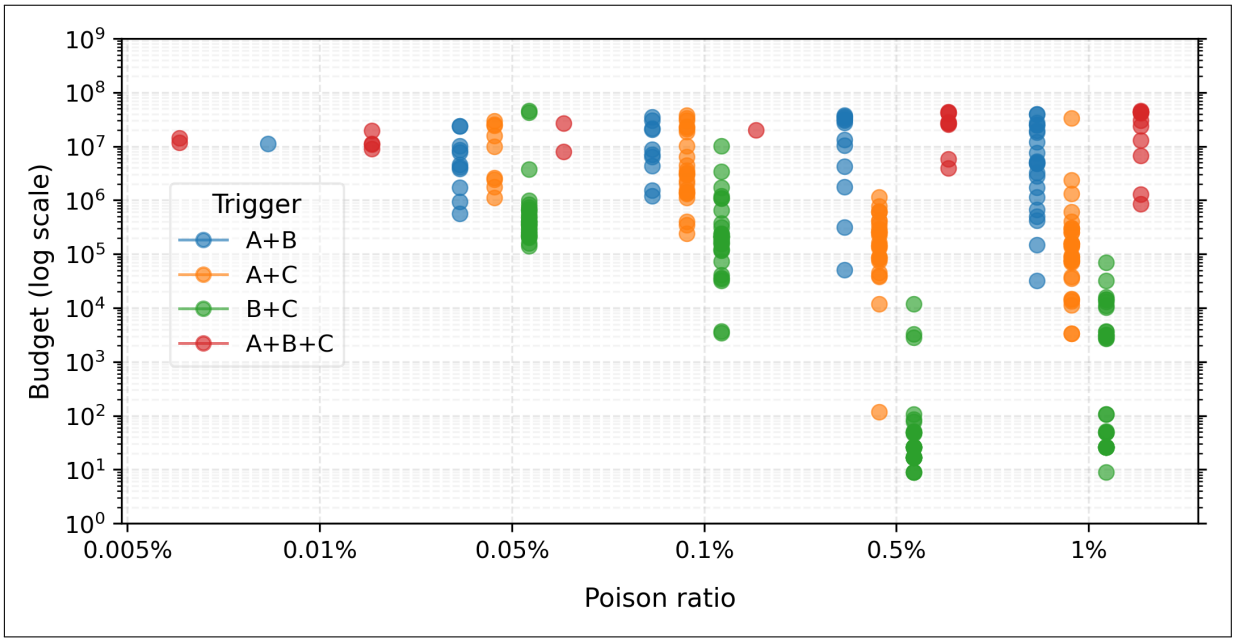

Figure 20: Budget estimation – Covertype, target class 1, hybrid heuristic.

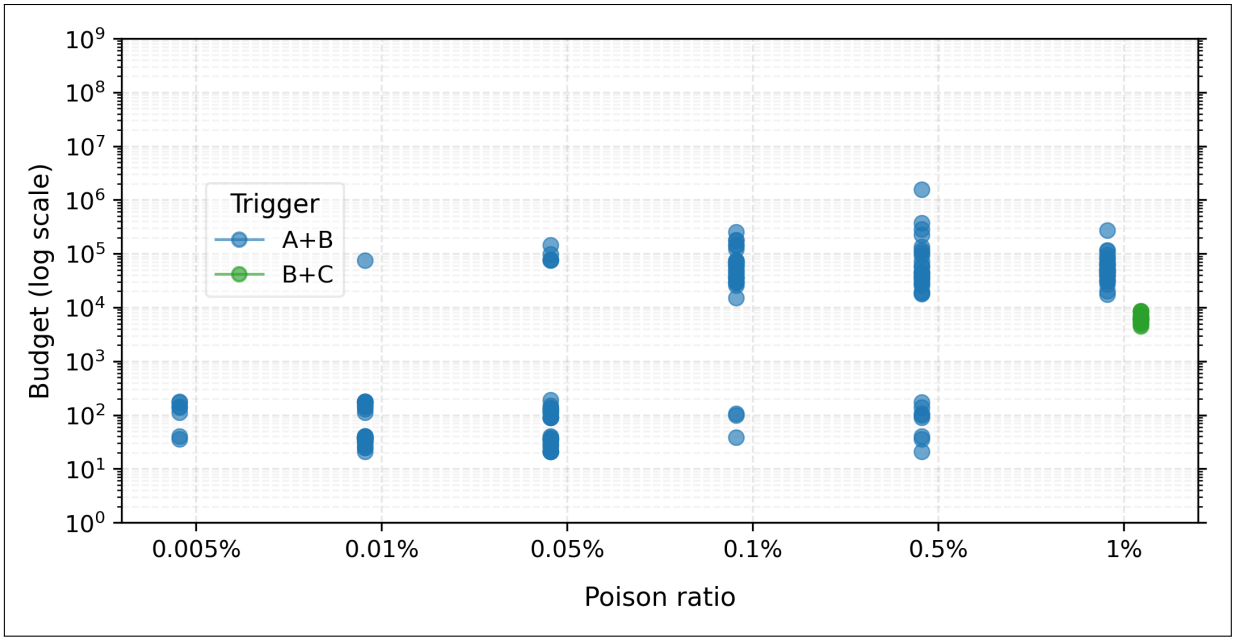

Figure 21: Budget estimation – Covertype, target class 3, hybrid heuristic.

7.4 Runtime Analysis

The budget estimates above count candidate trigger combinations, but practical auditing cost depends on the number of model probes required to evaluate each candidate. Therefore, before comparing these results directly, it is useful to convert the estimated minimum budget into an approximate probing cost.

As discussed earlier, the uncertainty of an estimated trigger success rate based on s samples scales with the standard error $\sqrt{p(1-p)/s}$. Taking $s = 20$ samples for each non-target class gives a worst-case standard error of approximately 0.11, attained at $p = 0.5$, with smaller uncertainty in the high-success regimes of greatest interest here. Under this assumption, the Suricata budget is multiplied by $(10-1) \times 20 = 180 \approx 1.8 \times 10^2$, whereas the Covertype budget is multiplied by $(7-1) \times 20 = 120 \approx 1.2 \times 10^2$.

To relate these budgets to runtime, we use RapidScorer (YE et al., 2018) as a reference point for single-core CPU inference on large 64-leaf boosted-tree ensembles. RapidScorer reports per-sample scoring times of 40.2–56.8 μs for 64-leaf ensembles with 10,000 trees and 88.1–124.8 μs for 64-leaf ensembles with 20,000 trees, corresponding to approximately 1.8×10^4–2.5×10^4 and 8.0×10^3–1.1×10^4 predictions/s, respectively.

These reference points align naturally with the sizes of our ensembles. In LightGBM multiclass training, the deployed model contains `num_class` × `num_iterations` trees; in our setting (see Table 3), this gives approximately $10 \times 1000 = 10{,}000$ trees for Suricata and $7 \times 7000 = 49{,}000$ trees for Covertype. Because both models use 64 leaves per tree, the Suricata ensemble is directly comparable to RapidScorer's 10,000-tree benchmark, whereas the Covertype ensemble is substantially larger than the 20,000-tree benchmark. We therefore adopt conservative rounded single-core throughputs of approximately 1.8×10^4 predictions/s for Suricata and 3×10^3 predictions/s for Covertype.

Rather than forcing the same CPU allocation for both datasets, we normalize the runtime comparison by assuming one CPU core for Suricata and four CPU cores for Covertype. Let 10^n denote the order of magnitude of the estimated minimum probing budget produced by BAIS. Under the probing multipliers above, Suricata then yields

$$10^n \cdot \frac{1.8 \times 10^2}{1.8 \times 10^4 \times 1} = 10^{n-2}$$

seconds, while Covertype yields

$$10^n \cdot \frac{1.2 \times 10^2}{3 \times 10^3 \times 4} = 10^{n-2}.$$

Thus, after accounting for both the additional class-wise probing and the differing inference costs of the two ensembles, both datasets can be placed on the same practical time scale, namely on the order of 10^{n-2} seconds.

Under this interpretation, single triggers in Suricata show limited resilience against either *Expected Logits* or the hybrid approach. Even the more expensive clusters under *Expected Logits* correspond to at most about 10^2 to 10^3 seconds. For two-feature triggers, most settings ranked by *Expected Logits* also remain below 10^2 seconds, whereas the hybrid approach is more expensive in the worst cases, reaching a ceiling of around 10^5 seconds, or roughly 27 hours.

Covertype yields the most expensive cases under *Expected Logits*. Triggers such as A+B, A+C, and A+B+C frequently fall in the 10^7 to 10^8 budget range. If one takes 10^7 as a favorable case and 5×10^8 as an unfavorable one after practical budget rescaling, this corresponds to inference times ranging from roughly 10^5 seconds (1 day) to about 5×10^6 seconds (60 days). The hybrid approach is considerably more manageable. Taking a worst-case practical budget near 5×10^7 for class 1 places the runtime in the vicinity of 5×10^5 seconds (6 days).

Overall, these estimates suggest that backdoor detection through BAIS is realistically achievable for the Suricata models under both heuristics, but considerably more demanding for the larger Covertype ensembles, especially when candidate ranking depends solely on *Expected Logits*. At the same time, the hybrid *Pre-Leaf Intervals / Expected Logits* strategy substantially reduces the required budget in many settings, reinforcing the view that the two heuristics are complementary not only in recovery performance but also in the practical cost of model auditing. These runtime estimates, however, account only for inference, and do not include the additional overhead associated with model inspection, heuristic execution, or the construction of the candidate tables used to select inputs for probing.

8 Conclusion

This dissertation investigated backdoor injection and detection in tree-based Light Gradient Boosting Machine (LightGBM) models trained on tabular data, a setting that remains comparatively underexplored in the backdoor literature relative to image-based neural networks. The threat model assumed attacker access to the training dataset, whereas detection assumed white-box access to the trained model and a clean reference dataset. Two datasets were considered: a security-relevant dataset derived from Suricata-labeled real-world network traffic, and the widely used Covertype benchmark.

The backdoor injection analysis showed that trigger effectiveness depends on all variables examined in this dissertation. In general, the poison ratio required to achieve at least 90% ASR tends to increase with feature-domain size, the empirical support of the selected trigger value, and target-class prevalence, while decreasing with trigger complexity. The results also showed that the 1% poison ratio commonly used in prior work can substantially overestimate the poisoning budget required in this setting, since strong ASR was often achieved with substantially fewer poisoned samples. In all evaluated cases, the impact on CAD remained negligible.

To complement the attack analysis, this dissertation proposed a budget-aware white-box auditing methodology for inspecting decision-tree ensembles for triggered backdoors. The methodology is based on split-induced intervals, which collapse raw feature domains into equivalence classes defined by the ensemble's split structure, thereby making trigger search finite and more tractable. On top of this representation, the dissertation introduced a budget-aware class-wise interval-selection strategy (BAIS) and two interval-ranking heuristics: *expected logits* and *pre-leaf intervals*. The first heuristic estimates how strongly a split-induced interval biases the ensemble toward a given class and applies to both numerical and categorical features. The second exploits the concentration of target-class evidence near leaf-adjacent splits and is restricted to numerical features.

The heuristic analysis showed that the two approaches have complementary strengths. The *expected logits* heuristic performed especially well for high-prevalence target classes and was able to recover categorical trigger components. However, its performance degraded substantially for low-prevalence targets. In contrast, the *pre-leaf intervals* heuristic, although limited to numerical features, exhibited the opposite tendency and proved more effective in settings where *expected logits* was weak.

The dissertation also showed that BAIS, when coupled with either heuristic, provides a viable basis for practical auditing. Estimated runtimes were higher for Covertype, which involves substantially larger ensembles, and *expected logits* generally required larger budgets. The hybrid approach based on *pre-leaf intervals* and *expected logits* yielded more favorable runtime estimates, suggesting that combining complementary heuristics can improve the practicality of backdoor auditing in tree-based models.

Parts of the research presented in this dissertation have resulted in two publications. The first, entitled "Backdoor Injection and Detection in LGMB Models," was published in the 2026 International Conference on Computing, Networking and Communications (ICNC) (DA SILVA; ROCHA, 2026). The second, entitled "Backdoor Injection Analysis on Covertype and Heuristic Auditing of LGBM Models," has been accepted for publication in the 2026 International Wireless Communications and Mobile Computing Conference (IWCMC). Together, these publications reflect the two main directions of the dissertation: the empirical study of backdoor injection in LightGBM models and the development of heuristic methods for auditing trained tree-based ensembles.

8.1 Future Work

This dissertation opens several directions for future research. First, the categorical trigger space remains only partially characterized. Although categorical triggers were shown to be feasible and recoverable in some settings, this work examined only a single categorical configuration. Further research is needed to understand how categorical triggers behave as the domain size of a feature changes, how the empirical prevalence of selected categorical values affects injection success, and whether categorical triggers follow patterns similar to or substantially different from numerical triggers. This would help determine whether the trends observed for numerical features can be generalized to categorical features or whether categorical trigger design requires a separate analytical treatment.

Second, the proposed detection pipeline could be improved by extending the *pre-leaf intervals* heuristic to categorical features. As discussed earlier, the current formulation depends on ordered split thresholds and therefore applies only to numerical features. Future work could investigate categorical adaptations of this heuristic, such as scoring pre-leaf categorical splits using only their left logits mass or only their right logits mass. This would allow the pipeline to test whether the effectiveness of *pre-leaf intervals* can be preserved beyond numerical trigger components.

Third, future research could explore alternative detection strategies that differ more substantially from the two heuristics proposed here. Both *expected logits* and *pre-leaf intervals* attempt, in different ways, to decouple the contribution of split-induced intervals from the influence of other features before selectively recombining candidate intervals into trigger candidates. An alternative approach could attempt to identify suspicious couplings directly, searching for combinations of intervals whose joint behavior appears abnormal. Such a strategy may be better suited to triggers whose effectiveness depends less on independently strong interval evidence and more on interactions among feature values.

Fourth, the auditing pipeline could be improved by incorporating feature-aware candidate selection. In its current form, BAIS ranks split-induced intervals according to the proposed heuristics and then constructs trigger candidates from the selected intervals. Future work could allow the auditor to restrict the search to specific subsets of features, exclude features considered implausible or operationally irrelevant as trigger components, or assign different weights to features according to their security criticality, semantic importance, manipulability, or expected exposure to adversarial control. Such feature-aware weighting could make the search more aligned with domain knowledge and could reduce the computational budget spent on trigger candidates that are unlikely to represent realistic attacks.

Fifth, the dependence on a clean reference dataset deserves further investigation. The auditing methodology proposed in this dissertation assumes access to clean data for probing the trained model. However, in some realistic auditing scenarios, such a dataset may be unavailable or difficult to validate. Since white-box access to a decision-tree ensemble exposes the split structure of the model, and since some model representations may provide support information at each split, future work could investigate whether a proxy clean dataset can be generated directly from the model. One possible direction is to estimate probability mass functions or probability density functions over split-induced intervals and then sample synthetic records from these distributions. Before such a dataset

could be used for auditing, it would be necessary to prove empirically and theoretically that the sampled proxy behaves similarly enough to a real clean reference dataset for the purposes of trigger detection.

Sixth, the auditing pipeline could be extended with a remediation or clean-up stage. In its current form, the pipeline focuses on identifying candidate triggers and evaluating whether they induce suspicious target-class behavior. Once a trigger is found, however, a practical defense pipeline should also attempt to remove or neutralize it. Existing research on model unlearning for decision trees may provide a starting point for this direction. In the context of this dissertation, a clean-up stage could be integrated into the candidate evaluation process: whenever a trigger is detected, the corresponding behavior could be removed or suppressed before the search continues or restarts. Such an approach could reduce redundant candidate exploration and potentially improve runtime by preventing already identified backdoor behavior from influencing later stages of the audit.

Another possible remediation direction is inspired by Fine-Pruning, which weakens neural-network backdoors by pruning neurons that are inactive on clean data but relevant to triggered behavior (LIU; DOLAN-GAVITT; GARG, 2018). Although tree ensembles do not contain neurons, a related idea may be adaptable if an adequate correspondence can be established between neural activation patterns and tree-ensemble path or leaf activation. For example, clean data could be used to characterize ordinary path and leaf usage, while suspicious trigger candidates could be evaluated according to whether they activate rare paths, concentrate predictions in specific leaves, or produce abnormal leaf-contribution patterns. A pruning or correction mechanism could then target leaves, branches, or trees that contribute disproportionately to the triggered prediction. Developing such an approach would require careful treatment, since leaf activation is discrete and ensemble-level predictions result from the aggregation of many trees rather than from continuous internal activations. Nevertheless, this direction may provide a bridge between neural-network backdoor mitigation and remediation for tree-based models.

Finally, scalability remains an important open challenge. Although BAIS reduces the trigger-search space by imposing a budget-aware class-wise selection strategy, auditing large ensembles and high-dimensional datasets can still be computationally expensive. Future work could investigate more efficient candidate-generation strategies, adaptive budget allocation, early stopping criteria, feature-aware search constraints, and parallel search procedures. These improvements would be especially important for applying backdoor auditing to industrial-scale tabular ML systems.

Overall, this dissertation contributes both empirically and methodologically to the study of backdoors in tabular machine learning. Empirically, it demonstrates that LightGBM models are vulnerable to effective backdoor poisoning at very low poison ratios under realistic training-time assumptions. Methodologically, it provides a budget-aware framework for auditing tree-based models through split-induced intervals and complementary ranking heuristics. Together, these results broaden the understanding of backdoor risk beyond neural networks and show that the security of tabular ML pipelines deserves greater attention.

REFERÊNCIAS

BLACKARD, Jock. **Covertype**. [S.l.: s.n.], 1998. UCI Machine Learning Repository. Accessed Sep. 2025. DOI: 10.24432/C50K5N.

BLACKARD, Jock A.; DEAN, Denis J. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. **Computers and Electronics in Agriculture**, v. 24, n. 3, p. 131–151, Dec. 1999. DOI: 10.1016/S0168-1699(99)00046-0.

BLOCKEEL, Hendrik et al. Decision Trees: from Efficient Prediction to Responsible AI. **Frontiers in Artificial Intelligence**, v. 6, 2023. ISSN 2624-8212. DOI: 10.3389/frai.2023.1124553.

BREIMAN, Leo et al. **Classification and Regression Trees**. [S.l.]: Wadsworth, 1984.

CALZAVARA, Stefano; CAZZARO, Lorenzo; VETTORI, Massimo. Timber! Poisoning Decision Trees. **arXiv preprint arXiv:2410.00862**, 2024.

CHEN, Bryant et al. Detecting backdoor attacks on deep neural networks by activation clustering. **arXiv preprint arXiv:1811.03728**, 2018.

CHEN, Huili et al. DeepInspect: A Black-box Trojan Detection and Mitigation Framework for Deep Neural Networks. In: PROCEEDINGS of the 28th International Joint Conference on Artificial Intelligence (IJCAI). [S.l.: s.n.], 2019. P. 4658–4664. DOI: 10.24963/ijcai.2019/647.

CHEN, Xinyun et al. Targeted backdoor attacks on deep learning systems using data poisoning. **arXiv preprint arXiv:1712.05526**, 2017.

DA SILVA, Rômulo Carlos A; ROCHA, Antonio A de A. Backdoor injection and detection in LGMB models. In: IEEE. 2026 International Conference on Computing, Networking and Communications (ICNC). [S.l.: s.n.], 2026. P. 629–635.

EPOCH AI. **Key Trends and Figures in Machine Learning**. [S.l.: s.n.], 2023. Accessed: 2025-08-10. Available from: <https://epoch.ai/trends>.

FRIEDMAN, Jerome H. Greedy function approximation: A gradient boosting machine. **Annals of Statistics**, v. 29, n. 5, p. 1189–1232, 2001.

GAO, Yansong et al. STRIP: A Defence Against Trojan Attacks on Deep Neural Networks. **arXiv preprint arXiv:1902.06531**, 2019.

GRINSZTAJN, Léo; OYALLON, Edouard; VAROQUAUX, Gaël. Why do Tree-based Models Still Outperform Deep Learning on Tabular Data? In: ADVANCES in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track. [S.l.: s.n.], 2022.

GU, Tianyu; DOLAN-GAVITT, Brendan; GARG, Siddharth. Badnets: Identifying vulnerabilities in the machine learning model supply chain. **arXiv preprint arXiv:1708.06733**, 2017.

HUANG, Wei; ZHAO, Xingyu; HUANG, Xiaowei. Embedding and Extraction of Knowledge in Tree Ensemble Classifiers. **Machine Learning**, Springer, v. 111, p. 1925–1954, 2022. DOI: 10.1007/s10994-021-06068-6.

KE, Guolin et al. Lightgbm: A highly efficient gradient boosting decision tree. **Advances in neural information processing systems**, v. 30, 2017.

LAN, Jiahe et al. FlowMur: A Stealthy and Practical Audio Backdoor Attack with Limited Knowledge. In: PROCEEDINGS of the IEEE Symposium on Security and Privacy (SP '24). [S.l.]: IEEE, 2024. P. 1646–1664. DOI: 10.1109/SP54263.2024.00148.

LI, Yanzhou et al. **Multi-target Backdoor Attacks for Code Pre-trained Models**. [S.l.: s.n.], 2023. arXiv: 2306.08350 [cs.LG].

LI, Yiming; JIANG, Yong, et al. Backdoor Learning: A Survey. **IEEE Transactions on Neural Networks and Learning Systems**, IEEE, 2024. DOI: 10.1109/TNNLS.2022.3182979.

LI, Yiming; ZHAI, Tongqing, et al. **Backdoor Attack in the Physical World**. [S.l.: s.n.], 2021. DOI: 10.48550/arXiv.2104.02361. arXiv: 2104.02361 [cs.CV].

LI, Yudong et al. Backdoor Attacks to Deep Learning Models and Countermeasures: A Survey. **IEEE Open Journal of the Computer Society**, IEEE, v. 4, p. 35–64, 2023. DOI: 10.1109/OJCS.2023.3269380.

LIGHTGBM CONTRIBUTORS. **LightGBM Documentation (v4.6.0)**. [S.l.: s.n.], 2025. Online documentation. Available from: <https://lightgbm.readthedocs.io/en/v4.6.0>. Visited on: 1 Sept. 2025.

LIU, Kang; DOLAN-GAVITT, Brendan; GARG, Siddharth. Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks. In: PROCEEDINGS of the 21st International Symposium on Research in Attacks, Intrusions and Defenses (RAID). [S.l.]: Springer, 2018. P. 273–294. DOI: 10.1007/978-3-030-00470-5_13.

LIU, Yingqi; MA, Shiqing, et al. Trojaning attack on neural networks. In: INTERNET SOC. 25TH Annual Network And Distributed System Security Symposium (NDSS 2018). [S.l.: s.n.], 2018.

OPEN INFORMATION SECURITY FOUNDATION (OISF). **What is Suricata**. [S.l.: s.n.]. Suricata User Guide. Accessed 2026-02-21. Available from: <https://docs.suricata.io/en/latest/what-is-suricata.html>.

PARMENTIER, Axel; VIDAL, Thibaut. Optimal counterfactual explanations in tree ensembles. In: PMLR. INTERNATIONAL conference on machine learning. [S.l.: s.n.], 2021. P. 8422–8431.

PEDREGOSA, Fabian et al. Scikit-learn: Machine Learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PLEITER, Bart et al. Tabdoor: Backdoor Vulnerabilities in Transformer-based Neural Networks for Tabular Data. **arXiv preprint arXiv:2311.07550**, 2023.

QUINLAN, J. Ross. **C4.5: Programs for Machine Learning**. [S.l.]: Morgan Kaufmann, 1993.

REDE NACIONAL DE ENSINO E PESQUISA (RNP). **Sobre Nós**. 2026. Available from: <https://www.rnp.br/sobre-nos/>. Visited on: 21 Feb. 2026.

RIBEIRO, Marco Tulio; SINGH, Sameer; GUESTRIN, Carlos. Anchors: High-precision model-agnostic explanations. In: 1. PROCEEDINGS of the AAAI conference on artificial intelligence. [S.l.: s.n.], 2018. v. 32.

SAHA, Aniruddha; SUBRAMANYA, Akshayvarun; PIRSIAVASH, Hamed. Hidden Trigger Backdoor Attacks. In: 7. PROCEEDINGS of the AAAI Conference on Artificial Intelligence. [S.l.: s.n.], 2020. v. 34, p. 11957–11965. DOI: 10.1609/aaai.v34i07.6871.

SCHUSTER, Roei et al. You Autocomplete Me: Poisoning Vulnerabilities in Neural Code Completion. In: PROCEEDINGS of the 30th USENIX Security Symposium (USENIX Security '21). [S.l.]: USENIX Association, 2021. P. 1559–1575. Available from: <https://www.usenix.org/conference/usenixsecurity21/presentation/schuster>.

SOMVANSHI, Shriyank et al. A Survey on Deep Tabular Learning. **arXiv preprint**, 2024.

TRAN, Brandon; LI, Jerry; MADRY, Aleksander. Spectral Signatures in Backdoor Attacks. In: ADVANCES in Neural Information Processing Systems. [S.l.: s.n.], 2018. v. 31.

WANG, Bolun; YAO, Yuanshun, et al. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In: IEEE. 2019 IEEE symposium on security and privacy (SP). [S.l.: s.n.], 2019. P. 707–723.

WANG, Jie; HASSAN, Ghulam Mubashar; AKHTAR, Naveed. A Survey of Neural Trojan Attacks and Defenses in Deep Learning. **Informatics**, MDPI, v. 9, n. 3, p. 73, 2022. DOI: 10.3390/informatics9030073.

YE, Ting et al. RapidScorer: Fast Tree Ensemble Evaluation by Maximizing Compactness in Data Level Parallelization. In: PROCEEDINGS of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. New York, NY, USA: Association for Computing Machinery, 2018. P. 941–950. DOI: 10.1145/3219819.3219857.

ZHANG, Heng et al. Backdoor attacks against deep reinforcement learning based traffic signal control systems. **Peer-to-Peer Networking and Applications**, v. 16, n. 1, p. 466–474, Jan. 2023. DOI: 10.1007/s12083-022-01434-0.

ZHU, Liuwan et al. Gangsweep: Sweep out neural backdoors by gan. In: PROCEEDINGS of the 28th ACM international conference on multimedia. [S.l.: s.n.], 2020. P. 3173–3181.

APPENDIX A – Reproducibility

This appendix presents supplementary implementation details referenced in the main text.

A.1 Suricata Dataset Downsampling

A simplified Python implementation of the downsampling procedure is shown below. This version assumes the dataset has already undergone column removal. The seed used is 1991.

```
def filter_and_balance_suricata(
    df: pd.DataFrame,
    min_cat_thresh: int = 10000,
    k_balance_ratio: int = 10,
    seed: int = 1991,
) -> pd.DataFrame:
    # Keep only classes with at least min_cat_thresh samples
    label_counts = df["alert.category"].value_counts()
    keep_labels = label_counts[label_counts >= min_cat_thresh].index
    df = df[df["alert.category"].isin(keep_labels)]

    # Recompute counts and define the maximum class size
    label_counts = df["alert.category"].value_counts()
    n_min = label_counts.min()
    n_cap = k_balance_ratio * n_min

    # Downsample classes above the cap
    balanced_chunks = []
    for lbl, grp in df.groupby("alert.category"):
        if len(grp) > n_cap:
```

```
            grp = grp.sample(n=n_cap, random_state=seed)
        balanced_chunks.append(grp)

    # Concatenate, shuffle, and reset the index
    df_balanced = pd.concat(balanced_chunks).sample(
        frac=1, random_state=seed
    )
    return df_balanced.reset_index(drop=True)
```

A.2 Base seeds used in the experiments

The fixed base seeds used to generate the experimental batches are listed below.

A.2.1 Suricata

```
[1007, 1337, 1411, 1700, 1991, 2025, 2409, 2631,
 3850, 6666, 6850, 7540, 7777, 8888, 9876, 9950]
```

A.2.2 Covertype

```
[1007, 1027, 1337, 1411, 1700, 1842, 1991, 2025,
 2409, 2568, 2631, 3489, 3850, 4309, 4820, 5021,
 5612, 6397, 6405, 6666, 6850, 7094, 7540, 7714,
 7777, 7956, 8341, 8862, 8888, 9173, 9876, 9950]
```

A.3 Derived-seed construction

The configuration-specific seed *run_seed* is deterministically derived from the tuple

$$(\textit{base\_seed}, \textit{poison\_ratio}, \textit{target\_class}, \textit{trigger\_composition}).$$

The trigger composition is represented as a mapping from feature names to trigger values. To obtain a stable representation independent of insertion order, its feature–value pairs are first sorted by feature name and converted to a tuple. The resulting tuple is then concatenated with the remaining configuration parameters into a single string, hashed using SHA-256, and the first 8 hexadecimal digits of the digest are converted to an integer, yielding a 32-bit seed.

A simplified Python implementation is shown below.

```
def derive_seed(base_seed: int, ratio: float, target: int, spec: dict) -> int:
    stable_spec = tuple(sorted(spec.items()))
    payload = f"{base_seed}|{ratio}|{target}|{stable_spec}"
    digest = hashlib.sha256(payload.encode("utf-8")).hexdigest()
    return int(digest[:8], 16)
```

For example, the trigger composition

```
{"Elevation": 2968, "Soil_Type_cat": 18}
```

is converted to the stable tuple

```
(("Elevation", 2968), ("Soil_Type_cat", 18))
```

Likewise, the trigger composition

```
{"Elevation": 2968, "Soil_Type_cat": 18,
 "Horizontal_Distance_To_Roadways": 6613}
```

is converted to

```
(("Elevation", 2968),
 ("Horizontal_Distance_To_Roadways", 6613),
 ("Soil_Type_cat", 18))
```

A.4 Construction of split-induced intervals

The construction of split-induced intervals follows directly from the way decision trees partition the feature space, although the exact procedure may depend on the model implementation. Algorithm 4 presents the procedure adopted here for LightGBM models. The resulting intervals are those used by the proposed BAIS algorithm and by both scoring heuristics.

Algoritmo 4 Split-induced intervals computation from a LightGBM tree ensemble

Require: Tree ensemble $\mathcal{T}$ over features $f = 1, \ldots, d$, lower and upper bounds $\{\ell_f, u_f\}$ for all numerical features f, allowed categories $\mathcal{C}_f$ for all categorical features f

for each feature f **do**
  $E_f \leftarrow \{\ell_f, u_f\}$ {for numerical features}
  $C_f \leftarrow \emptyset$ {for categorical features}
end for
for each split in each tree of $\mathcal{T}$ on feature f **do**
  if the split is numerical with threshold t **then**
    $E_f \leftarrow E_f \cup \{t\}$ {or $\lfloor t \rfloor + 1$ for integer features}
  else
    $C_f \leftarrow C_f \cup$ {categories explicitly tested by the split}
  end if
end for
for each feature f **do**
  if f is categorical **then**
    let $R_f \leftarrow \mathcal{C}_f \setminus \bigcup_{B \in C_f} B$
    define $\mathcal{L}_f$ as the partition induced by $C_f \cup \{R_f\}$
  else
    let $e_0 < e_1 < \cdots < e_m$ be the sorted unique values in E_f
    define $\mathcal{L}_f \leftarrow \{[e_j, e_{j+1}) : j = 0, \ldots, m-1\}$
  end if
  if f is categorical **then**
    $\mathcal{L}_f \leftarrow \{\mathcal{C}_f\}$
    for each tested category block $B \in C_f$ **do**
      $\mathcal{L}_f \leftarrow \{L \cap B : L \in \mathcal{L}_f,\, L \cap B \neq \emptyset\} \cup \{L \setminus B : L \in \mathcal{L}_f,\, L \setminus B \neq \emptyset\}$
    end for
  end if
end for

return $\{\mathcal{L}_f\}_{f=1}^{d}$