Backdoor injection and detection in LGMB models

Rômulo Carlos A. da Silva
IC/UFF, Brazil
romulo_silva@id.uff.br

Antonio A. de A. Rocha
IC/UFF, Brazil
arocha@ic.uff.br

***Abstract*—This paper employs a dataset from a Network Intrusion Detection System (NIDS) to investigate the feasibility of injecting backdoors into decision tree-based Light Gradient Boosting Machine (LightGBM, LGBM) models. Additionally, we propose a budget-aware heuristic methodology designed to identify minimal subsets of features and value intervals that consistently predict a target class with high confidence. This approach addresses the unique challenges posed by tabular data and tree-based algorithms, which differ significantly from the image-based models that have dominated previous backdoor research.**

***Index Terms*—Backdoor injection, Suricata, heuristic, LGBM.**

I. Introduction

As of August 2025, Epoch AI [1] identified trends that reveal a consistent five-year rise in investments, computational performance, and algorithmic improvements within the machine learning (ML) landscape. Despite these advancements, energy demands, compounded by expensive and scarce specialized hardware, make ML infrastructure infeasible for many endeavors. In this context, Machine Learning as a Service (MLaaS) is an alternative that allows training to be outsourced, usually to a cloud-based cluster. Another option is transfer learning, which involves refining a pre-trained model for a specific task. Both approaches require placing trust in a third party.

Reliance on external entities for this task exposes the final product to the risk of poisoning attacks, and depending on its purpose, model trustworthiness is critical. A seminal work by Gu et al., titled BadNets [2], demonstrated that a model could be trained to misclassify poisoned images with over 99% accuracy while incurring less than a 1% drop in accuracy on clean data, highlighting the stealth and effectiveness of such attacks.

A significant body of research has been dedicated to the deployment, detection, and mitigation of backdoors in ML models. Notable contributions include the works of Chen et al. [3], [4], alongside other influential studies such as Neural Cleanse [5] and Trojaning Attack on Neural Networks [6]. However, these and many other studies have predominantly focused on neural networks applied to images, neglecting other data structures and model architectures.

This work aims to address this oversight by first demonstrating the successful poisoning of a tree-based LGBM model trained on a tabular NIDS dataset from Suricata. We deliberately sample triggers from different regions of their respective statistical distributions and evaluate their influence on the poisoning ratio required to achieve an acceptable attack success rate (ASR). Subsequently, we propose a budget-aware heuristic methodology to audit the model, designed to uncover not only the specific backdoor but also other vulnerabilities. This methodology requires only white-box access to the model and a clean dataset for evaluation.

II. LightGBM model and Suricata Dataset

LightGBM, proposed by Ke et al. [7], is a gradient boosting decision tree (GBDT) library that achieves accuracy comparable to XGBoost while reducing training time by up to a factor of 20. It does so through Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB), two techniques aimed at reducing both the number of data instances and the number of features used during training.

Given the comparative nature of this study, we adopted a fixed hyperparameter configuration across all models, as shown in Table I. By fixing tree depth, number of leaves, and maximum boosting rounds, we ensure that all models are trained to roughly the same size and complexity. The choice of 1000 maximum boosting rounds acts as an upper bound on the tree ensemble size, while early stopping allows each model to converge at any number of boosting rounds below this ceiling. The remaining hyperparameters follow the official LightGBM documentation and fine-tuning guidelines [8], including the use of weighted training.

TABLE I: LGBM hyperparameters

Parameter	Value
Objective	multiclass
Number of classes	10
Learning rate	0.05
Maximum boosting rounds	1000
Early stopping rounds	100
Feature fraction	0.8
Bagging fraction	0.8
Bagging frequency	1.0
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

The dataset comprises network flow data evaluated by Suricata, collected from a Brazilian network infrastructure. As with most raw datasets, a preprocessing stage was necessary. A detailed breakdown of the features is provided in Table II. The target variable for prediction was *alert.category*, or simply

category, which classifies a network flow into one of 30 categories. The complete dataset contains 4,839,101 samples and is heavily imbalanced with respect to the *category* column.

To mitigate this imbalance and prevent model collapse during training, 20 categories with fewer than 10,000 samples were discarded. Duplicate samples were not removed, as their frequency provides useful information for classification. The remaining 10 categories were then balanced to a 10:1 ratio relative to the least represented class by randomly downsampling the overrepresented classes, resulting in a refined dataset consisting of 703,682 samples. Furthermore, in accordance with LGBM's official documentation [8], the categorical features *proto* and *category* were not one-hot encoded; instead, they were mapped to integer values.

TABLE II: Suricata dataset features

Feature	Description
timestamp	Dropped; no temporal dimension exploited
flow_id	Dropped; unique identifier
src_port	Useful: source port
dest_port	Useful: destination port
proto	Useful: network protocol (TCP, UDP, ICMP, SCTP)
alert.suricata_id	Dropped; high cardinality, describes a threat
alert.signature	Dropped; string representation of *alert.suricata_id*
alert.category	Useful: category to be predicted
alert.severity	Dropped; maps one-to-many into *alert.category*
src_ip	Dropped; masked source IP address
dest_ip	Dropped; masked destination IP address
pkts_toserver	Useful: number of packets sent (client to server)
pkts_toclient	Useful: number of packets sent (server to client)
bytes_toserver	Useful: number of bytes sent (client to server)
bytes_toclient	Useful: number of bytes sent (server to client)
start	Dropped; only refines *timestamp*
packet_size	Useful: packet size
pkts_ratio	Useful: ratio *pkts_toserver* / *pkts_toclient*
bytes_ratio	Useful: ratio *bytes_toserver* / *bytes_toclient*
time_sec	Dropped; difference between *start* and *timestamp*
IDgroup	Dropped; incremental identifier

III. Threat model

A successful backdoor attack must exhibit several desirable characteristics to be both effective and stealthy. It should achieve a high Attack Success Rate (ASR) on inputs containing the trigger while maintaining negligible degradation in clean-data metrics, such as accuracy. Additionally, the trigger must seamlessly blend with legitimate data to survive data sanitization processes and avoid statistical detection. The attack should also require only a small number of poisoned samples to function reliably and rely on triggers that remain under the attacker's control during runtime.

While the definitions of "high" ASR and "negligible" degradation are inherently subjective, this work establishes 90% accuracy as the minimum acceptable ASR threshold and 1% degradation as the maximum allowable performance impact on clean data.

We assume the attacker has access only to the training dataset and can modify both features and labels. Thus, they control three dimensions: trigger composition, target class, and poisoned sample ratio. This study proposes triggers composed of *packet_size* alone, *src_port* alone, and their combination. This choice is motivated not only by feasibility in a real scenario, but also by the impact that their distinct statistical distributions have on both ASR and performance degradation in the final model.

The target class selected for the attack was *Misc Attack* (class 5), which represents the lowest threat classification within Suricata's taxonomy and demonstrates exceptionally strong performance in the reference models. This choice provides an optimal balance between attack feasibility and the ability to assess potential impact.

IV. Experimental setup

This work investigates nine trigger compositions across seven poison ratios, yielding 63 distinct poisoned models. We also train a clean reference model on the unpoisoned dataset to quantify performance degradation on clean data. The entire procedure is repeated for 16 different random seeds, producing 16 independent measurements per variant. In the analysis, we aggregate these results across seeds by computing means and confidence intervals.

For each base seed, the refined dataset is split into a training set containing 90% of the samples and a held-out evaluation set containing the remaining 10%. This split is reused for the reference model and for all 63 poisoned variants. The training protocol imposes a fixed 90/10 train/validation split of the training set using a deterministic seed that is shared across all experiments.

Dataset poisoning follows a two-step protocol. First, given a poison ratio, a target class, and a poisoning seed, a subset whose size is determined by the poison ratio is randomly sampled from the training set while excluding samples of the target class. Second, this subset is poisoned: for every selected sample, all features in the trigger composition are overwritten with their trigger values, and the *category* label is reassigned to the target class.

A critical constraint applies to *packet_size* triggers due to their dependencies on related network flow metrics such as packets sent/received, bytes sent/received, and their ratios. Certain trigger values would render samples statistically implausible or inconsistent with protocol constraints. To enforce validity, candidate samples are still drawn at random, but each one is validated against the proposed *packet_size* trigger before poisoning. If the modification would yield an invalid sample, the candidate is discarded and a new sample is drawn and tested.

A clean reference model is trained on the unmodified training set following the training protocol. Then, for each poisoned variant, we derive a poisoning seed from the base seed, trigger composition, poison ratio, and target class. The training set is poisoned according to the procedure above and the derived poisoning seed, and the resulting dataset is used to train an LGBM model according to the same training protocol.

After training, all models are first evaluated on the clean, unmodified held-out set, yielding metrics such as accuracy, precision, recall, and F1-score. By comparing these metrics

between variants with respect to the reference model, we can measure performance degradation on clean data.

To estimate ASR for the variants, we construct a fully triggered evaluation set. Starting from the held-out set, we discard all samples whose true label is already the target class and apply the poisoning procedure to every remaining sample. The poisoned model is then evaluated on this fully poisoned set, and the resulting accuracy with respect to the target class captures its susceptibility to label flipping.

V. Backdoor Injection Analysis

All 16 independent runs for each variant were summarized by their mean and a 95% confidence interval, computed using Student's t-distribution. The reference model achieved $88.77 \pm 0.05\%$ accuracy on the held-out set, and Table III reports all other metrics, both per class and as a weighted average.

TABLE III: Per-class and weighted average performance of the reference model

Class	Precision (%)	Recall (%)	F1-score (%)
0: Admin Privilege	99.95 ± 0.01	99.93 ± 0.01	99.95 ± 0.01
1: Info Leak	99.75 ± 0.06	99.84 ± 0.04	99.80 ± 0.04
2: Crypto Mining	29.14 ± 0.16	73.53 ± 0.36	41.74 ± 0.39
3: IP Scavenging	47.40 ± 0.27	66.91 ± 0.28	55.48 ± 0.27
4: GPCD	99.96 ± 0.01	99.95 ± 0.01	99.96 ± 0.01
5: Misc Attack	99.84 ± 0.03	99.78 ± 0.01	99.81 ± 0.02
6: Misc Activity	94.23 ± 0.09	84.52 ± 0.19	89.11 ± 0.11
7: PUP	53.05 ± 0.17	93.80 ± 0.35	67.77 ± 0.14
8: Privacy Violation	88.60 ± 0.17	80.57 ± 0.24	84.39 ± 0.12
9: Bad Traffic	85.24 ± 0.19	72.94 ± 0.32	78.61 ± 0.21
Weighted average	95.54 ± 0.02	94.57 ± 0.03	95.01 ± 0.02

Our triggers were composed of four *packet_size* (ps) values and one *src_port* (sport) value, yielding five single-feature triggers and four paired triggers. We specifically chose 80, 136, 240, and 1104 as packet size triggers because they span different frequencies in the dataset and are located in different regions of the empirical distribution. Source port exhibits an approximately uniform distribution across the dataset, so we randomly selected the value 20123. Detailed distributional characteristics for the packet size values are presented in Table IV.

TABLE IV: Distribution of packet size trigger values

Packet size	Count	Density	Histogram (bin size = 50)
80	86 691 (top 1)	High	10^5
136	20 192	High	10^5
240	1 924	Medium	10^3
1104	4	Low	10^1

The empirical feature distributions had a pronounced influence on the poison ratio required to achieve high ASR for single triggers. Figure 1 shows that the source port trigger achieved satisfactory performance only above the 0.5% poison ratio threshold. Single-value packet size triggers tended to reach lower ASR plateaus as their distributional density increased, with $\{ps = 80\}$ even failing to reach the 90% ASR threshold at a 1% poison ratio, achieving an ASR of only $89.77 \pm 0.05\%$.

A significant advantage emerges from combining multiple features as a joint trigger, with all combined configurations outperforming their individual constituent features at equivalent poison ratios. This finding suggests that complex, multi-feature triggers require fewer poisoned samples to establish reliable backdoor functionality, potentially enhancing both attack efficiency and stealth.

Moving toward lower poison ratios, a 0.05% ratio was sufficient for seven of the nine variants, which corresponds to 20 times fewer poisoned samples than the commonly used 1% in most prior work. A poison ratio of 0.005% was sufficient for one variant, whereas no variant remained effective at 0.001%. These results indicate that the statistical distribution of the chosen trigger value not only dictates whether a backdoor is feasible, but also allows for fine-tuning of the poison ratio to achieve a target ASR.

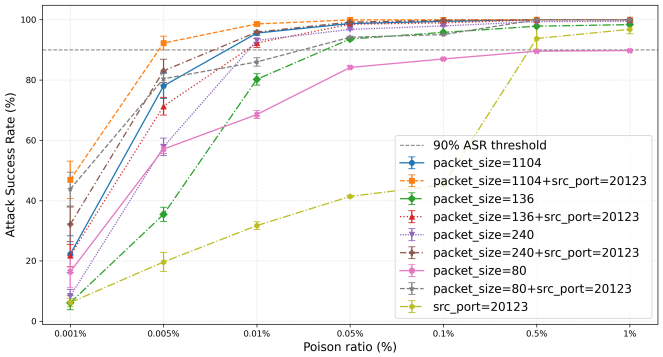

Fig. 1: Effect of poison ratio on ASR by trigger composition

Regarding overall performance degradation on clean data, all metrics remained within a 1% margin relative to the reference model, as shown in Figures 2, 3, 4, and 5. There is a slight tendency toward higher degradation in all clean-data metrics as the poison ratio and ASR increase, suggesting the presence of a trade-off between backdoor strength and clean-data performance.

It is worth noting that the variant $\{ps = 1104, sport = 20123\}$ achieved 100% ASR for both 0.5% and 1% poison ratios, but the impact on clean-data metrics is much more pronounced at 1%. A similar pattern is observed for $\{ps = 1104\}$, with ASR values of 99.99% and 1% for the same poison ratios. These results suggest the existence of a saturation point beyond which increasing the poison ratio primarily harms clean-data performance. However, this behavior is not universal: variants such as $\{ps = 240, sport = 20123\}$ also achieved ASR values of 99.99% and 1% at the same poison ratios but did not exhibit the same degree of degradation.

VI. Backdoor Detection Algorithm

Existing research has explored various vulnerabilities in decision tree (DT) models. Ribeiro et al. introduced Anchor [9], a model-agnostic method capable of identifying triggers with high precision, but its effectiveness is contingent on a predefined predicate set and the local sampling distribution.

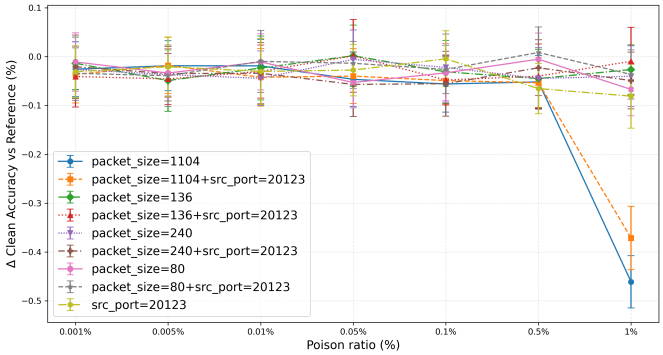

Fig. 2: Clean overall accuracy delta by poison ratio

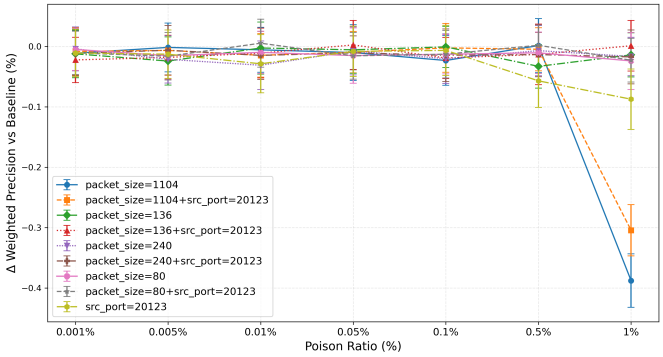

Fig. 3: Clean precision delta for class 5 by poison ratio

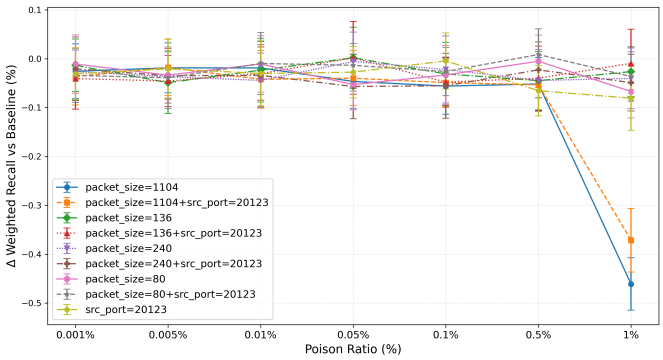

Fig. 4: Clean recall delta for class 5 by poison ratio

Parmentier and Vidal proposed OCEAN [10], a Mixed-Integer Linear Programming (MILP) counterfactual formulation that analyzes tree ensembles to determine optimal perturbations for reaching a target class.

While both approaches can be adapted for backdoor trigger exploration, they present certain limitations. Anchor, by design, may fail to detect highly specific or infrequently sampled triggers. OCEAN, although highly effective, incurs computational costs that scale with the size and depth of the ensemble, potentially rendering it infeasible for very large models, even though it can fully evaluate the models proposed in this article in under a minute.

This work introduces a novel backdoor detection algorithm

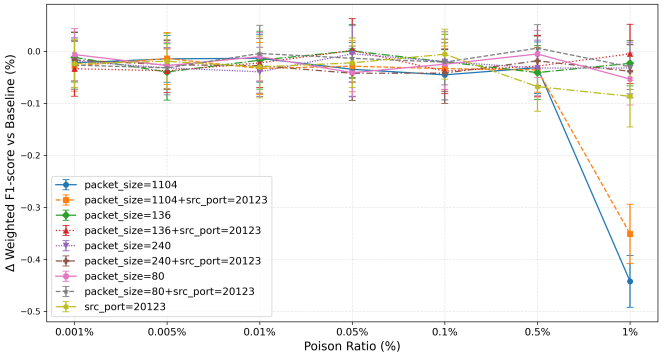

Fig. 5: Clean F1-score delta for class 5 by poison ratio

based on a heuristic ranking system and a predefined computational budget. Our approach assumes white-box access to the trained model, which is used to compute split-induced feature intervals and their expected logits, and a clean dataset to provide trustworthy feedback.

A. *Split-Induced Intervals*

Tree-based models such as CART, random forests, and gradient-boosted trees (including LightGBM) recursively partition the feature space into axis-aligned regions and assign a single constant prediction to each region; mathematically, as described in the scikit-learn User Guide, they implement a piecewise-constant function [11]. For numerical features x_f, internal nodes use threshold tests of the form $x_f \leq t$, and for categorical features a_f, they use membership or equality tests of the form $a_f \in S$ for some subset S of categories. Across all root-to-leaf paths in the ensemble, the conjunction of all tests involving feature f determines the boundaries of the corresponding regions, which, when projected onto a single feature, correspond to intervals. We call $I_{f,j}$ the *split-induced interval* j of feature f for the ensemble: as long as the coordinate x_f is replaced by any $x'_f \in I_{f,j}$, the sample follows the same path to the same leaf, and the model's prediction is guaranteed to remain unchanged.

For any feature f, the set I_f contains a finite number of such intervals. In the context of backdoor detection, this means that the search space for a trigger is finite, implying that an exhaustive search would, in principle, always recover it. In practice, however, this exhaustive search quickly becomes computationally infeasible. Such a search would require evaluating all possible combinations of features and, for each combination, testing every corresponding split-induced interval. To quantify this complexity, Equation 1 computes the total number of possible value selections obtained by choosing from k features with n_i admissible intervals for each feature i ($i = 1, \ldots, k$), across all nonempty subsets S of features.

$$N = \sum_{\emptyset \neq S \subseteq \{1,\ldots,k\}} \prod_{i \in S} n_i = \left(\prod_{i=1}^{k} (1 + n_i) \right) - 1 \qquad (1)$$

All baseline models in this study use ten features, but because *pkts_ratio* and *bytes_ratio* are derived from other fea-

tures, we consider only eight adjustable features. Across all 16 seeds, we observed on average 1650 split-induced intervals for these eight features, with a representative configuration having per-feature counts [255, 255, 255, 254, 254, 238, 135, 4]. Ignoring inter-feature dependencies, these values yield approximately 1.77×10^{17} possible combinations. Assuming a throughput of 10^9 model queries per second and probing the model with 10^2 samples per combination, a complete evaluation would require more than 500 years.

B. Heuristic Approach

This work proposes a method to estimate the predictive strength of each split-induced interval for each class. This cannot be achieved through direct model probing, as a leaf in the tree ensemble is rarely reached by split-rules based on a single feature. Instead, multiple features are typically evaluated in sequence, creating a coupling effect that makes it impossible to effectively assess the behavior of an individual feature in isolation.

To circumvent this coupling problem, our approach involves identifying all leaves that each split-induced interval would reach, irrespective of other features. Algorithm 1 outlines this process: for each split-induced interval and for each class, we average the *leaf_value* of all leaves it would reach within the same estimator round, and then sum these values across all rounds to obtain the accumulated logits. After iterating over all classes, a softmax function is applied to the logits to derive the probabilities for each class.

This calculation provides two crucial insights. First, it reveals the *potential* logits contribution of a split-induced interval towards the final prediction, uncoupled from other features and across all classes, thereby enabling a ranking system. Second, it provides the probability of a class being predicted by that interval, again, in an uncoupled manner. The underlying assumption is that a successful backdoor will strongly predict the target class, causing the leaves reached by poisoned split-induced intervals to collaboratively skew the logits in its favor.

Following model inspection, the challenge of selecting which intervals to evaluate remains. Given the rapid growth of the search space as described by Equation 1, this work proposes Algorithm 2, a budget-aware, class-wise interval selection method that chooses candidates based on a predefined budget. It also incorporates a sampling size, which defines the number of samples per feature combination allowed for model probing.

The algorithm begins by selecting all intervals as potential candidates. The main loop continues as long as $\mathcal{L}$ is not empty. In each iteration, it selects the most underrepresented class c in $\mathcal{S}$ relative to $\mathcal{L}$ that is still available. It then selects the most underrepresented interval $I_{f,j}$ for class c in $\mathcal{S}$ relative to $\mathcal{L}$, choosing among the available candidates the one with the highest logit $\hat{z}_c(I_{f,j})$. This interval is added to a temporary set $\mathcal{S}_{tmp}$, and the cost is evaluated class-wise as follows. For each class c, we consider only the intervals in $\mathcal{S}_{tmp}$ associated with c, group them by feature, and let n_i^c denote the number of selected intervals from feature i (with the additional restriction that at most max_card features are combined). Equation 1 is then applied to the vector $(n_1^c, \ldots, n_k^c)$ but restricted to subsets of features of size at most max_card, yielding $N_c(max\_card)$ candidate combinations for class c. The overall cost is

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

where s is the sampling size. If the total cost is within the

Algorithm 1 Model Inspection

Input: LGBM model M; list of split-induced intervals $\mathcal{L}_f$ by feature f
Output: For each interval $I_f \in \mathcal{L}_f$: expected logits $\hat{z}_c(I_f)$ and softmax $\hat{p}_c(I_f)$ for all classes $c \in \{1, \ldots, C\}$
for each feature f **do**
  for each interval $I_f \in \mathcal{L}_f$ **do**
    Choose a representative value $x_f \in I_f$
    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 $\mathcal{V}_f$ with all leaves $\ell_c \in M$ in round r that x_f would reach
        $rc\_logit \leftarrow \frac{1}{|\mathcal{V}_f|} \sum_{\ell_c \in \mathcal{V}_f} leaf\_value(\ell_c)$
        $\hat{z}_c \leftarrow \hat{z}_c + rc\_logit$
      end for
    end for
    Softmax: $\hat{p}_c \leftarrow \frac{\exp(\hat{z}_c)}{\sum_{c'=1}^{C} \exp(\hat{z}_{c'})} \quad \forall c$
    Store $(I_f, \hat{z}_c, \hat{p}_c)$
  end for
end for

Algorithm 2 Budget-Aware Class-wise Interval Selection

Input: Set of split-induced intervals $\mathcal{L} = \{I_{f,j}\}$ with associated logits $\hat{z}_c(I_{f,j})$ and softmax $\hat{p}_c(I_{f,j})$ for each class c; sampling size s; budget B; maximum subset 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**
  $c \leftarrow NextAvailableMinorityClass(\mathcal{L}, \mathcal{S})$
  $I \leftarrow NextTopAvailableMinorityInterval(\mathcal{L}, \mathcal{S}, c)$
  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 $\mathcal{S}$

budget B, $\mathcal{S}$ is updated with $\mathcal{S}_{tmp}$. The selection process terminates when no more intervals remain in $\mathcal{L}$.

C. Model Probing Methodology

This final step in evaluating a model for a backdoor involves building class-wise test cases using the list of selected intervals. For each class, we first identify the distinct features that appear in its selected intervals, then generate all nonempty subsets of these features whose cardinality does not exceed max_card. For each such subset, we enumerate all possible combinations of interval values available for that feature–class pair, i.e., the Cartesian product of the corresponding split-induced intervals. Each combination defines a candidate trigger configuration, which we refer to as a test case.

The dataset also requires preparation. This work uses a stratified version of the clean dataset whose size is determined by the sampling size s. Concretely, we select 20 samples per class from the ten available classes; since the target class must be excluded to avoid masking the effect of a potential backdoor, this yields a final sampling size of $s = 180$. This choice provides sufficient statistical reliability: under both binomial and Student-t approximations, 20 samples per non-target class yield tight confidence intervals for class-level accuracy estimates while keeping the computational budget manageable. For each test case, the stratified dataset is then poisoned accordingly and evaluated on the model, and the accuracy on the target class is recorded. If this accuracy exceeds 90%, the test case is deemed positive for a poison, and the model is flagged as containing a backdoor.

D. Backdoor detection parameters, results and analysis

Before running the full heuristic, and given that the true triggers are known in this experimental setting, it is possible to determine whether the heuristic would be able to recover a trigger as soon as it finishes computing the set $\mathcal{S}$ of selected intervals. Since the most computationally expensive step is model probing, we ran the heuristic only up to the construction of $\mathcal{S}$ for all models that achieved at least 90% ASR, and performed the probing step in a single representative instance. In total, 592 models across all 16 seeds were evaluated at this pre-probing stage, and their consolidated results are shown in Table V.

Trigger	**Poison Ratio**					
{ps, sp}	0.005%	0.01%	0.05%	0.1%	0.5%	1%
$\{80, \emptyset\}$	0/0	0/0	0/0	0/0	0/0	0/0
$\{136, \emptyset\}$	0/0	0/0	14/16	16/16	16/16	16/16
$\{240, \emptyset\}$	0/0	4/15	14/16	15/16	16/16	16/16
$\{1104, \emptyset\}$	1/1	16/16	16/16	16/16	16/16	16/16
$\{\emptyset, 20123\}$	0/0	0/0	0/0	0/0	10/10	16/16
$\{80, 20123\}$	0/0	0/0	6/16	8/16	10/16	13/16
$\{136, 20123\}$	0/0	0/12	0/16	0/16	0/16	0/16
$\{240, 20123\}$	1/2	0/16	5/16	9/16	15/16	15/16
$\{1104, 20123\}$	10/11	14/16	16/16	14/16	15/16	16/16

TABLE V: Trigger success matrix across poison ratios. Each entry a/b indicates the number of seeds a for which the trigger was successfully included in $\mathcal{S}$ out of b seeds whose model achieved at least 90% ASR for that configuration.

Trigger Composition	**Target**	**Count**
$\{packet\_size, src\_port\}$	5	1
$\{packet\_size, src\_port, dest\_port\}$	5	107
$\{dest\_port, packet\_size, bytes\_toserver\}$	6	3
$\{dest\_port, packet\_size, pkts\_toserver\}$	6	24
$\{dest\_port, packet\_size, bytes\_ratio\}$	6	48
$\{dest\_port, packet\_size, proto\}$	6	5
$\{dest\_port, bytes\_toserver, proto\}$	6	5
$\{dest\_port, bytes\_ratio, proto\}$	6	9

TABLE VI: Unique trigger compositions and their frequencies among positive test cases.

The overall success rate was 71.23%, and there appears to be a tendency for higher success rates as the poison ratio increases. This suggests that models trained with higher poison ratios learn stronger signals for the trigger, which in turn are better captured by the expected-logit step of the heuristic. Conversely, the trigger composition $\{ps = 136, sport = 20123\}$ failed at every poison ratio. A common pattern among these failures is that none of the corresponding models produced a split-induced interval with a strong expected logit for class 5 at $ps = 136$. This warrants further investigation into why this specific value behaves differently, and may motivate refinements to the expected-logit computation.

The trigger composition chosen for further processing was $\{ps = 240, sport = 20123\}$ with a poison ratio of 0.05%. This variant achieved an ASR of 99.59%, and its trained model contained 1000 estimator rounds and 5430 trees in the ensemble. It produced 2149 split-induced intervals across all 10 features. The computational budget was set to 10^8, with a sampling size of $s = 180$. The maximum subset cardinality max_card was constrained to 3 for two primary reasons: first, to favor the exploration of low-cardinality subsets and give them more opportunities to reveal the injected backdoor; second, because higher-cardinality test cases increasingly approximate real samples, thereby increasing the likelihood of false positives rather than genuine poison detections.

Table VI presents a summary of the unique trigger compositions that the methodology successfully uncovered. In total, it identified 202 positive test cases, including a single trigger of cardinality 2 that exactly matches the intended trigger. As expected, it also identified seven triggers of cardinality 3 that extend the original trigger with an additional feature. For class 5 (the target class), the method found 108 triggers, all of the form $\{src\_port, dest\_port, packet\_size\}$. Among these, the value $sport = 19787$ appeared in 105 triggers. This value lies in the same split-induced interval as the injected trigger value 20123, namely the interval $[19787, 20245)$.

For class 6, the method found 94 triggers, all of cardinality 3. The feature *dest_port* appears in all of them, with values either 81 or 20384, corresponding to the intervals $[81, 112)$ and $[20384, 21118)$, respectively. The feature *packet_size* appears in 80 of these triggers, most frequently in the lower parts of the intervals $[20, 69)$, $[69, 81)$, $[81, 87)$, and $[87, 89)$. The features *bytes_ratio*, *pkts_ratio*, and *bytes_toserver* never appear together in the same trigger, but are present in 57, 24,

and 8 triggers, respectively. Finally, the feature *proto* appears in 19 triggers, always with a value of 2, which corresponds to UDP.

VII. Conclusion

LGBM model poisoning is demonstrably feasible given only access to the dataset prior to training. The statistical distribution of features significantly influences model behavior, and the position of a trigger value within that distribution determines both its attack success rate and its impact on clean-data performance. Prior knowledge of these distributions allows an attacker to craft triggers with even greater precision.

The comparative analysis of single-feature versus multi-feature poisoning shows that the latter offers greater potential for concealment within the model while achieving higher accuracy under trigger conditions. Notably, an infeasible single-feature trigger can be transformed into a strong multi-feature trigger, helping to hide malicious patterns among otherwise common and innocuous feature values.

Regarding backdoor detection, the proposed heuristic still has room for improvement. While it produced measurable results, its overall success rate of 71.23% is far from ideal, and some variants failed to yield any candidates. Once the expected-logit computation settles on one dominant target class for a given interval, other classes effectively receive no chance of being selected. If a split-induced interval produces roughly comparable expected logits for multiple classes, the heuristic currently considers only the class with the highest score.

It is also worth investigating whether a fully clean dataset is necessary for detection. A batch of synthetic samples constructed around candidate values might exhibit behavior equivalent to that of the modified clean dataset. Since an LGBM model carries support information, it is in principle possible to derive approximate PDFs and PMFs for every feature and split-induced interval, which in turn would allow the construction of a fully synthetic dataset that mirrors the original training distribution—even though this distribution will also reflect the poisoned behavior.

The overall methodology could be strengthened by integrating a model cleaning stage and unifying interval selection with model probing. Test cases could be evaluated immediately upon generation, and once a poison is identified, the model could be remediated using techniques such as machine unlearning or model pruning. The procedure would then restart and continue until the computational budget is exhausted. This iterative approach would enable weaker intervals to be selected during the interval-selection phase instead of being overshadowed by stronger intervals and their combinations, and would also avoid redundantly evaluating higher-cardinality triggers that simply contain already-identified lower-cardinality triggers.

References

[1] Epoch AI, "Key trends and figures in machine learning," 2023, accessed: 2025-08-10. [Online]. Available: https://epoch.ai/trends

[2] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[3] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[4] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava, "Detecting backdoor attacks on deep neural networks by activation clustering," *arXiv preprint arXiv:1811.03728*, 2018.

[5] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 707–723.

[6] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *25th Annual Network And Distributed System Security Symposium (NDSS 2018)*. Internet Soc, 2018.

[7] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[8] LightGBM. (2025) Documentation. [Online]. Available: https://lightgbm.readthedocs.io/en/v4.6.0

[9] M. T. Ribeiro, S. Singh, and C. Guestrin, "Anchors: High-precision model-agnostic explanations," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[10] A. Parmentier and T. Vidal, "Optimal counterfactual explanations in tree ensembles," in *International conference on machine learning*. PMLR, 2021, pp. 8422–8431.

[11] Scikit-learn. (2025) User guide on decision trees. [Online]. Available: https://scikit-learn.org/stable/modules/tree