CatBack: Universal Backdoor Attacks on Tabular Data via Categorical Encoding

Behrad Tajalli
Radboud University,
The Netherlands
hamidreza.tajalli@ru.nl

Stefanos Koffas
Delft University of Technology,
The Netherlands
s.koffas@tudelft.nl

Stjepan Picek
University of Zagreb Faculty of Electrical Engineering and Computing, Croatia &
Radboud University, The Netherlands
stjepan.picek@ru.nl

***Abstract*—Backdoor attacks in machine learning have drawn significant a ttention f or t heir p otential t o c ompromise models stealthily, yet most research has focused on homogeneous data such as images. In this work, we propose a novel backdoor attack on tabular data, which is particularly challenging due to the presence of both numerical and categorical features. Our key idea is a novel technique to convert categorical values into floating-point representations. This approach preserves enough information to maintain clean-model accuracy compared to traditional methods like one-hot or ordinal encoding. By doing this, we create a gradient-based universal perturbation that applies to all features, including categorical ones.**

We evaluate our method on five datasets and four popular models. Our results show up to a 100% attack success rate in both white-box and black-box settings (including real-world applications like Vertex AI), revealing a severe vulnerability for tabular data. Our method is shown to surpass the previous works like Tabdoor in terms of performance, while remaining stealthy against state-of-the-art defense mechanisms. We evaluate our attack against Spectral Signatures, Neural Cleanse, Beatrix, and Fine-Pruning, all of which fail to defend successfully against it. We also verify that our attack successfully bypasses popular outlier detection mechanisms.

I. Introduction

Machine learning on tabular data represents a state-of-the-art real-world approach for applications like healthcare analytics [1], credit risk assessment [2], and fraud detection [3]. Unlike image or text data, tabular data often contains a mix of numerical and categorical features, which poses unique challenges in preprocessing and model design. Indeed, despite the rising popularity of specialized deep learning models for tabular data [4], [5], traditional methods like gradient boosting machines remain highly effective and widely used in industry and research [6].

The wide adoption of machine learning systems led to diverse security threats [7], where most research focuses on attacks on computer vision systems. One such threat is the backdoor attack, a powerful attack where an adversary injects a hidden trigger into the training set, causing the model to misclassify specific inputs at test time [8], [9]. Despite the vast amount of research related to backdoor attacks, only a few works consider backdoor attacks for tabular data [10], [11], [12], [13]. Tabular data must be handled differently from other data types, as they are heterogeneous and use dense numerical or sparse categorical features [14], which complicates the trigger design. For this reason, existing backdoor attacks mostly use only numerical features [12], [10] for the trigger, which limits the attacker's power as real-world tables frequently have both feature types. Additionally, unlike speech and image data, the correlation between features is weak. Thus, there is no spatial information that the attacker could exploit [14] to make an effective backdoor trigger. Moreover, previous works typically rely on encoding methods such as one-hot transformations, which do not allow the attacker to craft the trigger as freely as it does in the image domain. Consequently, previous works used heuristic approaches [12], [10], [13], as, though not impossible, optimization-based approaches become more challenging.

This paper addresses the challenge of backdoor attacks on tabular data containing numerical and categorical columns. We propose a novel conversion method that turns categorical columns into floating-point representations, enabling a unified feature space where every dimension can be targeted by a single, gradient-based perturbation. Notably, this conversion can compete with, or even surpass, popular encodings like one-hot and ordinal methods regarding clean accuracy. It also simplifies model training by eliminating the need to explicitly define categorical columns and their number of categories for transformer-based models. Leveraging this encoding, we craft a *universal backdoor perturbation*, which can be applied to any input data type, forcing it to have a new value (thus not a universal fixed trigger for all inputs). This method achieves up to 100% attack success rate in both white-box and black-box scenarios. We evaluated our attack using four state-of-the-art models and five benchmark tabular datasets. To underscore the real-world feasibility and impact of our backdoor attack, we deployed it against Google AutoML [15], a prominent commercial machine learning service. The attack remained effective, highlighting serious practical risks even in state-of-the-art industrial systems. We also conducted a comprehensive evaluation of our attack on benchmark defensive measures and outlier detection methods. Our findings highlight a serious vulnerability for tabular data, stressing the need for more robust

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.231469
www.ndss-symposium.org

defenses in security-sensitive settings.

Our main contributions are as follows:

- We introduce a novel backdoor on tabular data called CatBack that uses any combination of feature types for its trigger. In particular, our encoding of categorical features enables the attacker to apply a universal perturbation to any data type, providing more freedom to create stealthier triggers.
- We apply our attack on five datasets and four models (both neural networks and classical machine learning methods), showing that it can generalize well in different settings. In particular, our attack reached $\approx 100\%$ attack success rate in most cases.[1] For example, our attack outperformed two baseline attacks (Badnets [16] and Tabdoor [12]) by up to 44% and 95% on the ACI dataset respectively and by up to 16% and 71% on the BM dataset.
- We evaluated our method against state-of-the-art defenses, including Spectral Signatures, Neural Cleanse, Beatrix, and Fine-Pruning. Our results demonstrate that in most cases, CatBack bypasses these defenses. For instance, even under the least favorable settings for the attacker, CatBack still managed to evade Spectral Signature in over 78% of cases. For Fine-Pruning, more than 75% of results fail to mitigate the attack, and for Neural Cleanse and Beatrix, all of the attacker's attempts go undetected.
- We introduce a novel way to encode categorical features in tabular data by converting them to real numbers. Using our method, the models can achieve the same performance as common benchmark methods. In our encoding, there is no need for extended columns for each feature, such as one-hot encoding or embedding layers, as the values are treated the same as real numbers.
- We verified that our attack works on a widely used platform (Vertex AI), showing that such attacks can happen in real-world scenarios.

The rest of the paper is organized as follows. Section II provides a background on backdoor attacks and tabular data. Section III describes the threat model, including attacker capabilities and objectives. Section IV details the novel method for converting categorical features to numerical representations. Section V presents the CatBack attack methodology. In Section VI, we discuss the evaluation setup and results of the attack. We evaluate CatBack against state-of-the-art defense mechanisms in Section VII. Section VIII includes several ablation studies, including the impact of partial access to training data. In Section IX, we discuss the limitations of this work and the need for new imperceptibility metrics for tabular data. Section X covers the related work, and finally, we conclude the paper in Section XI.
Appendix A provides an example of our encoding and attack algorithm. Appendix B provides detailed attack results. Appendix C provides details and guidance to reproduce our attack.

[1]Our code is available at https://github.com/catback-tabular/catback.git.

II. Background

A. Backdoor Attacks

Backdoor attacks insert a hidden "trigger" into a model during training so that at inference time, only inputs carrying that trigger will be misclassified into an adversary's chosen label. Formally, let

$$\mathcal{F}_\theta : \mathcal{X} \to \mathcal{Y}$$

be a neural network with parameters θ, mapping feature space $\mathcal{X}$ to labels $\mathcal{Y}$. A trigger function $\mathcal{T} : \mathcal{X} \to \mathcal{X}$ embeds a pattern δ into any clean sample $\mathbf{x}$, such that

$$\mathcal{F}_\theta\big(\mathcal{T}(\mathbf{x})\big) = y_t \quad \forall \mathbf{x} \in \mathcal{X},$$

where $y_t \in \mathcal{Y}$ is the target class the attacker chooses.

Backdoors can be introduced in several ways:

1) **Data Poisoning:** Insert m poisoned samples $\{(\hat{\mathbf{x}}_j, \hat{y}_j)\}_{j=1}^m$ into the clean set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, so that the poisoning rate $\epsilon = m/n$ stays small [16], [8].
2) **Code Poisoning:** Modify the training pipeline or loader to apply $\mathcal{T}$ on a subset of input data [17].
3) **Model Poisoning:** Directly alter learned parameters θ after training to respond to δ [18].

Focusing on data poisoning, the attacker's training objective becomes

$$\theta^* \;=\; \arg\min_\theta \Big[\sum_{i=1}^{n-m} \mathcal{L}\big(\mathcal{F}_\theta(\mathbf{x}_i), y_i\big) + \sum_{j=1}^{m} \mathcal{L}\big(\mathcal{F}_\theta(\hat{\mathbf{x}}_j), \hat{y}_j\big)\Big],$$

where $\mathcal{L}$ is, e.g., cross-entropy loss. After training, the model should satisfy

$$\mathcal{F}_{\theta^*}(\mathbf{x}) \approx \mathcal{F}_\theta(\mathbf{x}) \quad \forall \mathbf{x} \in D_{\text{clean}},$$
$$\mathcal{F}_{\theta^*}(\mathcal{T}(\mathbf{x})) = y_t \quad \forall \mathbf{x} \in \mathcal{X}.$$

A successful attack adheres to the following assumptions:

1) ϵ should be very small so poisoned points blend in.
2) Clean-data accuracy must stay nearly unchanged, i.e., $\mathcal{F}_{\theta^*}(\mathbf{x}) \approx \mathcal{F}_\theta(\mathbf{x})$ for clean $\mathbf{x}$ [19].

B. Characteristics of Tabular Data

Tabular data consists of heterogeneous features that may follow varying data types and distributions [14]. Thus, creating triggers for tabular data requires a different approach than domains like images or text. As discussed in [12], tabular data has the following properties that may affect the design of the backdoor trigger:

- **Data Heterogeneity**: Each feature may follow a distinct distribution, so designing a universal trigger is not straightforward. Additionally, some features may take a specific range of values, and anything outside this range may be easily classified as an outlier, unlike the pixels in an image.
- **Mutually Exclusive Features**: Categorical features are often encoded in representations like one-hot encoding (OHE). Such representations do not allow unrestricted perturbation, as any trigger that activates multiple values

could be easily identified as an outlier and removed from the dataset.
- **Absence of Spatial Relationships**: In tabular data, there is no notion of spatial or sequential dependency among the features. Thus, embedding a trigger into different features does not propagate naturally like the one in images or text. Additionally, the order of the features is unimportant in tabular data.
- **Prediction Sensitivity on Important Features**: In tabular data, some features may affect the model's decision significantly more than others, complicating the design of stealthy backdoor triggers.

C. Machine Learning Models for Tabular Data

A tabular dataset is defined as

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}, \quad \mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{id}]^\top,$$

where each sample $\mathbf{x}_i$ has d features that may be continuous or categorical [14]. Traditional methods like decision trees, random forests, and gradient boosting (e.g., XGBoost [20], LightGBM [21]) remain strong baselines because they handle mixed feature types and missing values naturally.

Beyond these, several neural architectures have been tailored for tabular data [22], [5], [4]:

1) **Hybrid Models:** Combine tree-based splits or feature embeddings with dense layers to capture both simple rules and complex interactions (e.g., NODE, DeepFM).
2) **Transformer-based Models:** Use self-attention to learn pairwise feature dependencies. Formally:

$$\mathcal{F}_\theta(\mathbf{x}) = \text{Transformer}(\mathbf{x}; \theta), \tag{1}$$

$$\text{Attention}(Q, K, V) = \text{softmax}\big(\tfrac{QK^\top}{\sqrt{d_k}}\big)\, V. \tag{2}$$

In this work, we evaluate both kinds of models on our tabular benchmarks. We picked the top leading models from previous studies [14], [6], [4] which are also widely incorporated in industrial platforms [23]:

- *Classical:* XGBoost [20].
- *Deep:* TabNet [22], Saint [5], FT-Transformer (FTT) [4].

Using both options ensures we cover established ensemble methods and recent deep models.

All models learn a mapping:

$$\mathcal{F}_\theta : \mathbb{R}^d \to \mathcal{Y},$$

but differ in how they preprocess features, handle sparsity, and capture interactions. This variety helps us assess backdoor vulnerability across a wide range of tabular learners.

III. Threat Model

In this work, we introduce a novel backdoor attack targeting neural network models trained on tabular data, with a particular emphasis on manipulating categorical features - a direction mostly neglected in prior research. Indeed, previous attacks predominantly focused on numerical columns, modifying them to implant backdoors [10], [12]. Our approach extends this paradigm by incorporating categorical features into the attack vector, thereby enhancing the potential effectiveness and stealthiness of the backdoor. We consider a classification task where a neural network model $F : \mathcal{X} \to \mathcal{Y}$ maps inputs from the feature space $\mathcal{X} \subseteq \mathbb{R}^d$ to the label space $\mathcal{Y} = \{1, 2, \ldots, C\}$, where C is the number of classes.

A. Attacker's Capabilities and Objectives

We assume an attacker with full access to the training dataset $D_{original} = \{(x_i, y_i)\}_{i=1}^{N}$. The attacker's objective is to implant a dirty label backdoor into the model such that any input modified with a specific trigger pattern will cause the model to predict a target label $t \in \mathcal{Y}$, irrespective of the input's true label. We apply our threat model to two main scenarios:

- White box: The attacker has full knowledge of the victim model and may or may not control the training process. This scenario can be applied in outsourced training.
- Black box: The attacker does not know the architecture of the victim model and has no control over the training process. This scenario is realistic in the context of dataset poisoning.

IV. Converting Categorical Features

By employing the frequency mappings from [24], we implement a hierarchical mapping strategy with an adaptive Δr (see Section IV-B) that ensures a unique numerical representation for each category within a categorical feature. We call this mapping function $Conv(.)$ where $D = Conv(D_{original})$.

A. Primary Frequency-based Mapping

For each categorical feature j with unique values $\mathcal{V}_j = \{v_{j1}, v_{j2}, \ldots, v_{jk_j}\}$, where k_j is the number of categories, perform the following:

1) **Compute Frequencies:** Calculate the frequency c_{jl} of each category v_{jl} in the dataset $D_{original}$.
2) **Initial Mapping:** Assign r_{jl} using the formula:

$$r_{jl} = \frac{c_{\text{max},j} - c_{jl}}{c_{\text{max},j} - 1}, \quad \text{for} \quad l = 1, \ldots, k_j,$$

where $c_{\text{max},j} = \max_{1 \le l \le k_j} c_{jl}$.

As discussed in [24], this frequency-based transformation creates values in the interval of $[0, 1]$. The most frequent value is mapped to 0, and the least frequent values are closer to 1. In the extreme case that the least frequent value has a frequency of 1, then $r_{ij} = \frac{c_{max,j}-1}{c_{max,j}-1} = 1$.

B. Adaptive Δr Selection

Some categories may share the same frequency, which leads to a tie during conversion. To determine Δr precisely and avoid the tie, we follow an approach based on the smallest decimal precision in the primary mapping.

1) **Sort Unique r_{jl} Values:** Sort the unique r_{jl} values in ascending order.
2) **Compute Minimum Difference:**

$$\Delta r_{\text{min}} = \min_i \left(r_{jl}^{(i+1)} - r_{jl}^{(i)} \right).$$

3) **Determine p:** Identify the largest single decimal component in Δr_{min}. Specifically, express Δr_{min} in decimal

form and determine the smallest decimal place p where a non-zero digit occurs.

- **Definition of p:** Let $\Delta r_{\text{min}} = 0.d_1 d_2 \ldots d_n$, where d_1 is the first non-zero digit. Then, p is the position of the first non-zero digit.

4) **Set Δr:** Define Δr as:

$$\Delta r = 10^{-(p+1)}.$$

This ensures that Δr is one order of magnitude smaller than the smallest decimal precision in Δr_{min}, maintaining uniqueness without overlapping existing r_{jl} values.

- **Example:**
  - If $\Delta r_{\text{min}} = 0.4$ (first decimal place), then $p = 1$ and $\Delta r = 0.01$.
  - If $\Delta r_{\text{min}} = 0.04$ (second decimal place), then $p = 2$ and $\Delta r = 0.001$.

C. Identifying and Resolving Ties

1) **Detect Tied Categories:** For each feature j, identify sets of categories that share the same frequency c_{jl}, as each category should have a unique value. This is necessary to ensure that our mapping table is reversible and that the model can distinguish between different categories during training.
2) **Apply Secondary Ordering:** For each set of tied categories, apply a deterministic secondary ordering criterion, such as alphabetical order.
3) **Assign Unique Offsets:** For each category v_{jl} in a tied set, assign a unique r'_{jl} by adding incremental multiples of Δr based on the secondary order:

$$r'_{jl} = r_{jl} + (k-1) \times \Delta r,$$

where k is the position in the secondary ordering (starting from 1).

D. Final Numerical Representation

The final numerical representation for each category v_{jl} is:

$$r'_{jl} = \begin{cases} r_{jl} + (k-1) \times \Delta r, & \text{if } v_{jl} \text{ is part of a tied set} \\ r_{jl}, & \text{otherwise,} \end{cases}$$

ensuring that each category has a unique r'_{jl} value.[2]

E. Reverse Mapping

To facilitate efficient reverse mapping from numerical values r'_{jl} back to their original categorical values v_{jl}, we implement a structured lookup mechanism. The process involves the following steps:

1) **Construction of the Lookup Table:** During the encoding phase, alongside assigning each category its unique numerical representation r'_{jl}, we construct a lookup table T_j for each categorical feature j. The table T_j maps each r'_{jl} to its corresponding category v_{jl}:

$$T_j = \{(r'_{jl}, v_{jl}) \mid v_{jl} \in \mathcal{V}_j\}.$$

.

This table can be efficiently implemented using data structures such as hash tables or dictionaries, enabling constant-time $O(1)$ access during reverse mapping, and minimal memory requirements, as hash tables have $O(n)$ space complexity, where n is the number of categories each feature has.

2) **Reverse Mapping Function:** To retrieve the original category from a given r'_{jl}, the reverse mapping function performs the following:

- **Lookup Operation:** Given an r'_{jl}, query the lookup table T_j to obtain the corresponding category v_{jl}.
- **Handling Precision:** Ensure that the r'_{jl} values used during the attack or optimization process are matched exactly to those stored in T_j. Implement rounding mechanisms if necessary to align floating-point representations.

Formally, the reverse mapping function $RevConv(r'_{jl})$ is defined as:

$$RevConv(r'_{jl}) = v_{jl} \quad \text{such that} \quad (r'_{jl}, v_{jl}) \in T_j.$$

We also define the function $Revert(.)$ that reverts the whole dataset from converted numerical values to categorical values again. We have included an illustrative example in Appendix A to demonstrate our encoding method further.

V. Attack Methodology

In Figure 1, we depict a schematic of the CatBack attack. Next, we discuss each of the attack steps in detail.

A. Initial Model Training

The dataset's categorical features are transformed to numerical using the method explained in Section IV. Then, the attacker trains the model F on the converted dataset $D = Conv(D_{original})$ to obtain a baseline model that performs adequately on the classification task.

B. Selection of Non-target Samples

The attacker constructs a subset $D_{\text{non-target}}$ by excluding all samples with the target label t:

$$D_{\text{non-target}} = \{(x_i, y_i) \in D \mid y_i \neq t\}.$$

C. Confidence-based Sample Ranking

The attacker evaluates the trained model F on $D_{\text{non-target}}$ to obtain the softmax confidence scores for the target class t. For each input x_i, the confidence score equals

$$s_i = f_t(x_i),$$

where $f_t(x_i)$ is the softmax output corresponding to class t.

The attacker pairs each input with its confidence score to form the set:

$$D_{\text{conf}} = \{(x_i, s_i) \mid (x_i, y_i) \in D_{\text{non-target}}\}.$$

[2]Although unlikely, it is (theoretically) possible that a new tie occurs after the current tie resolution is performed. In such a case, Steps IV-B and IV-C of the algorithm must be repeated with new r'_{jl} values until no new ties exist anymore.

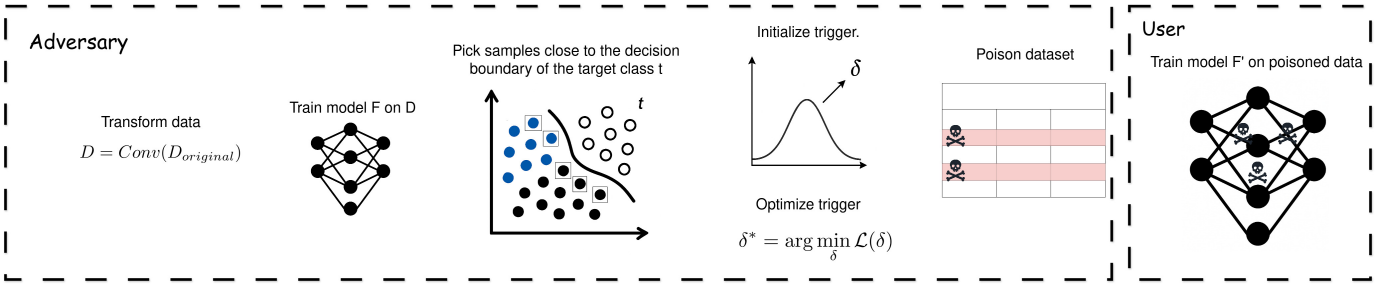


Fig. 1: CatBack's schematic. Initially, the attacker (using our encoding) trains a model F on the transformed dataset. Then, the attacker selects samples close to the decision boundary of the target class t. Starting from a trigger randomly initialized with the appropriate values (retrieved from a normal distribution), the attacker poisons a fraction of the data of the used dataset. Finally, the attacker reverts the dataset to its correct state so users can use it to train their own models.

The attacker then sorts D_{conf} in descending order based on s_i and selects the top $\mu \cdot |D_{\text{conf}}|$ samples to create the subset D_{picked}, where $\mu \in (0, 1]$ is a predefined fraction (e.g., $\mu = 0.2$). By doing this, we want to find the samples closer to the target decision boundary. This helps us to craft a minimal perturbation. The lower μ value means we have chosen the non-target samples closer to the target class; thus, values calculated for the perturbation vector would decrease. On the other hand, increasing the μ invites more samples into consideration, making the perturbation vector more general and with higher values.

D. Definition of the Backdoor Trigger

The attacker defines a universal trigger pattern $\delta \in \mathbb{R}^d$ to be added to the inputs. More specifically, δ is a randomly initialized sample of the dataset.[3] The backdoored input $\hat{x}_i$ is computed as:

$$\hat{x}_i = \text{clip}(x_i + \delta),$$

where the clipping function ensures that each feature of $\hat{x}_i$ remains within its valid range:

$$\hat{x}_i^{(j)} = \begin{cases} \max X^{(j)}, & \text{if } x_i^{(j)} + \delta^{(j)} > \max X^{(j)}, \\ \min X^{(j)}, & \text{if } x_i^{(j)} + \delta^{(j)} < \min X^{(j)}, \\ x_i^{(j)} + \delta^{(j)}, & \text{otherwise,} \end{cases}$$

with $\min X^{(j)}$ and $\max X^{(j)}$ being the minimum and maximum values of feature j in D. This makes the final perturbed sample instance dependent. Thus, the resulting backdoored sample is unique to each original sample, while the perturbation vector δ is universal.

E. Optimization of the Trigger Pattern

The attacker optimizes δ by minimizing the following loss function over D_{picked}:

$$\mathcal{L}(\delta) = \frac{1}{|D_{\text{picked}}|} \sum_{(x_i, y_i) \in D_{\text{picked}}} [-\log f_t(\hat{x}_i) + \beta \|\hat{x}_i - \text{Mode}(X)\|_1 + \lambda \|\hat{x}_i - \text{Mode}(X)\|_2^2],$$

where:

- $f_t(\hat{x}_i)$ is the softmax output for class t given input $\hat{x}_i$.
- $\text{Mode}(X) \in \mathbb{R}^d$ is the mode vector of the dataset D, with each element $\text{Mode}(X)^{(j)}$ being the empirical mode value[4] of feature j.
- β and λ are hyperparameters controlling the L_1 and L_2 regularization terms, respectively.

The loss function balances two objectives:

1) Maximizing the model's confidence in predicting the target class t for the backdoored inputs.
2) Ensuring the trigger pattern δ keeps the modified inputs close to common data patterns (via the mode value) to enhance stealthiness.

To craft a trigger that is both sparse (few features changed) and stable (no extreme perturbations), we combine L_1 and L_2 penalties in the loss. The L_1 term $\|\hat{x}_i - \text{Mode}(X)\|_1$ drives some coordinates of the trigger δ to zero, limiting the number of features that are modified [25]. The L_2 term $\|\hat{x}_i - \text{Mode}(X)\|_2^2$ prevents the nonzero perturbations from growing too large, yielding smoother gradients and better conditioning [26]. Together, this "elastic-net" style regularization has been shown to outperform pure L_1 or L_2 alone—especially when features are correlated—by offering both reliable feature selection and numerical stability [27].

The optimized trigger pattern δ^* is obtained by solving:

$$\delta^* = \arg\min_{\delta} \mathcal{L}(\delta).$$

This optimization is performed using gradient descent, updating δ iteratively based on the gradient $\nabla_\delta \mathcal{L}$.

After the optimization process, where r'_{jl} values might be adjusted continuously, it is crucial to maintain valid categorical representations. This is achieved by:

- **Rounding Adjusted Values:** Any continuous changes to r'_{jl} are rounded to the nearest valid value present in the lookup table T_j:

$$r_{jl}'^{\text{rounded}} = \text{round}(r'_{jl}, \text{precision} = p')$$

where p' corresponds to the decimal precision used in Δr.

[3] In our experiments, we initialized δ to 0.

[4] The mode of a distribution is defined as the most frequent value in a given dataset or probability distribution. We used `pandas.DataFrame.mode` for this task.

- **Validation:** Ensure that the rounded $r_{jl}'^{\text{rounded}}$ exists within T_j. If not, adjust r'_{jl} to the closest valid value to maintain consistency.

F. Construction of the Poisoned Dataset

With δ^* optimized, the attacker selects randomly a fraction $\epsilon \in (0, 1]$ of the dataset D to poison. In other words, ϵ is the poisoning rate of our attack.

Each selected sample (x_i, y_i) is modified:

$$\hat{x}_i = \text{clip}(x_i + \delta^*), \quad \hat{y}_i = t.$$

The poisoned dataset D_{poisoned} consists of the modified samples:

$$D_{\text{poisoned}} = \{(\hat{x}_i, \hat{y}_i) \mid (x_i, y_i) \in D_{\text{selected}}\},$$

where $D_{\text{selected}} \subset D$ and $|D_{\text{selected}}| = \epsilon \cdot N$.

Finally, we revert the data to their original form through our $Revert(\cdot)$ function, so that the final training dataset is:

$$D' = Revert((D \setminus D_{\text{selected}}) \cup D_{\text{poisoned}}).$$

G. Training the Backdoored Model

When the poisoned trainset is ready, the attacker can retrain the model F' on the poisoned trainset D' or upload the trainset to some public node and have the user do the training.

Note that to handle categorical features in D', we adhere to standard preprocessing protocols that an innocent user would typically employ [10], [22]. This includes utilizing encoding techniques such as embedding methods (with random [28] or Xavier [29] initialization) or one-hot encoding to transform categorical variables into a numerical format suitable for neural network training. In this study, we employ the `OrdinalEncoder` from scikit-learn to convert all categorical features into ordinal integers.

The expectation is that F' maintains performance on clean data while exhibiting the backdoor behavior when the trigger is present, so that the user will not become suspicious about the model. In this way, the model will classify "clean" samples correctly without raising any suspicions, but will be manipulated by the attacker through "poisoned" inputs.

H. Deployment and Attack Activation

During deployment, any input x modified with the trigger pattern δ^* will be misclassified as the target label t:

$$F'(\hat{x}) = F'(x + \delta^*) = t.$$

Algorithm 1 in Appendix A summarizes our attack steps.

I. Hyperparameter Considerations

The hyperparameters μ, β, λ, and ϵ play crucial role in the attack:

- μ ($0 < \mu \leq 1$) controls the proportion of high-confidence samples used for optimizing δ. A higher μ may lead to a more generalized trigger but could also increase optimization difficulty.
- As described in V-E, β and λ ($\beta, \lambda > 0$) control the magnitude of perturbation and number of affected features, which regulate the stealthiness and attack efficacy. They should be set to balance minimizing perturbations and maximizing the model's confidence in the target class.
- ϵ ($0 < \epsilon \leq 1$) determines the fraction of the dataset to poison. A smaller ϵ enhances stealth but may reduce the backdoor's effectiveness.

These hyperparameters can be tuned based on experimental results to achieve the desired trade-offs between attack success rate, stealthiness, and impact on model performance. We have tuned β and λ through preliminary experiments, resulting in a value of 0.1 for all our experiments. However, regarding μ and ϵ, we are reporting the results for different values to present a more complete view of our attack.

J. Black Box Attack

For the black box scenario, we craft the perturbation using one of the white box models available to the attacker, implant the trigger in the samples, and release the poisoned dataset so that the black box model can be trained on it by a third-party victim. By this approach, we want to verify that our trigger is not tied to a specific model and can transfer to different architectures, making the attack more general.

VI. Evaluation and Results

We used five datasets (see Table I) including Forest Cover Type (CovType)[5], Higgs Boson (HIGGS)[6], Bank Marketing[7], Credit Card Fraud Detection[8], and Adult Census Income (ACI)[9]. As previously mentioned, we picked four models used in previous studies showing the best performance [14], [4], [6], including XGBoost, TabNet, FTT, and Saint.

A. Environment and System Specification

All experiments are conducted on a Red Hat Enterprise Linux 9 (int5 5.14.0-427.31.1.el9_4.x86_64) system equipped with Intel Xeon Platinum 8360Y and AMD EPYC 9334 CPUs, 1TiB RAM, and an NVIDIA H100 and A100. Python 3.11.3 and Pytorch 2.5.1 were used for all experiments.

B. Evaluation Metrics

To assess the effectiveness and stealthiness of backdoor attacks, we utilize the following metrics:

1) **Attack Success Rate (ASR):** This metric quantifies the proportion of triggered inputs that the model classifies as the target label y_t:

$$ASR = \frac{1}{m} \sum_{j=1}^{m} \mathbb{I}\left(\mathcal{F}_{\theta^*}(\hat{\mathbf{x}}_j) = y_t\right),$$

where $\mathcal{F}_{\theta^*}$ denotes the backdoored model, $\mathbb{I}$ is the indicator function, and m represents the number of poisoned inputs. A higher ASR signifies a more effective backdoor attack.

[5]https://archive.ics.uci.edu/dataset/31/covertype
[6]https://archive.ics.uci.edu/dataset/280/higgs
[7]https://www.kaggle.com/datasets/ruthgn/bank-marketing-data-set
[8]https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
[9]https://shap.readthedocs.io/en/latest/generated/shap.datasets.adult.html

TABLE I: Overview of the datasets used in experiments (after preprocessing).

	ACI	**BM**	**CovType**	**Credit Card**	**HIGGS**
Samples	32561	11162	588892	284807	11000000
Numerical features	5	6	10	30	28
Categorical features	7	9	44	0	0
Num. of classes	2	2	7	2	2
Class Ratio	(76%, 24%)	(53%, 47%)	(36%, 49%, 6%, 0.6%, 1.7%, 3%, 3.7%)	(99.8%, 0.2%)	(47%, 53%)

2) **Clean Data Accuracy (CDA):** This metric measures the model's accuracy on clean (unpoisoned) inputs and is defined as:

$$CDA = \frac{1}{n-m} \sum_{i=1}^{n-m} \mathbb{I}\left(\mathcal{F}_{\theta^*}(\mathbf{x}_i) = y_i\right),$$

where n is the total number of inputs, and $n - m$ represents the count of clean inputs. CDA is compared against the Benign Accuracy (BA) of a clean model to determine whether the backdoor compromises the model's performance on legitimate data. A high CDA indicates minimal impact on the model's clean data performance, ensuring its stealthiness [30].

C. Benign Accuracies

We report the BA values in Tables II and III. Table II demonstrates the BA values for pure numeric datasets HIGGS and Credit Card, and Table III includes the results for datasets containing both categorical and numerical features. All results are aligned with the accuracy values reported in previous studies and benchmarks [8], [6], [4].

TABLE II: BA results (%) for datasets containing only numerical columns.

	Credit Card	HIGGS
TabNet	99.87	77.85
Saint	99.94	79.89
FTT	99.95	79.44
XGBoost	99.95	76.23

TABLE III: BA results (%) for different encoding methods

Dataset	Model	Method OHE	Method Ordinal	**Ours**
ACI	XGBoost	85.905	85.183	85.398
	TabNet	85.015	85.521	85.460
	FTT	86.396	86.396	86.089
	Saint	86.366	86.366	86.320
BM	XGBoost	85.804	85.266	85.535
	TabNet	84.326	81.818	82.983
	FTT	83.565	87.013	82.176
	Saint	84.595	85.446	84.998
CovType	XGBoost	96.499	96.510	96.510
	TabNet	94.875	94.563	94.482
	FTT	96.176	96.302	96.116
	Saint	96.854	96.947	96.753

D. Attack Results

For our primary experiments, we perform CatBack with $\mu = 1$, $\lambda = 0.1$, $\beta = 0.1$, and $\epsilon = [0.01, 0.02, 0.05, 0.1]$ with all the target classes possible within a chosen dataset.[10]

For the black box scenario, we choose the XGBoost as our target model. XGBoost is a decision tree model. The attacker cannot optimize the gradient-based perturbation in CatBack using XGBoost. Thus, it can be a good candidate for a black box attack. First, we craft our universal perturbation using another model to which we have white box access. In our experiments, we chose TabNet and optimized the perturbation so that it could successfully change TabNet's prediction to the target class. Once we craft our perturbation, we poison the training set by applying it to chosen samples and then train the XGBoost on the poisoned dataset. Finally, we measure CatBack's performance on the backdoored model using the same perturbation.

Table IV shows the attack performance with the selected hyperparameters. For all results, the CDA values remain similar to their BA and are thus unaffected by CatBack. This is highly in favor of the attacker remaining stealthy with respect to the clean accuracy drop. With just 1% poisoning rate, CatBack can achieve more than 90% ASR in most cases. The results, though, depend on the dataset and target label. The μ hyperparameter extensively affects the trigger optimization overhead. This might not be noticeable for regular datasets where it takes seconds for the trigger to be obtained, but for large datasets like HIGGS (where, in our worst-case report, just the regular clean training of Saint might take hours), it is highly evident. As we show in Section B, μ also affects ASR for datasets with fewer samples (e.g., BM). With higher values for μ, more samples distant to the target class manifold get involved in the perturbation optimization process, thus making it more general and effective. This makes it easier for the perturbation to shift the distribution toward the target class.

On the other hand, bigger μ increases the chance of backdoor detection, as we see later. The only exception we detected in experiments was the results for TabNet on the Credit Card dataset on target label 1 (see Figure 7). Although this did not happen with other models, we assume such results come from the dataset being heavily unbalanced (only around 0.1% of samples in the whole dataset belong to class number 1), making it difficult for a universal perturbation to perform perfectly on all random samples.

[10]Full experiments results, including the effect of choosing different μ values, are provided in Appendix B.

TABLE IV: CDA and ASR (CDA/ASR) for $\mu = 1$.

Dataset	Model	Target Label	ϵ			
			0.01	0.02	0.05	0.1
ACI	FTT	0	86.46/96.85	86.09/97.10	86.20/98.17	85.94/98.96
		1	86.34/98.50	86.35/99.62	85.94/99.77	86.10/100.0
	Saint	0	85.91/90.99	86.20/98.11	85.55/99.55	85.95/99.91
		1	85.95/92.35	85.74/93.38	85.51/78.84	85.52/95.55
	TabNet	0	85.67/97.39	85.40/97.62	84.83/96.53	84.25/98.42
		1	85.37/82.60	86.04/97.11	85.06/100.0	84.78/98.27
	XGBoost	0	85.09/99.11	85.26/99.05	85.24/99.74	85.32/99.91
		1	85.41/98.50	85.20/99.51	85.28/99.91	85.44/99.94
BM	FTT	0	83.70/87.33	86.52/95.75	86.25/99.96	86.30/100.0
		1	86.61/98.52	86.43/99.91	86.57/100.0	85.49/100.0
	Saint	0	85.09/96.33	84.51/98.88	84.68/100.0	85.85/99.96
		1	85.36/10.00	85.85/100.0	85.31/100.0	84.51/100.0
	TabNet	0	83.88/91.27	84.06/89.57	83.48/96.78	83.16/98.30
		1	83.74/84.73	82.22/88.22	82.80/97.49	79.94/96.95
	XGBoost	0	84.68/93.55	84.51/95.39	84.73/97.85	84.55/99.46
		1	84.86/95.21	84.51/97.27	84.95/99.24	84.55/99.96
CovType	FTT	0	95.76/99.98	96.23/100.0	96.21/100.0	95.90/100.0
		1	96.30/100.0	96.28/100.0	96.23/100.0	96.17/100.0
		2	96.31/99.99	96.15/100.0	96.29/100.0	96.03/100.0
		3	96.27/99.99	96.23/99.99	96.21/100.0	96.15/100.0
		4	96.16/99.98	96.20/100.0	96.15/100.0	96.03/100.0
		5	96.30/100.0	96.20/100.0	96.01/100.0	96.15/100.0
		6	97.12/99.99	96.86/100.0	97.09/100.0	97.01/100.0
	Saint	0	96.76/100.0	96.88/99.96	96.83/100.0	96.58/99.99
		1	96.81/99.98	96.73/99.97	96.79/99.70	96.65/100.0
		2	96.83/99.98	96.75/99.85	96.74/100.0	96.61/99.91
		3	96.83/99.97	96.84/99.84	96.77/99.85	96.49/99.87
		4	96.85/99.92	96.85/99.99	96.72/100.0	96.60/97.64
		5	96.93/99.97	96.94/100.0	96.89/99.98	96.54/99.95
		6	98.18/99.97	98.14/99.96	98.16/100.0	97.60/99.99
	TabNet	0	94.13/99.93	94.72/99.97	94.36/100.0	93.73/100.0
		1	94.10/99.90	94.54/99.99	94.37/100.0	94.38/100.0
		2	93.32/99.97	92.80/100.0	94.47/100.0	94.66/100.0
		3	94.53/99.95	91.07/100.0	94.21/100.0	94.22/99.98
		4	92.18/99.88	94.39/99.99	94.59/100.0	93.32/100.0
		5	93.20/99.97	94.75/99.91	94.70/100.0	94.75/100.0
		6	95.24/99.96	95.64/99.96	95.41/99.99	95.58/100.0
	XGBoost	0	96.51/99.98	96.49/100.0	96.44/100.0	96.32/100.0
		1	96.50/100.0	96.51/100.0	96.42/100.0	96.38/100.0
		2	96.55/99.99	96.49/100.0	96.42/100.0	96.38/100.0
		3	96.52/100.0	96.47/100.0	96.47/100.0	96.35/100.0
		4	96.51/99.99	96.47/99.99	96.46/100.0	96.40/100.0
		5	96.54/99.99	96.54/100.0	96.43/100.0	96.39/100.0
		6	98.73/99.99	98.69/100.0	98.71/100.0	98.56/100.0
Credit	FTT	0	99.95/99.86	99.95/99.86	99.92/99.86	99.94/99.87
		1	99.95/99.99	99.95/99.98	99.95/99.99	99.95/99.98
	Saint	0	99.95/99.87	99.96/99.85	99.96/99.86	99.96/99.86
		1	99.95/99.90	99.94/99.87	99.95/99.99	99.96/99.98
	TabNet	0	99.85/99.98	99.93/99.85	99.83/100.0	99.92/99.82
		1	99.93/99.71	96.93/95.15	99.93/99.95	99.91/86.18
	XGBoost	0	99.96/99.87	99.96/99.87	99.96/99.88	99.95/99.90
		1	99.96/99.93	99.96/99.97	99.96/99.95	99.96/99.99
Higgs	FTT	0	83.98/99.99	83.95/100.0	83.87/100.0	83.59/100.0
		1	83.89/100.0	83.96/100.0	83.95/100.0	83.81/100.0
	Saint	0	80.70/95.00	80.74/99.13	80.70/99.35	80.63/100.0
		1	80.67/94.94	80.64/99.95	80.59/95.58	80.48/99.95
	TabNet	0	77.70/99.98	78.47/100.0	78.18/100.0	78.49/100.0
		1	78.21/100.0	78.48/100.0	78.00/99.95	78.30/100.0
	XGBoost	0	77.38/100.0	77.35/100.0	77.36/100.0	77.35/100.0
		1	77.41/100.0	77.34/100.0	77.37/100.0	77.35/100.0

Another interesting observation is the results of a black box attack. We can notice that CatBack performs the same or even better in most cases on XGBoost than other models like TabNet. These observations suggest that CatBack can also be a very powerful threat in scenarios in which the poisoned dataset is provided to a third party (e.g., outsourced training), even though the attacker is unaware of the target model. The detailed results for all experiments regarding CatBack are provided in Figures 5 to 9 and discussed in Appendix B.

E. Baselines

We have also compared our attack with two baseline attacks. First, we tested Badnets [16]. To transfer Badnets to the tabular data, we applied the equivalent of an 8×8 square trigger in an MNIST image on each poisoned sample. In particular, the perturbation is 0.08% of all features (MNIST images are 28×28 pixels so $(8 \times 8)/(28 \times 28)$ = 0.08). We chose a random value within the valid range for each feature for the trigger. We also ran experiments with Tabdoor [12] using the in-bounds method. We did not try the out-of-bounds attack since the trigger uses values that are 10% higher than the maximum value of the poisoned features, which can be easily spotted as outliers. The same is true for the approach used in DBA, where the authors, to create a poisoned sample, changed the values in 6 features of the LOAN dataset using values larger than the maximum value for each of these features.

The results from these experiments are shown in Table V. We did not include the CDA in the tables, as in all cases the performance of the poisoned models was similar to the clean model's performance. According to the tables, our attack performs similarly to the baselines for CovType, Credit Card, and Higgs. In these cases, we see that the attack performance is almost perfect (e.g., close to 100%) in all datasets, meaning that these settings are the easiest to attack. For this reason, CatBack cannot show significantly better performance than the baselines. On the other hand, CatBack performs significantly better for ACI (up to 95% and 44% better performance than Tabdoor and Badnets, respectively) and BM (up to 71% and 16% better performance for Tabdoor and Badnets) datasets. Additionally, our trigger can be applied to any data type (numerical and categorical), while the previous attacks were applied only to numerical data.

TABLE V: Comparison of CatBack with baseline attacks. Catback Settings: $\mu = 1.0$, $\epsilon = 0.02$, target_label=1.

Dataset	Attack	FTT	SAINT	TabNet	XGBoost
ACI	Badnets	86.79	87.76	53.02	99.98
	Tabdoor	15.16	17.49	2.02	18.61
	CatBack	99.62	93.38	97.11	99.51
BM	Badnets	96.04	95.90	72.97	99.73
	Tabdoor	28.84	78.10	24.23	99.31
	CatBack	99.91	100.00	88.22	97.27
CovType	Badnets	100.00	100.00	100.00	100.00
	Tabdoor	99.96	99.98	99.77	100.00
	CatBack	100.00	99.97	99.99	100.00
Credit	Badnets	99.99	99.90	99.94	100.00
	Tabdoor	99.85	99.77	97.26	99.97
	CatBack	99.98	99.87	95.15	99.97
HIGGS	Badnets	100.00	100.00	99.98	100.00
	Tabdoor	100.00	93.63	99.99	100.00
	CatBack	100.00	99.95	100.00	100.00

F. Additional Threat Scenario

As we mentioned in Section V, the attacker reverts the poisoned dataset back to its original form after the poisoning is completed. In this way, the users can download and use the dataset in their own pipelines in its original form. $Revert(.)$ is an extra step of our attack, which could be omitted if our encoding method is used directly for the model training by

the users, making our attack more efficient. In this case, our encoding method should not affect the model's performance.

To show that this is a viable scenario, we compared the performance of models trained using our encoding with the performance of models trained using ordinal or categorical encoding. From the results shown in Table III, we see that our encoding method could be used as an alternative method for training models on tabular data without hurting the model's performance. This would make our attack easier, as in that case, the attacker should not revert the dataset back to its original form after the training completes. Additionally, our numerical encoding could facilitate input preprocessing in some cases (e.g., transformers like Saint and FTT) as it does not require an embedding layer as the model's input. Finally, our encoding does not result in an increase in the number of the table's columns, as this is common for methods like one-hot encoding, where each categorical column is converted to several binary columns.

G. Evaluation on Vertex AI

Cloud computing has become the backbone of modern computing infrastructure, with widespread adoption across industries. Organizations of all sizes increasingly rely on cloud platforms to power machine learning workflows. Google's Vertex AI, Amazon SageMaker, and Azure Machine Learning are the top three services used by almost every company in the market [31]. Among them, Vertex AI provides a dedicated framework for tabular data through AutoML [15]. AutoML has an automated pipeline and algorithms to train and deploy the model. While specific algorithmic defenses within AutoML are not publicly detailed, they include automated data preprocessing (data cleaning, feature engineering, and outlier detection) and robust training practices (input validation, model armor, monitoring for prediction drift, etc.) [32].

As a real-world use case for our black box scenario, we decided to test whether CatBack can be effective on Google AutoML and see whether AutoML's internal mechanisms can mitigate or prevent our attack. Table VI demonstrates the feasibility of such attacks in practice. In particular, in all cases, the attack is almost perfect ($> 97\%$), and the platform did not provide any warnings about the data or the backdoor. Thus, even though this platform includes tools that perform automated data processing, the attack has not been spotted. Such an attack can affect the performance of deployed models in production through these platforms (highly likely for many instances, like the financial sector).

TABLE VI: CatBack Performance on Google AutoML. The attack settings: $\mu = 1.0,\ \epsilon = 0.02$, target_label=1.

Dataset	BA (%)	CDA (%)	ASR (%)
ACI	87.31	87.21	98.11
Credit Card	99.95	99.97	100.0
CovType	96.19	97.33	100.0
HIGGS	76.97	76.63	100.0
BM	86.18	87.86	96.99

VII. Evaluation Against Defenses

In this section, we investigate how Catback performs against benchmark defenses. We have evaluated our attack against various defense types (including backdoor detection and backdoor removal methods). We also evaluated the poisoned dataset on popular outlier detection methods.

A. Spectral Signatures

To evaluate CatBack's stealthiness, we implemented Spectral Signatures [33] on the poisoned datasets. Using the default settings from the paper, we assume the detector already knows ϵ and pinpoints $1.5 \times \epsilon$ of samples with the highest score as suspicious to remove them from the dataset. We experiment with different μ and ϵ values. As expected, there is an inverse relationship between ϵ and the stealthiness of the attack, as lower poisoning rates bypass the detection mechanism. Nonetheless, we observe that μ is a more important hyperparameter.

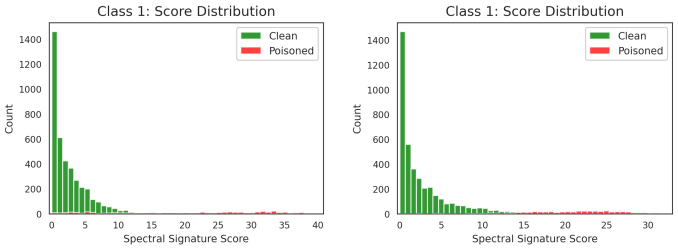
The higher μ values mean the universal perturbation is more general and should shift more random samples toward the decision manifold of the target class. On the other hand, smaller μ means choosing the samples closer to the decision manifold, which leads to a smaller perturbation needed for the shift. Thus, the poisoned samples are stealthier. For some datasets like BM, ASR is affected by μ value, especially when the poisoning rate is small (Figure 6). As the BM dataset is the smallest dataset in our experiments, we believe that there are fewer samples very close to the decision boundary, as it can be easier for the model to create a decision boundary in a sparse feature space. When we poison a dataset, we randomly select data from the training set, and by using a small poisoning rate in this case, we may select samples that are further away from the decision boundary. Thus, when μ is small and the trigger magnitude is also small, the trigger fails to alter the class of the poisoned data. However, if μ is large, the trigger magnitude is larger, which increases the ASR even if the samples are originally further away from the decision boundary. Thus, we expect it to leave a stronger effect on stealthiness as well.

We report the results for what be believe to be the least favorable condition for the attacker. Here, we chose the highest μ value of 1, $\epsilon = 0.02$ and target label 1. In this setting, the attack is mostly discoverable for the BM dataset (as we already expected since μ had the most influence on ASR for this dataset). Nonetheless, Spectral Signatures fails to remove or detect the poisoned samples in more than 75% cases. It is worth noting that with other settings, the defense failure rate is even higher, and for brevity, we only report the best defense performance. As an example, we see that by decreasing μ value to 0.5, it becomes more difficult to remove all poisoned samples for BM (Figure 2), and it entirely fails to detect poisoned samples for CovType with the FTT model (Figure 3).[11]

Moreover, in most cases, the score distributions for non-target classes are the same as the target class, which leads

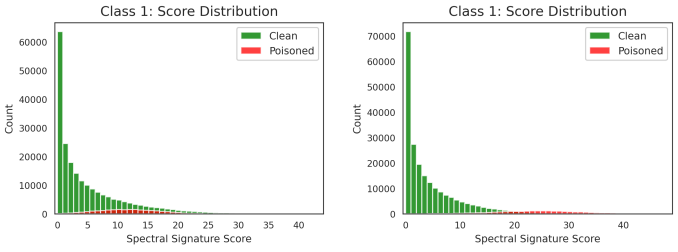
[11]Due to space limitations, we include the rest of the figures in the supplementary file in our repository.

to removing the same number of samples from every non-target class, resulting in a performance reduction of the model on clean data. Previous works that used simple heuristics for the backdoor trigger, such as [12], resulted in more obvious triggers that could be spotted by Spectral Signatures. On the contrary, Spectral Signatures defense is not successful against CatBack, indicating that our attack is stealthy.



(a) $\mu = 0.5$. (b) $\mu = 1$.

Fig. 2: Spectral Signatures: target class 1, Model: FTT, Dataset: BM, $\epsilon = 0.05$.



(a) $\mu = 0.5$. (b) $\mu = 1$.

Fig. 3: Spectral Signatures: target class 1, Model: FTT, Dataset: CovType, $\epsilon = 0.05$.

B. Neural Cleanse

We evaluated our attack against Neural Cleanse [34] to measure how detectable the target class is. We adapted the defense for tabular data but adhered to the same hyperparameters and settings from the original paper. Our results show that Neural Cleanse fails to detect the target class. All the anomaly scores we got were below 2 (the threshold), which indicates a complete failure of the method. In fact, most of the scores show that the retrieved mask was around $0.67 \times MAD$, similar to clean scores. Interestingly, this is the standard deviation of the Laplace distribution used in calculating the z-score values for outlier detection, which refers to the normal range of values around the Median (for a z-score to be an outlier, the value should be 3 times more than that). In Table VII, we show our results for target_label=1. The results for other target labels are omitted as they demonstrate similar behavior.

C. Beatrix

Beatrix [35] leverages the second-order statistics of internal feature representations to identify anomalies introduced by backdoor triggers. The mechanism operates by computing

TABLE VII: Anomaly Scores for Neural Cleanse. The attack settings: $\mu = 1.0$, $\epsilon = 0.02$, target_label=1.

Model	Dataset	Anomaly Score Backdoored/Clean
FTT	ACI	0.6745 / 0.6744
Saint	ACI	0.3699 / 0.6735
TabNet	ACI	0.6694 / 0.6731
FTT	BM	0.6745 / 0.6745
Saint	BM	0.6716 / 0.6730
TabNet	BM	0.6735 / 0.6744
FTT	CovType	0.0033 / 0.0091
Saint	CovType	1.5170 / 0.6742
TabNet	CovType	0.0046 / 1.9131
FTT	Credit Card	0.6745 / 0.6744
Saint	Credit Card	0.6744 / 0.6744
TabNet	Credit Card	0.6644 / 0.6856
FTT	HIGGS	0.6745 / 0.6844
Saint	HIGGS	0.6699 / 0.6838
TabNet	HIGGS	0.6740 / 0.6846

Gram matrices from intermediate activations, which capture the correlations among features. Deviations from the expected behavior of clean samples are quantified using the median and the median absolute deviation (MAD). These deviations are further aggregated through a Kernel Maximum Mean Discrepancy (KMMD) metric and standardized to yield an anomaly score, denoted as J^*. High values of J^* (i.e., $ln(J^*) > 2$) indicate significant discrepancies from normal feature distributions, suggesting the potential presence of a backdoor. We evaluated our attack against Beatrix. The results indicate that Beatrix fails to detect the presence of Catback. For most cases, we got $ln(J^*) \approx -0.39$, which indicates the proximity of the poisoned Gram Matrix to the median. Table VIII showcases our results for target_label = 1.

TABLE VIII: Score results for Beatrix. The attack settings: $\mu = 1.0$, $\epsilon = 0.02$, target_label=1.

Model	Dataset	KMMD	$\ln(J^*)$
FTT	ACI	14.6131	-0.3938
Saint	ACI	74.0251	-0.3938
TabNet	ACI	43.0890	-0.3939
FTT	BM	19.5594	-0.3938
Saint	BM	28.0811	-0.3938
TabNet	BM	51.3612	-0.3938
FTT	CovType	3.7468	-0.3938
Saint	CovType	13.2158	-0.0416
TabNet	CovType	24.1161	0.1113
FTT	Credit Card	32.1099	-0.3938
Saint	Credit Card	0.0001	-9.2103
TabNet	Credit Card	39.4631	-0.3938
FTT	HIGGS	18.4841	-0.3938
Saint	HIGGS	24.2661	-0.3938
TabNet	HIGGS	101.8080	-0.3938

Overall, the results from both Beatrix and Neural Cleanse highlight the high stealthiness of Catback, underscoring the importance of exercising caution when trusting third parties with trained models or ready-to-train datasets.

D. Fine Pruning

To measure how robust our attack is, we evaluated it against Fine Pruning [36]. We pruned all feed-forward layers (FNN) within the transformers [37], [38] (SAINT and FTT) for up to 90% of neurons and fine-tuned the model for 5 epochs. For SAINT, we also pruned the penultimate layer to make the pruning stronger. Our results show that overall, fine pruning is not effective enough to prevent the attack from fully functioning. For SAINT, although it can restore the accuracy after fine tuning [39], ASR also remains considerably high. For FTT, however, Fine Pruning can successfully remove the attack for binary classification tasks with a smaller number of samples. Nevertheless, it fails to remove the backdoor if the dataset is large (e.g., Higgs). We hypothesize that this is because of a higher number of neurons in FNN being involved in learning the representations when facing more complicated datasets. With simpler datasets, few neurons can learn the binary manifold like a logistic regression task. This can also be observed when we observe the results for a multiclass dataset (Covtype), where the method fails in both preventing the attack and keeping the CDA high. To confirm this, we conducted extra experiments on another multiclass dataset: Poker Hand[12], consisting of 10 classes. First, we conduct the attack and evaluate its performance. Table X demonstrates the CatBack results on Poker Hand, with a successful high ASR. Then we perform the Fine Pruning whose results are shown in Table XI, and as we see, it likewise fails to mitigate the attack thoroughly.

TABLE IX: Fine Pruning Results for the pruning_rate = 0.9. The attack settings: $\mu = 1.0,\ \epsilon = 0.02$, target_label=1.

Model	Dataset	FP-CDA (%)	FP-ASR (%)
Saint	ACI	87.90	84.34
FTT	ACI	85.28	25.43
Saint	BM	86.94	99.96
FTT	BM	80.92	23.29
Saint	CovType	90.55	65.50
FTT	CovType	79.34	98.74
FTT	Credit Card	99.95	0.15
Saint	Credit Card	99.95	91.49
FTT	HIGGS	77.24	63.99
Saint	HIGGS	78.89	77.67

TABLE X: Attack Results on Poker Dataset. The attack settings: $\mu = 1.0,\ \epsilon = 0.02$, target_label=1.

Model	BA (%)	CDA (%)	ASR (%)
Saint	99.78	99.99	100
FTT	99.79	99.96	99.90
TabNet	99.14	94.62	97.51
XGBoost	96.82	99.63	100

E. Evaluation with the Outlier Detection Algorithms

We have also investigated whether the poisoned samples created by CatBack can be detected using outlier detection methods. For this, we chose the two most popular outlier detection algorithms [40]: DBSCAN and Isolation Forest, which are commonly used in real-world applications, like fraud detection in finance or network intrusion detection [41]. We applied these two methods to our poisoned datasets, and both failed to detect and separate the poisoned and clean samples correctly. As a showcase, we report two of our results for the ACI (binary) and CovType (multiclass) datasets. Table XII demonstrates the confusion matrices for Isolation Forest and DBSCAN, respectively. As we can observe, DBSCAN fails to achieve an F1-score of higher than 0.02 while Isolation Forest performs even worse by reaching a precision and recall of 0, meaning not a single poisoned sample can be detected. We omit the results for the remaining datasets as they showed a similar behavior.

[12]https://www.kaggle.com/datasets/hosseinah1/poker-game-dataset/data

TABLE XI: Fine Pruning Results on Poker Dataset. The attack settings: $\mu = 1.0,\ \epsilon = 0.02$, target_label=1.

Model	FP-CDA (%)	FP-ASR (%)
Saint	99.96	66.41
FTT	67.55	74.91

TABLE XII: Confusion matrices for poisoned-sample detection (rows: Actual, columns: Predicted; N=Normal, P=Poisoned). Parameters: $\mu = 1.0$, $\epsilon = 0.02$, *target_label* $= 1$.

Isolation Forest

ACI

Act. \ *Pred.*	N	P
N	25008	520
P	520	0

$F_1 = 0$

Covtype

Act. \ *Pred.*	N	P
N	446217	9296
P	9296	0

$F_1 = 0$

DBSCAN

ACI

Act. \ *Pred.*	N	P
N	11324	14204
P	426	94

$F_1 \approx 0.0127$

Covtype

Act. \ *Pred.*	N	P
N	392434	63079
P	7173	2123

$F_1 \approx 0.0143$

VIII. Ablation Studies

A. Partial Access to Training Data

In our main threat model, we assume the attacker has full access to the complete training dataset. This might help the attacker observe more samples to calculate the trigger. To evaluate CatBack's effectiveness in stricter conditions, we assume a restricted threat model in which the attacker has a partial auxiliary dataset. We left only 10% of the dataset in the hands of the attacker and repeated the experiments. The results in Table XIII show that for datasets with a sufficient number of samples, the ASR remains almost the same, indicating that the trigger is still being calculated successfully. For smaller datasets like ACI and BM, with fewer samples, there is a noticeable drop in ASR; yet, in most cases, the attack remains effective.

B. Frequency Mappings

Our Conv$(\cdot)$ mapping is strictly frequency-based. Each category is assigned a real value in $[0, 1]$ according to how often it occurs in the dataset (plus a small Δr to break ties). Thus, there is always a one-to-one mapping between frequency-mapped categories and their corresponding values in one-hot

TABLE XIII: ASR when the attacker controls only 10% of the data. The attack settings: $\mu = 1.0,\ \epsilon = 0.02$, target_label=1.

Model	Dataset				
	ACI	BM	CovType	Credit Card	HIGGS
FTT	90.63	60.55	100	99.42	100
Saint	93.35	81.33	100	99.65	100
TabNet	88.15	52.44	99.99	99.39	100
XGBoost	95.35	82.69	99.99	99.87	99.51

encoding or any other type of representation. This assures that our method preserves the same semantics needed by neural networks to learn and distinguish between categories of each feature. However, our method might induce an extra semantic to the domain knowledge for neural networks by representing the occurrence of each category when the network learns its relations with other co-occurred values in other features.

Another important aspect to notice is that when applying the trigger, the frequency of the categories might change from the original encoding. However, in our attack, the attacker reverts back the dataset to its original values, so the defender would not get suspicious of inconsistency between the frequency mapping values and the frequency of categories in the contaminated dataset, except by accessing the lookup table. Here, the lookup table serves as a secret key, owned and used only by the attacker. One more interesting question is what happens if the user wants to use the raw frequency mappings as the default encoding. Then, the user would still not be able to observe this inconsistency, as the attacker conversion is only for poisoning purposes. After poisoning is done, the poisoned dataset can be converted again to another frequency mapping representation, and the new lookup table is given to the user, but the user never knows there was another lookup table owned by the attacker for poisoning conversion. Even in the worst-case scenario, if the user can obtain the lookup table, the frequency mappings of the clean and poisoned datasets are close, barely raising suspicion in the user. As an example, we demonstrate the Race and Relationship frequency mappings in the ACI dataset in Figure 4.

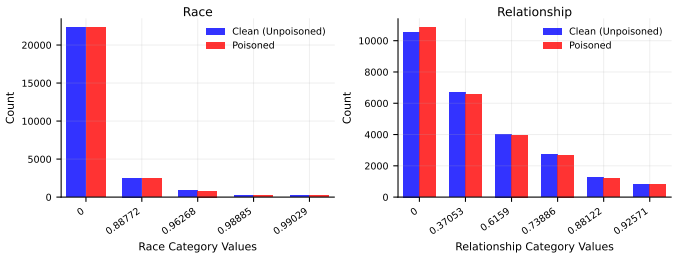


Fig. 4: Race and Relationship values before and after poisoning in the ACI dataset.

C. Performance Under Distribution Shifts

To conduct further analysis, we designed two experiments for observing attack performance under distribution shifts:

1) In our first experiment, we decided to randomly change the distribution of the entire test set. For doing this, we iterate through all samples, and for each sample, we randomly pick 25% of all features, whether categorical or numerical (thus, for each sample, the chosen features are different). Then, for each chosen feature, we draw a new value using a Gaussian distribution (within the valid range) and substitute the new value with the old one. Then, we test the model on this newly sampled dataset (containing a new distribution). Table XIV shows that the attack is successful under the presumed condition.

TABLE XIV: Distribution Shift Results on the ACI and Covtype datasets. Attack settings: $\mu = 1.0,\ \epsilon = 0.02$, target_label=1.

Dataset	Model	CDA (%)	ASR (%)
ACI	Saint	69.80	96.26
	Tabnet	68.75	90.78
Covtype	Saint	49.70	91.61
	Tabnet	63.07	98.68

2) In our second experiment, we implemented concept drift with retraining. Since there should be a pattern in concept drift for the model to learn, we first choose 25% of the columns randomly from the whole dataset (unlike the previous experiment, this time the chosen columns are fixed for all samples). Then, for each of these columns, if they are categorical, we simply replace them with the least common value for that category in the entire dataset. If the column is numerical, we perform a covariate shift on the values of that column.
To simulate covariate shift, we apply a sudden distributional change to a subset of samples in selected feature columns. Let $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ denote the training and test datasets respectively, and let $X \in \mathbb{R}^{n \times d}$ represent the feature matrix with n samples and d features.
Let $j \in \{1, \ldots, d\}$ denote the index of a selected numerical feature. We denote the values of this feature in the training set as:

$$X^{(j)}_{\text{train}} = \left\{ x^{(j)}_i \right\}_{i=1}^{n_{\text{train}}}.$$

We compute the empirical mean and standard deviation of the feature in the training set:

$$\mu^{(j)}_{\text{train}} = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} x^{(j)}_i, \quad \sigma^{(j)}_{\text{train}} = \sqrt{\frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \left(x^{(j)}_i - \mu^{(j)}_{\text{train}} \right)^2}.$$

We then define transformation parameters for drift:
* A random mean shift factor $\alpha \sim \mathcal{U}(0.1, 0.3)$
* A random scale factor $\beta \sim \mathcal{U}(0.1, 0.3)$
Using these, we compute:

$$\Delta\mu = \alpha \cdot \sigma^{(j)}_{\text{train}}, \quad \gamma = 1 + \beta.$$

The transformation applied to each selected test sample $x^{(j)}$ is:

$$x^{(j)}_{\text{drifted}} = \Big(x^{(j)} + \Delta\mu\Big) \cdot \gamma.$$

This results in a shifted and scaled version of the original feature, effectively altering both the first and second moments of the feature distribution. The operation introduces a sudden covariate shift.

We apply the concept drift on the entire test set and 25% of the train samples randomly. Then we perform a complete retraining of the model on the drifted training set and evaluate the attack on the test set. The results are demonstrated in Table XV. Results suggest that concept drift and retraining together can impact the ASR; however, this highly depends on the model.

As a takeaway, we can regard this as a possible solution to mitigate the attack efficacy. Thus, by monitoring and responding to distribution shifts (scheduled retaining), practitioners can remove implanted triggers over time.

TABLE XV: Concept Drift (with retraining) Results on the ACI and Covtype datasets. Attack settings: $\mu = 1.0$, $\epsilon = 0.02$, target_label=1.

Dataset	Model	CDA (%)	ASR (%)
ACI	Saint	84.08	99.43
	Tabnet	85.23	19.38
Covtype	Saint	92.32	91.5
	Tabnet	92.24	24.29

IX. Limitations and Discussion

Imperceptibility refers to how unnoticeable the trigger is to a human observer or simple statistical checks. Its focus is on the input space (features), and the goal is to make the trigger visually or statistically subtle so it does not raise suspicion [42]. Stealthiness, however, usually refers to how difficult the attack is to detect by automated defenses or manual inspections, not just in the input, but also in model behavior, the training process, or data patterns. Its focus is on the training process, model internals, and activation patterns [43]. Its goal is to make the attack difficult for defenders to detect using anomaly detection, activation clustering, or mitigating techniques. Thus, the attack could be perceptible, yet highly stealthy, hiding from all detection methods [43].

A variety of domain-specific metrics have been proposed to quantify imperceptibility in non-tabular settings:

- **Image Domain:** Metrics based on L_p norms (e.g., L_2, L_∞) measure the magnitude of pixel perturbations, while perceptual similarity measures such as Structural Similarity Index (SSIM) and Learned Perceptual Image Patch Similarity (LPIPS) model human visual perception to assess perturbation visibility [44], [45], [46]. Frequency-domain metrics (e.g., DCT distortion) further ensure that the trigger does not introduce atypical spectral components.
- **Text Domain:** Metrics such as word-embedding distance and perplexity change quantify the semantic drift introduced by triggers. Approaches like BLEU and ROUGE scores, alongside language model-based fluency measures, evaluate whether inserted or substituted tokens remain coherent in the sentence context.
- **Voice Domain:** Imperceptibility is measured by signal-to-noise ratio (SNR), Mel-cepstral distortion (MCD), and perceptual evaluation of speech quality (PESQ) to ensure that the backdoor trigger (e.g., audio watermark) remains inaudible or indistinguishable from natural background noise.

One of the limitations we face during the current study is the lack of an imperceptibility metric in the tabular domain. Although we design our backdoor to be stealthy, there is no accepted measure in the community so that we can calculate how imperceptible the backdoor trigger is in tabular entry. Though designing and proposing a new metric specific to tabular data is out of the scope of this work, we try to illustrate some aspects of a good imperceptibility metric tailored for tabular data, and hope this can provide a good insight for future studies. Similar to BlEU or SSIM, we assume that there is a reference sample for the given synthetic sample (here, we consider the original clean sample as the reference and its corresponding backdoored sample as the synthetic). Let us denote the synthetic row as

$$\mathbf{x} = (x_1, x_2, \ldots, x_F)$$

against a real reference row:

$$\mathbf{y} = (y_1, y_2, \ldots, y_F)$$

We assume that there is a tabular imperceptibility score denoted as TIS. We believe that TIS should be a function that is dependent on two important factors:

$$TIS = f(FLS, IF).$$

- Feature-Level Similarity (FLS): A per-feature score that measures how similar each candidate feature is to its reference counterpart.
- Interaction Fidelity (IF): A penalty/bonus component that captures the similarity of pairwise (or higher-order) interactions among features.

We conjecture that a good TIS should have these values calculated in its process, either explicitly or implicitly. FLS calculation might be just a distance measure (e.g., L_2 distance). However, calculating IF might be more complicated and vary from dataset to dataset. For example, we assume there is a tabular dataset consisting of different columns of patient data. Age and BP (bacterial peritonitis) columns might be highly correlated with each other. However, this is something that could be known only by specialists or researchers who obtained the correlation function between these features (e.g., through linear regression). Thus, formulating these correlation functions varies significantly between each dataset and may require the assistance of a specialist in that field. We leave this as an open question that might be an interesting topic for future studies.

X. Related Work

The first backdoor attacks were introduced in [16], [47]. In these works, the authors demonstrated that an attacker can insert a backdoor into a trained model just by altering a few training samples. A large number of different attacks have been introduced since then, most of them targeting computer vision [8], but also targeting different domains like graph neural networks [48], language models [49], or speech processing [50].

Only a few works studied backdoors on tabular data. [10] performed distributed backdoor attacks on federated learning. For a backdoor on tabular data, the authors performed a feature importance ranking and chose the 6 least important features. For each of those chosen features, they set their value to an outbound number larger than the maximum value in the dataset. [12] conducted a more comprehensive study on backdoors in tabular data, resulting in the Tabdoor attack. Contrary to [10], the authors observed less of a relation between feature importance and attack success rate. Since the authors investigated more datasets, they suggested that using the most important features results in a slightly higher attack success rate. To improve over [10], they proposed two different attacks: Out-of-bounds and In-bounds attacks. For Out-of-bounds, they followed the same idea as in [10], but with fewer features. The authors tried 1, 2, and 3 features and set their values to 10% higher than the maximum values of each feature in the whole dataset. Since the Out-of-bounds attacks can be easily discovered as outliers, [12] performed the In-bounds attack as their main contribution. For In-bounds, they chose the top 3 most important features within the dataset and set their values to the most common value for that feature. As presented in previous sections, CatBack works significantly better than Tabdoor, especially for more complex datasets.

[51] performs a data-free backdoor attack. They used an approach similar to teacher-student type of training to induce the trigger pattern in the victim model without accessing the dataset. For this, they collect a substitute dataset (which may be irrelevant to the main dataset) and try to implant the trigger value inside the substitute dataset. For tabular data, they chose ACI as their main task dataset and CovType as the substitute. They selected two features of ACI (fnlwgt, sex) with a fixed value and implanted them in the Covtype (with 20% poisoning rate), while changing the label to target, and then fine-tuned the victim model. Unfortunately, due to a significantly different setup from ours (different models, types of attacks, and different poisoning rates), direct comparison with our work is not possible. [13] performs a backdoor on Binary click-through rate (CTR) datasets using a Factorization model (Deep-FM). To do so, they conducted a feature importance ranking based on the Deep-FM embedding layer (they consider features with higher average weight values to be more important). Then, they select the top 4 most important features and set them to the least frequent value. However, the authors did not provide the code, and we could not reproduce the results. For this reason, we did not consider this attack in our evaluation.

XI. Conclusions and Future Work

In this work, we have introduced a targeted backdoor attack on tabular data by crafting a universal trigger pattern that can be added to any input within the domain. By leveraging frequency-based floating-point mapping for categorical values, we unify all feature types into a continuous space and enable a single, elastic-net-regularized perturbation to reliably shift any sample toward an adversarial target. By enabling the trigger to involve any column of a table, including categorical features, we broaden the attack surface.

We evaluated CatBack across five widely used benchmark datasets and four distinct models, achieving near-perfect attack success rates (up to 100%) with negligible impact on clean-data accuracy, under constraints such as 1% poisoning rate or only 10% auxiliary data access. Moreover, our evaluation against state-of-the-art defenses shows that CatBack remains largely undefeated, and it readily transfers to real-world cloud services, such as Google Vertex AI.

These findings underscore the need for future studies to rethink trust assumptions around tabular pipelines, including the development of custom defenses for tabular data, such as preprocessing-agnostic detection strategies, and incorporating robust certification for mixed-type inputs. Future work could extend CatBack to dynamic, sample-specific triggers, investigate defenses based on causal feature attribution, and evaluate these threats in federated or privacy-preserving settings. Ultimately, securing mission-critical domains such as finance and healthcare will require holistic frameworks that treat tabular ML with the same adversarial scrutiny long applied to vision and language.

XII. Ethical Considerations

Our work investigates backdoor attacks for tabular data, highlighting how efficient transformation of input data can lead to high-performing and stealthy backdoors. While our goal is to increase awareness about this attack vector, considering the prevalence of tabular data in real-world applications, we acknowledge that disclosing such vulnerabilities could potentially be exploited for malicious purposes. All experiments conducted in this research were designed to avoid exposing sensitive data or causing real-world harm. Evaluations were performed in controlled environments, using publicly available datasets. Despite these precautions, we acknowledge that revealing any new attack carries some risk. Still, we believe that transparent disclosure of such vulnerabilities, combined with responsible communication, benefits the community.

References

[1] D. Ulmer, L. Meijerink, and G. Cinà, "Trust issues: Uncertainty estimation does not enable reliable ood detection on medical tabular data," in *Machine Learning for Health*. PMLR, 2020, pp. 341–354.

[2] J. M. Clements, D. Xu, N. Yousefi, and D. Efimov, "Sequential deep learning for credit risk monitoring with tabular financial data," *arXiv preprint arXiv:2012.15330*, 2020.

[3] F. Cartella, O. Anunciacao, Y. Funabiki, D. Yamaguchi, T. Akishita, and O. Elshocht, "Adversarial attacks for tabular data: Application to fraud detection and imbalanced data," *arXiv preprint arXiv:2101.08030*, 2021.

[4] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko, "Revisiting deep learning models for tabular data," *Advances in Neural Information Processing Systems*, vol. 34, pp. 18 932–18 943, 2021.

[5] G. Somepalli, M. Goldblum, A. Schwarzschild, C. B. Bruss, and T. Goldstein, "Saint: Improved neural networks for tabular data via row attention and contrastive pre-training," *arXiv preprint arXiv:2106.01342*, 2021.

[6] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" *Advances in neural information processing systems*, vol. 35, pp. 507–520, 2022.

[7] B. Liu, M. Ding, S. Shaham, W. Rahayu, F. Farokhi, and Z. Lin, "When machine learning meets privacy: A survey and outlook," *ACM Computing Surveys (CSUR)*, vol. 54, no. 2, pp. 1–36, 2021.

[8] Y. Li, Y. Jiang, Z. Li, and S.-T. Xia, "Backdoor learning: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 1, pp. 5–22, 2022.

[9] B. Tajalli, S. Koffas, G. Abad, and S. Picek, "Elms under siege: A study on backdoor attacks on extreme learning machines," in *Proceedings of the 2024 Workshop on Artificial Intelligence and Security*, 2024, pp. 125–136.

[10] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "Dba: Distributed backdoor attacks against federated learning," in *International conference on learning representations*, 2019.

[11] B. Joe, Y. Park, J. Hamm, I. Shin, J. Lee *et al.*, "Exploiting missing value patterns for a backdoor attack on machine learning models of electronic health records: Development and validation study," *JMIR Medical Informatics*, vol. 10, no. 8, p. e38440, 2022.

[12] B. Pleiter, B. Tajalli, S. Koffas, G. Abad, J. Xu, M. Larson, and S. Picek, "Tabdoor: Backdoor vulnerabilities in transformer-based neural networks for tabular data," *arXiv preprint arXiv:2311.07550*, 2023.

[13] L. Meng, X. Gong, and Y. Chen, "Bad-fm: Backdoor attacks against factorization-machine based neural network for tabular data prediction," *Chinese Journal of Electronics*, vol. 33, no. 4, pp. 1077–1092, 2024.

[14] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep neural networks and tabular data: A survey," *IEEE transactions on neural networks and learning systems*, 2022.

[15] Google Cloud, "Train models with vertex ai (tabular data)," https://cloud.google.com/vertex-ai/docs/training-overview#tabular, 2025, accessed: 2025-04-23.

[16] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "Badnets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019.

[17] E. Bagdasaryan and V. Shmatikov, "Blind backdoors in deep learning models," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1505–1521.

[18] S. Hong, N. Carlini, and A. Kurakin, "Handcrafted backdoors in deep neural networks," *Advances in Neural Information Processing Systems*, vol. 35, pp. 8068–8080, 2022.

[19] G. Abad, J. Xu, S. Koffas, B. Tajalli, and S. Picek, "A systematic evaluation of backdoor trigger characteristics in image classification," *arXiv preprint arXiv:2302.01740*, 2023.

[20] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[21] 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.

[22] S. Ö. Arik and T. Pfister, "Tabnet: Attentive interpretable tabular learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, 2021, pp. 6679–6687.

[23] Amazon Web Services, "Built-in sagemaker ai algorithms for tabular data," https://docs.aws.amazon.com/sagemaker/latest/dg/algorithms-tabular.html, amazon SageMaker Developer Guide, accessed April 22, 2025.

[24] A. Dhurandhar, T. Pedapati, A. Balakrishnan, P.-Y. Chen, K. Shanmugam, and R. Puri, "Model agnostic contrastive explanations for classification models," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2024.

[25] R. Tibshirani, "Regression shrinkage and selection via the Lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[26] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[27] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 67, no. 2, pp. 301–320, 2005.

[28] Ł. Delong and A. Kozak, "The use of autoencoders for training neural networks with mixed categorical and numerical features," *ASTIN Bulletin: The Journal of the IAA*, vol. 53, no. 2, pp. 213–232, 2023.

[29] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[30] S. Koffas, B. Tajalli, J. Xu, M. Conti, and S. Picek, "A systematic evaluation of backdoor attacks in various domains," in *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing: Use Cases and Emerging Challenges*. Springer, 2023, pp. 519–552.

[31] F. Richter. (2024) Amazon and microsoft stay ahead in global cloud market. Accessed: 2025-04-24. [Online]. Available: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/

[32] Google Cloud, "Google cloud ai and security documentation," https://cloud.google.com/vertex-ai/docs/model-monitoring/monitor-explainable-ai, https://cloud.google.com/security-command-center/docs/model-armor-overview, https://cloud.google.com/security-command-center/docs/security-posture-overview, 2025, accessed: 2025-04-18.

[33] B. Tran, J. Li, and A. Madry, "Spectral signatures in backdoor attacks," *Advances in neural information processing systems*, vol. 31, 2018.

[34] 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.

[35] W. Ma, D. Wang, R. Sun, M. Xue, S. Wen, and Y. Xiang, "The "beatrix" resurrections: Robust backdoor detection via gram matrices," in *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/the-beatrix-resurrections-robust-backdoor-detection-via-gram-matrices/

[36] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *International symposium on research in attacks, intrusions, and defenses*. Springer, 2018, pp. 273–294.

[37] M. A. Gordon, K. Duh, and N. Andrews, "Compressing bert: Studying the effects of weight pruning on transfer learning," *arXiv preprint arXiv:2002.08307*, 2020.

[38] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, "What does bert look at? an analysis of bert's attention," *arXiv preprint arXiv:1906.04341*, 2019.

[39] F. Lagunas, E. Charlaix, V. Sanh, and A. M. Rush, "Block pruning for faster transformers," *arXiv preprint arXiv:2109.04838*, 2021.

[40] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial intelligence review*, vol. 22, pp. 85–126, 2004.

[41] M. T. R. Laskar, J. X. Huang, V. Smetana, C. Stewart, K. Pouw, A. An, S. Chan, and L. Liu, "Extending isolation forest for anomaly detection in big data via k-means," *ACM Transactions on Cyber-Physical Systems (TCPS)*, vol. 5, no. 4, pp. 1–26, 2021.

[42] Y. Qin, N. Carlini, G. Cottrell, I. Goodfellow, and C. Raffel, "Imperceptible, robust, and targeted adversarial examples for automatic speech recognition," in *International conference on machine learning*. PMLR, 2019, pp. 5231–5240.

[43] Z. Liu, F. Li, J. Lin, Z. Li, and B. Luo, "Hide and seek: on the stealthiness of attacks against deep learning systems," in *European Symposium on Research in Computer Security*. Springer, 2022, pp. 343–363.

[44] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[45] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 586–595.

[46] H. Wang, Z. Xiang, D. J. Miller, and G. Kesidis, "Mm-bd: Post-training detection of backdoor attacks with arbitrary backdoor pattern types using a maximum margin statistic," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 1994–2012.

[47] 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.
[48] Z. Xi, R. Pang, S. Ji, and T. Wang, "Graph backdoor," in *30th USENIX security symposium (USENIX Security 21)*, 2021, pp. 1523–1540.
[49] X. Chen, A. Salem, D. Chen, M. Backes, S. Ma, Q. Shen, Z. Wu, and Y. Zhang, "Badnl: Backdoor attacks against nlp models with semantic-preserving improvements," in *Proceedings of the 37th Annual Computer Security Applications Conference*, 2021, pp. 554–569.
[50] T. Zhai, Y. Li, Z. Zhang, B. Wu, Y. Jiang, and S.-T. Xia, "Backdoor attack against speaker verification," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 2560–2564.
[51] P. Lv, C. Yue, R. Liang, Y. Yang, S. Zhang, H. Ma, and K. Chen, "A data-free backdoor injection approach in neural networks," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 2671–2688.

APPENDIX A
EXAMPLE OF OUR ENCODING AND ATTACK ALGORITHM

Here, we demonstrate our encoding method through an example. To this end, we consider a categorical feature j with the following category counts:

Category v_{jl}	**Count** c_{jl}
A	50
B	50
C	30
D	20

TABLE XVI: Category Counts for Feature j

Primary Mapping:

$$r_{jA} = \tfrac{50-50}{50-1} = 0.0000, \qquad r_{jB} = \tfrac{50-50}{50-1} = 0.0000$$
$$r_{jC} = \tfrac{50-30}{50-1} \approx 0.4082, \qquad r_{jD} = \tfrac{50-20}{50-1} \approx 0.6122$$

Determine Δr:

- $\Delta r_{\text{min}} = 0.204$
- Identify the largest single decimal component in Δr_{min}:
  - $\Delta r_{\text{min}} = 0.204$ has the first non-zero decimal at the first decimal place ($p = 1$).
- Set $\Delta r = 10^{-(p+1)} = 10^{-2} = 0.01$.

Apply Hierarchical Mapping:

$$r'_{jA} = 0.0000 + (1-1) \times 0.01 = 0.0000$$
$$r'_{jB} = 0.0000 + (2-1) \times 0.01 = 0.0100$$
$$r'_{jC} = 0.4082$$
$$r'_{jD} = 0.6122$$

Result:

Category v_{jl}	**Original** r_{jl}	**Updated** r'_{jl}
A	0.0000	0.0000
B	0.0000	0.0100
C	0.4082	0.4082
D	0.6122	0.6122

TABLE XVII: Updated Numerical Representation After Hierarchical Mapping

Lookup Table:

Referring to Table XVII, the lookup table T_j for feature j is constructed as:

When an r'_{jl} value of 0.0100 is encountered during reverse mapping, the corresponding category retrieved from T_j is B.

Numerical Value r'_{jl}	**Category** v_{jl}
0.0000	A
0.0100	B
0.4082	C
0.6122	D

TABLE XVIII: Lookup Table T_j for Feature j

Algorithm 1 CatBack: Universal Backdoor for Tabular Data

Require: $D_{\text{orig}}, t, \mathcal{F}, \mu, \beta, \lambda, \epsilon$
Ensure: Backdoored model $\mathcal{F}'$
$D \leftarrow \text{Conv}(D_{\text{orig}})$ ▷ frequency-based encoding & tie-resolution
Train $\mathcal{F}$ on D
$D_{\text{nt}} \leftarrow \{(x, y) \in D \mid y \neq t\}$
for all $(x_i, y_i) \in D_{\text{nt}}$ **do**
  $s_i \leftarrow f_t(x_i)$ ▷ softmax score for class t
end for
Sort D_{nt} by s_i descending
$D_{\text{picked}} \leftarrow$ top $\mu\,|D_{\text{nt}}|$ samples
Initialize $\delta \leftarrow \mathbf{0} \in \mathbb{R}^d$
repeat
  for all $(x_i, y_i) \in D_{\text{picked}}$ **do**
    $\hat{x}_i \leftarrow \text{clip}(x_i + \delta)$
    $\ell_i \leftarrow -\log f_t(\hat{x}_i) + \beta\|\hat{x}_i - \text{Mode}(D)\|_1 + \lambda\|\hat{x}_i - \text{Mode}(D)\|_2^2$
  end for
  $\delta \leftarrow \delta - \eta\,\nabla_\delta\big(\frac{1}{|D_{\text{picked}}|}\sum_i \ell_i\big)$
  Round categorical dimensions of δ to nearest valid r'_{jl}
until convergence
Randomly select $D_{\text{sel}} \subset D$, $|D_{\text{sel}}| = \epsilon N$
for all $x_i \in D_{\text{sel}}$ **do**
  $\hat{x}_i \leftarrow \text{clip}(x_i + \delta), \quad \hat{y}_i \leftarrow t$
end for
$D' \leftarrow \text{Revert}\big((D \setminus D_{\text{sel}}) \cup \{(\hat{x}_i, \hat{y}_i)\}\big)$
Preprocess D' (OHE/embeddings/ordinal), train $\mathcal{F}'$ on D' **return** $\mathcal{F}'$

APPENDIX B
DETAILED ATTACK RESULTS

In this section, we have included the results from our experiments for all the hyperparameters we evaluated. In particular, in Figures 5 to 9, we show ASR (solid lines) and CDA (dotted lines) for our experiments for all datasets, models, target labels, and μ values. In Figure 5, we see that our attack achieves a performance of more than 80% in most cases. Additionally, the performance slightly increases as we increase the poisoning rate. In this case, both the μ and the target label do not significantly influence CatBack's performance. In Figure 6, however, the target label can affect the attack performance significantly in some cases. As BM is a relatively small dataset, we believe that the small imbalance of the two classes could lead to such differences. Class 0 corresponds to 53% of the dataset, and for this reason, the attack is slightly more effective when the target class is 0. Additionally, in Figure 6, when the FTT is used, we see that μ also affects CatBack's performance for both target labels, with $\mu = 1$ leading to the best performance due to bigger perturbation values. Similar results were seen for the rest of the datasets as shown in Figures 7, 8, and 9.

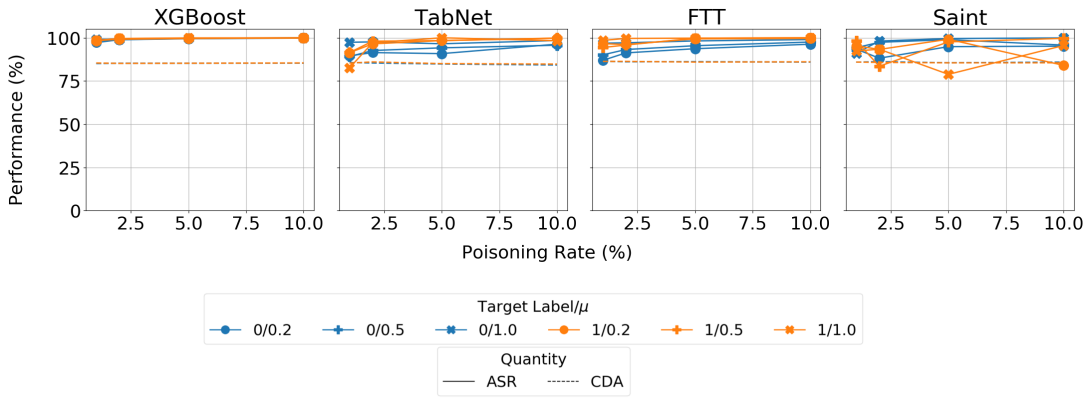


Fig. 5: CatBack's ASR and CDA vs. the poisoning rate for different μ using the ACI dataset.

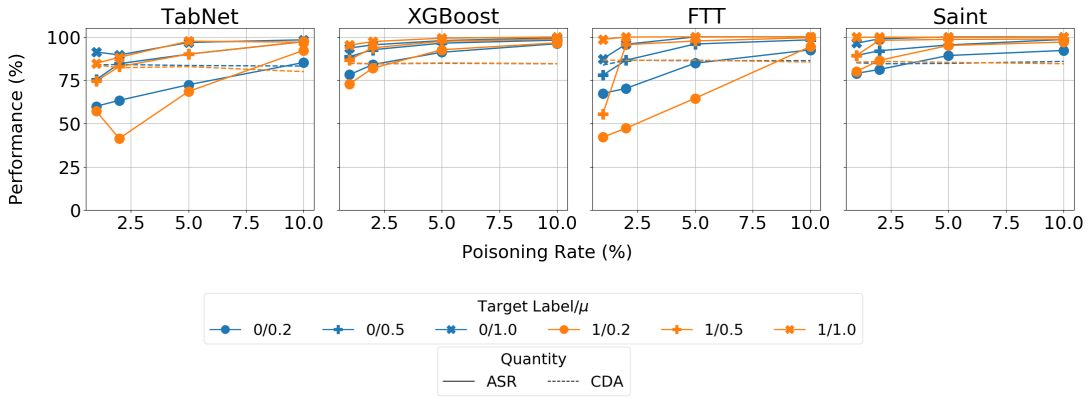


Fig. 6: CatBack's ASR and CDA vs. the poisoning rate for different μ using the BM dataset.

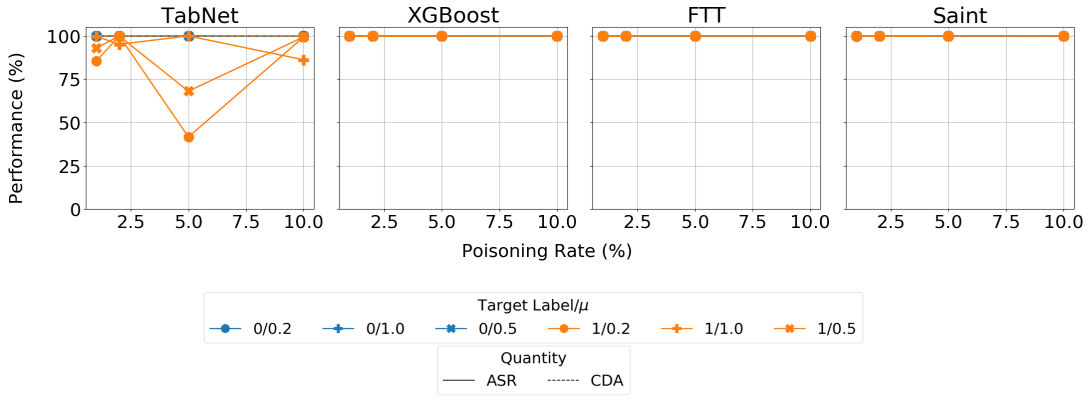


Fig. 7: CatBack's ASR and CDA vs. the poisoning rate for different μ using the Credit Card dataset.

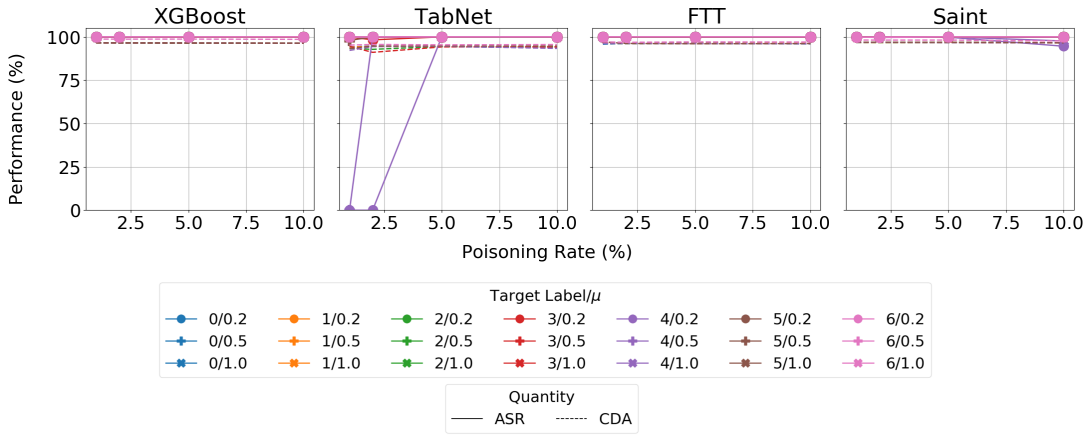


Fig. 8: CatBack's ASR and CDA vs. the poisoning rate for different μ using the Forest Cover Type dataset.

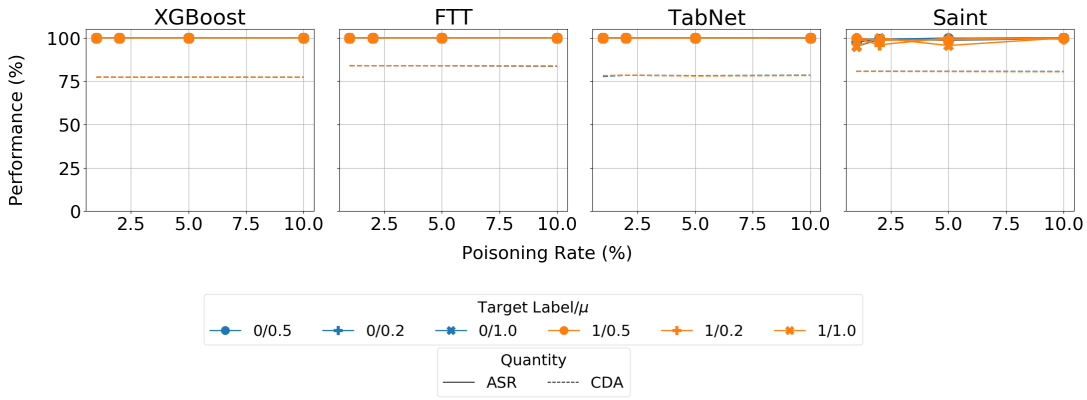


Fig. 9: CatBack's ASR and CDA vs. the poisoning rate for different μ using the HIGGS dataset.

APPENDIX C
ARTIFACT APPENDIX

A. Description & Requirements

1) How to access: The evaluator can clone our code from our publicly available repo https://github.com/catback-tabular/catback.[13]

2) Hardware dependencies:

- **CPU**: Modern multi-core processor (Intel/AMD x86_64). We have tested our code on Intel Xeon Platinum 8360Y @ 2.40GHz.
- **RAM**: Minimum 8GB, recommended 16GB+ for larger datasets. Our system has 32GB available.
- **Storage**: At least 16GB free space for datasets and model storage.
- **GPU**: Needed for faster training (CUDA-compatible GPU with 4GB+ VRAM). We have tested NVIDIA A100-SXM4-40GB.

[13]We also provide our implementation through the following link (although it is not updateable anymore and may become deprecated): https://doi.org/10.5281/zenodo.17035715.

3) Software dependencies:

- **Operating System**: Linux (we run our experiments on RHEL 9.4 and Ubuntu 24.04)
- **Python**: Version 3.8 or higher. We have run our experiments with Python 3.11.3.
- **CUDA**: version 11.0+ for using GPU acceleration. We have used CUDA 12.4.
- **PyTorch**: Compatible with CUDA version. We have tested PyTorch 2.5.1+cu124.

Our code can be run on commodity hardware (standard desktop/laptop). GPU acceleration is optional, but it significantly speeds up training.

4) Benchmarks: As we discuss in our repo's README file, we have run experiments on 5 datasets. The evaluator can test these datasets through the following steps:

- Adult Census Income (ACI): Automatically downloaded via `shap.datasets.adult()` when running the code.
- Forest Cover Type (CovType): Automatically downloaded via `sklearn.datasets.fetch_covtype()` when running the code.

- Bank Marketing (BM): Download from Kaggle: Bank Marketing Dataset and place `bank.csv` into `./data/`.
- Credit Card Fraud (CreditCard): Download from Kaggle: Credit Card Fraud Detection and place `creditcard.csv` in `./data/`.
- Download `HIGGS.csv.gz` from UCI Machine Learning Repository: HIGGS and extract `HIGGS.csv`. Then run the preprocessing script `data/HIGGS-preprocess.py` to generate `processed.pkl`. The preprocessing file is also available at GitHub: tabular-backdoors. Finally, place `processed.pkl` in `./data/`.
- Download from Kaggle: Poker Game Dataset and place `poker-hand-training.csv` and `poker-hand-testing.csv` in `./data/`.

B. Artifact Installation & Configuration

Our repo's README file contains installation instructions. The instructions are the following:

1) **Clone the repository:**

```
git clone <repository-url>
cd catback
```

2) **Create a virtual environment, upgrade pip, and install the required dependencies:**

```
python -m venv env
. env/bin/activate
python -m pip install --upgrade pip
pip install -r requirements.txt
```

We have pinned the versions in requirements.txt to match the development environment for reproducibility.[14] To resolve any issues with these specific versions (e.g., due to OS or Python version incompatibilities), the evaluator can install the latest stable versions by removing the `==version` part from the file or using `pip install <package>` without versions.

C. Major Claims

Our major claim that can be reproduced by our public code is the following:

- We applied our attack on five datasets and four models (both neural networks and classical machine learning methods), showing that it can generalize well in different settings. In particular, our attack reached $\approx$ 100% attack success rate in most cases. This claim is supported by Table 1 in our paper.

[14]By using the term reproducibility, we do not mean that the experiments will give identical results. We have not fixed any seed in our implementation for this matter. Moreover, it is worth noticing that even by fixing the seeds, the results will not be identical. This is due to the non-deterministic nature of operations in machine learning and also the hardware differences when running the experiments. Instead, by reproducible, we mean that the numbers will be in close range.

D. Evaluation

To evaluate our experiments and reproduce Table IV the evaluator can run our main script in the following way:

```
python step_by_step.py --dataset_name <dataset> \
                       --model_name <model> \
                       --target_label <target_label> \
                       --epsilon <epsilon>
```

The possible values for the dataset argument are: "aci", "bm", "higgs", "credit_card", "covtype", "poker". The values for the models are: "ftt", "tabnet", "saint", and "xgboost". Finally the values for epsilon are: 0.01, 0.02, 0.05, 0.1. Using all the possible combinations for these values will reproduce Table IV.

The duration of running experiments highly depends on the combination of the chosen dataset, model, and hyperparameters. Also the environment and selected software and hardware (e.g., OS, GPU type) highly affects the duration. For example, with our settings, the experiments could finish in as little as 5 minutes when using the Bank Marketing dataset, whereas the HIGGS dataset had the longest running time and could take several hours to finish (e.g., with our settings, it took around 26 hours to finish one Tabnet experiment on HIGGS).