DeepAID: Interpreting and Improving Deep Learning-based Anomaly Detection in Security Applications

Dongqi Han[1], Zhiliang Wang[1], Wenqi Chen[1], Ying Zhong[1], Su Wang[2], Han Zhang[1], Jiahai Yang[1], Xingang Shi[1], and Xia Yin[2]
[1]Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University, Beijing, China
[2]Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing, China
{handq19, chenwq19, zhongy18, wangsu17}@mails.tsinghua.edu.cn, {wzl, yang, shixg}@cernet.edu.cn, {zhhan,yxia}@tsinghua.edu.cn

ABSTRACT

Unsupervised Deep Learning (DL) techniques have been widely used in various security-related anomaly detection applications, owing to the great promise of being able to detect unforeseen threats and the superior performance provided by Deep Neural Networks (DNN). However, the lack of interpretability creates key barriers to the adoption of DL models in practice. Unfortunately, existing interpretation approaches are proposed for supervised learning models and/or non-security domains, which are unadaptable for unsupervised DL models and fail to satisfy special requirements in security domains.

In this paper, we propose DeepAID, a general framework aiming to (1) interpret DL-based anomaly detection systems in security domains, and (2) improve the practicality of these systems based on the interpretations. We first propose a novel interpretation method for unsupervised DNNs by formulating and solving well-designed optimization problems with special constraints for security domains. Then, we provide several applications based on our *Interpreter* as well as a model-based extension *Distiller* to improve security systems by solving domain-specific problems. We apply DeepAID over three types of security-related anomaly detection systems and extensively evaluate our Interpreter with representative prior works. Experimental results show that DeepAID can provide high-quality interpretations for unsupervised DL models while meeting several special requirements in security domains. We also provide several use cases to show that DeepAID can help security operators to understand model decisions, diagnose system mistakes, give feedback to models, and reduce false positives.

CCS CONCEPTS

• **Computing methodologies → Anomaly detection**; • **Security and privacy → Intrusion detection systems**; *File system security*.

KEYWORDS

Deep Learning Interpretability; Anomaly Detection; Deep Learning-based Security Applications

ACM Reference Format:
Dongqi Han, Zhiliang Wang, Wenqi Chen, Ying Zhong, Su Wang, Han Zhang, Jiahai Yang, Xingang Shi, and Xia Yin. 2021. DeepAID: Interpreting and Improving Deep Learning-based Anomaly Detection in Security Applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/3460120.3484589

1 INTRODUCTION

Anomaly detection has been widely used in diverse security applications [9]. Security-related anomaly detection systems are desired to be capable of detecting *unforeseen* threats such as zero-day attacks. To achieve this goal, *unsupervised* learning with only normal data, also known as "zero-positive" learning [11], becomes more promising since it does not need to fit any known threats (i.e., abnormal data) during training, compared with supervised methods.

Recently, deep learning (DL) has completely surpassed traditional methods in various domains, owing to the strong ability to extract high-quality patterns and fit complex functions [53]. Consequently, *unsupervised deep learning* models are more desirable in security domains for owing both the ability to detect unforeseen anomalies and high detection accuracy. So far, unsupervised deep learning models have been applied in various security-related anomaly detection applications, such as network intrusion detection [35], system log anomaly detection [12], advanced persistent threat (APT) detection [6], domain generation algorithm (DGA) detection [45] and web attack detection [51].

While demonstrated great promise and superior performance, DL models, specifically Deep Neural Networks (DNN) are lack of transparency and interpretability of their decisions. The black-box reputation creates key barriers to the adoption of DL models in practice, especially in security-related domains. Firstly, it is hard to establish trust on the system decision from simple binary (abnormal or normal) results without sufficient reasons and credible evidence. Secondly, black-box DL-based systems are difficult to incorporate with expert knowledge, troubleshoot and debug decision mistakes or system errors. Thirdly, reducing false positives (FP) is the most challenging issue for anomaly detection systems in practice. It is impossible to update and adjust DL models to reduce FPs without understanding how the models work. As a result, security operators are confused with over-simplified model feedback, hesitated to

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8454-4/21/11...$15.00
https://doi.org/10.1145/3460120.3484589

trust the model decisions, shriveled towards model mistakes, and overwhelmed by tons of meaningless false positives.

In recent years, several studies have attempted to develop techniques for interpreting decisions of DL models by pinpointing a small subset of features which are most influential for the final decision [37]. However, they are not directly applicable to DL-based anomaly detection in security applications for two reasons. Firstly, existing studies such as [20, 33] mostly focus on interpreting DL models for supervised classification rather than unsupervised anomaly detection. Since the mechanisms of these two types of learning methods are fundamentally different, directly applying prior techniques is improper and ineffective, as validated by our experiments in §6.2. Secondly, most prior methods are designed for non-security domains such as computer vision [16, 44, 62, 65] to understand the mechanism of DNNs. Instead, security practitioners are more concerned about how to design more reliable and practical systems based on interpretations. Besides, studies have shown that prior methods failed to be adopted in security domains due to their poor performance [14, 54].

Our Work. In this paper, we develop **DeepAID**, a general framework to interpret and improve DL-based anomaly detection in security applications. The high-level design goals of DeepAID include (1) developing a novel interpretation approach for unsupervised DL models that meets several special requirements in security domains (such as high-fidelity, human-readable, stable, robust, and high-speed), as well as (2) solving several domain-specific problems of security systems (such as decision understanding, model diagnosing and adjusting, and reducing FPs). To this end, we develop two techniques in DeepAID referred to as *Interpreter* and *Distiller*.

DeepAID Designs. *Interpreter* provides interpretations of certain anomalies in unsupervised DL models to help security practitioners understand why anomalies happen. Specifically, we formulate the interpretation of an anomaly as solving an optimization problem of searching a normal "*reference*", and pinpoint the most effective *deference* between it and the anomaly. We propose several techniques in formulating and solving the optimization problem to ensure that our interpretation can meet the special concerns of security domains. Based on the interpretations from Interpreter, we additionally propose a model-based extension *Distiller* to improve security systems. We "distill" high-level heuristics of black-box DL models and expert feedback of the interpretations into a rather simple model (specifically, finite-state machines (FSM)). Compared with the original black-box DL model, security practitioners can more easily understand and modify the simple FSM-based Distiller.

Implementation and Evaluations. We separate security-related anomaly detection systems into three types according to the structure of source data: *tabular*, *time-series*, and *graph data*. Then we provide prototype implementations of DeepAID Interpreter over three representative security systems (`Kitsune`[35], `DeepLog`[12] and `GLGV`[6]), and Distiller over tabular data based systems (`Kitsune`). We extensively evaluate our Interpreter with representative prior methods. Experimental results show that DeepAID can provide high-quality interpretations for unsupervised DL models while meeting the special requirements of security domains. We also provide several use cases to show that DeepAID can help security operators to understand model decisions, diagnose system mistakes, give feedback to models, and reduce false positives.

Contributions. This study makes the following contributions:

- We propose DeepAID, a general framework for interpreting and improving DL-based anomaly detection in security applications, which consists of two key techniques: Interpreter provides high-fidelity, human-readable, stable, robust, and fast interpretations for unsupervised DNNs. Distiller serves as an extension of Interpreter to further improve the practicality of security systems.
- We provide prototype implementations of DeepAID Interpreter over three types of DL-based anomaly detection applications in security domains using tabular, time-series, and graph data, as well as Distiller for tabular data based security applications[1].
- We conduct several experiments to demonstrate that our interpretation outperforms existing methods with respect to fidelity, stability, robustness, and efficiency.
- We introduce several use cases and improvements of security applications based on DeepAID such as understanding model decisions, model diagnosing and feedback, and reducing FPs.

The rest of the paper is organized as follows: In §2, we provide backgrounds on DL interpretability and unsupervised DL for anomaly detection in security applications, as well as the overview of DeepAID design. We introduce the motivation of this work in §3. §4 introduces the problem formulation for interpreting unsupervised DL models and its instantiations on three types of systems. §5 introduces the model-based extension Distiller. In §6, we extensively evaluate the performance of interpreters and showcase how DeepAID solves some critical problems of security systems. We make several discussions in §7 and introduce related work in §8. §9 concludes this study.

2 BACKGROUND AND OVERVIEW

In this section, we first introduce prior techniques for DL interpretability (§2.1). Then, two kinds of unsupervised DL for anomaly detection are introduced (§2.2). Next, we introduce the pipeline and three types of security-related anomaly detection systems (§2.3). Finally, the overview of DeepAID is introduced (§2.4).

2.1 Local Interpretations of DNNs

Local interpretation refers to interpreting certain decisions/outputs of DL models, which is the most prevailing form in the broad domain of DL Interpretability. We introduce three major categories of local interpretations and representative interpreters as follows.

Approximation-based Interpretations. These approaches use a rather simple and interpretable model to locally approximate original DNNs with respect to certain decisions. They commonly make the assumption that although the mapping functions in DNNs are extremely complex, the decision boundary of a specific sample can be simply approximated (e.g., by linear models). Then, the interpretation of the sample is provided by the simple model. For example, `LIME`[42] uses a set of neighbors of the interpreted sample to train an interpretable linear regression model to fit the local boundary of the original DNN. `LEMNA` [20] is another approach dedicated to Recurrent Neural Networks (RNN) in security domains. Unlike `LIME`, the surrogate interpretable model used in `LEMNA` is a non-linear mixture regression model. In addition, `LEMNA` leverages

[1] The implementation of DeepAID is available at: https://github.com/dongtsi/DeepAID

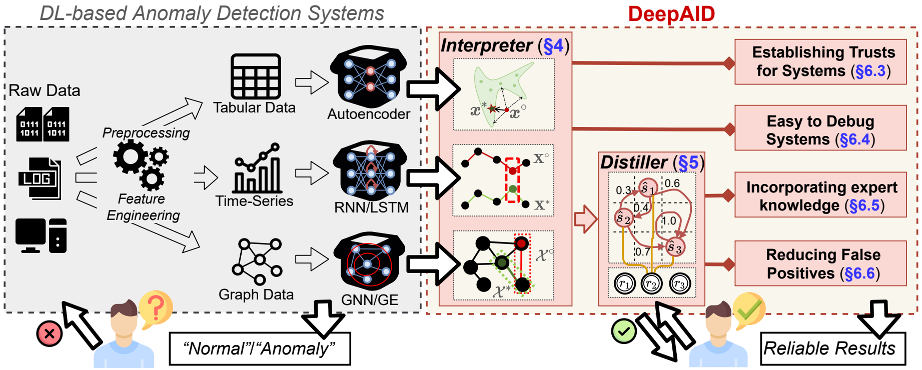


Figure 1: The overview of our work.

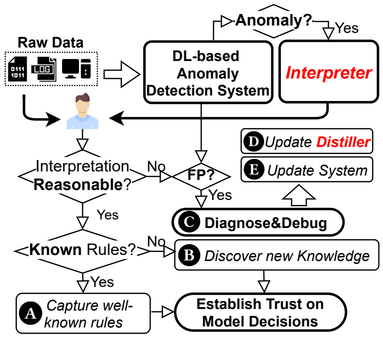


Figure 2: The workflow of applying DeepAID on security systems.

fused Lasso to cope with the feature dependence problem in RNNs. COIN [31] supports interpreting unsupervised models by training a surrogate linear SVM from interpreted anomalies and normal data.

Perturbation-based Interpretations. These approaches make perturbations on the input neurons and observe corresponding changes in model prediction. Impactful input features with small perturbations are more likely to induce major change in the outputs of DNNs [10, 15, 16, 65]. CADE [58] provides a more rational method for unsupervised interpretation of concept drift samples by perturbing the inputs and observing the distance changes to the nearest centroid in the latent space.

Back Propagation-based Interpretations. The core idea of back propagation-based approaches is to propagate the decision-related information from the output layer to the input layer of DNNs. The simple approach is to compute gradients of the output relative to the input based on the back propagation mechanism of DNNs [46]. However, gradients-based approaches have proven to be unreliable for corner cases and noisy gradients. Several improvements have been studied to solve the problem [3, 44, 48, 50]. Among them, DeepLIFT [44] outperforms others by defining a reference in the input space and computing the back propagate changes in neuron activation to compute the contribution of change between the reference and interpreted sample.

2.2 Unsupervised DL for Anomaly Detection

Unsupervised deep learning for anomaly detection is trained with purely normal data, thus are also called "zero-positive" learning [11]. Existing approaches enable DNNs to learn the distribution of normal data and find anomalies that are deviated from the distribution, which can be divided into the following two types:

Reconstruction-based Learning. This kind of learning approaches are achieved by generative DL models, such as Autoencoder [35, 56, 63] and Generative Adversarial Networks (GAN) [59]. Let $f_R : \mathbb{R}^N \to \mathbb{R}^N$ represent generative DL models mapping the input to the output space with the same dimensionality of N. The training process of these models is to minimize the distance between normal data x and its output through DNNs denoted with $f(x)$. Then, reconstruction-based models can detect anomalies by compute the reconstruction error (such as mean square error (MSE)) between given inputs and outputs. The intuition is that DNNs can learn the distribution of normal data after training, thus are more likely to reconstruct normal data with small errors, while failing to reconstruct anomalies that differ from the normal data.

Prediction-based Learning. This kind of learning approaches are usually used for time-series or sequential data and achieved by RNNs or Long short-term memory (LSTM) [11, 12]. Given a *normal* time-series $\mathbf{x} = x_1, x_2, ..., x_t$ with the length of t, prediction-based approaches force RNNs to increase the probability of predicting x_t through the sequence with first $t-1$ samples. In other words, the loss function of RNNs is to maximize $f(\mathbf{x}) = \Pr(x_t|x_1x_2...x_{t-1})$ during training. Similar to reconstruction-based models, such RNNs can predict the next (i.e., t-th) sample in normal time-series with higher probability, while wrongly predict t-th sample for anomaly (i.e., with lower probability).

2.3 Anomaly Detection Systems for Security

We start by introducing the general pipeline of unsupervised DL-based anomaly detection systems in security domains. Then, we briefly introduce three types of systems with several state-of-the-art works in diverse security domains.

Pipeline of DL-based Anomaly Detection Systems. As shown on the left of Figure 1, security applications often start with unstructured raw data, such as network traffic and system logs. These data cannot be directly fed into DNNs, thus pre-processing and feature engineering based on domain-specific knowledge are required. For example, network traffic can be processed by aggregating related packets into flows through Netflow [52], and free-text system log entries can be parsed into the sequence of log keys indicating message type for entries [12, 57]. After preprocessing, well-structured data can be fed into different kinds of DNNs, performing the aforementioned reconstruction/prediction based learning to detect anomalies. In this study, we separate security-related anomaly detection systems into three types according to different structures of source data: *tabular data*, *time-series* and *graph data*.

Tabular Data based Systems. Tabular data is the most common type that is structured into rows (also called feature vectors), each of which contains informative features about one sample. In general, most types of DNNs are designed to process tabular data. Tabular data based anomaly detection systems have been developed for network intrusion detection [35, 59] and key performance indicators (KPI) anomaly detection [56]. For example, Kitsune[35] is a

Table 1: Comparison of representative DL interpreters.

Interpreters	LIME[42]	LEMNA[20]	COIN[31]	CADE[58]	DeepLIFT[44]	**DeepAID**
Method (§2.1)	Approx.	Approx.	Approx.	Perturb.	B.P.	B.P.
Support Unsupervised*	○	○	●	●	◐	●
Support Tabular Data	●	◐	●	●	●	●
Support Time Seires	◐	●	◐	○	●	●
Support Graph Data	○	○	○	○	○	●
Stable†	○	○	◐	○	●	●
Efficient†	○	○	○	○	●	●
Robust†	○	○	◐	◐	◐	●

(**● = true, ◐ = partially true, ○ = false**);
∗ *Here we mean whether an interpreter can directly support unsupervised models without abnormal training data;*
† *These three indicators are measured through our experiments in* §6.

state-of-the-art network intrusion detection systems based on unsupervised DL. It collects raw packets from the monitoring network and then retrieves feature vector from the meta information of each packet which contains over 100 statistics. Finally, tabular features are fed into ensemble Autoencoders to perform reconstruction-based anomaly detection.

Time-Series based Systems. Time-series is a sequence of data samples indexed in time order. Any data with temporal information can essentially be represented as time-series, such as network traffic [61] and system logs [12]. Specifically, DeepLog [12] first abstracts each kind of system log as a discrete key and parse Hadoop file system (HDFS) logs to sequences of discrete log key indexes. Then, RNN/LSTM is leveraged for prediction-based anomaly detection.

Graph Data based Systems. Graph data structure is useful for modeling a set of objects (nodes) and their relationships (links) at the same time [64]. Graph data based anomaly detection is receiving more attention in security domains. Especially for advanced persistent threat (APT) detection, researchers have leveraged Graph Neural Networks (GNN) or Graph Embedding (GE) for unsupervised detection [6, 22]. For example, GLGV [6] is developed to detect lateral movement in APT campaigns. GLGV constructs authentication graphs from authentication logs within enterprise networks. The training data is collected from the purely benign authentication graphs through DeepWalk [41], which is a NN-based GE method. Thus, GLGV can use the reconstruction-based approach to detect abnormal authentications indicating lateral movement in APT.

2.4 Overview of DeepAID Design

The overview of DeepAID is shown on the right of Figure 1 including two core parts: *Interpreter* and *Distiller*. Interpreter provides high-quality interpretations dedicated to DL-based anomaly detection systems for security, which provides a general model-independent framework by formulating the interpretation to a unified optimization problem with security-related constraints. We instantiate Interpreter on aforementioned three types of security systems. Based on the interpretations from DeepAID Interpreter, we additionally propose a model-based extension called Distiller to facilitate human interaction with DL models. Distiller can store informative interpretations and expert feedback and into its FSM-based model. With Interpreter, security operators can better understand system behaviors and trust system decisions. With Distiller, DL-based security system becomes debuggable, transparent, and human-in-the-loop. In Figure 2, we provide the workflow of some use cases to improve the practicality of such security systems with the help of DeepAID, which will be discussed in detail in §6.

3 MOTIVATION

In this section, we provide the key motivations of this work from three aspects. Firstly, why unsupervised DL-based security systems need interpretations and improvements? (§3.1) Secondly, why existing interpretation approaches are insufficient for unsupervised DNNs? (§3.2) Thirdly, why we need Distiller? (§3.3)

3.1 Why Security Systems Need DeepAID?

In general, there are four major drawbacks in unsupervised DL-based security systems.

Hard to Establish Trusts for System Decisions. Security operators need sufficient reasons about the anomaly to take further actions since misoperation will induce a very high cost. Compared with traditional rule-based systems which contain detailed reasons in their alert logs, DL-based anomaly detection algorithms can only return "abnormal" or "normal". As a result, such systems become unreliable with over-simplified outputs.

Difficult to Diagnose System Mistakes. Without understanding the black-box models, it is almost impossible to diagnose and repair mistakes in DL-based systems. For example, it is difficult to tell whether a mistake is caused by implementation bugs, or over-fitting/under-fitting, or redundant features, or other reasons.

Failed to be Human-in-the-Loop. Unlike tasks in other domains such as image analysis, security-related systems should be human-in-the-loop instead of fully automated, owing to high cost of errors and reliance on expert knowledge. That is to say, security systems require to incorporate experts' knowledge and be adjusted according to expert feedback. However, the black-box nature makes it difficult for experts to interact with DL models or use expert knowledge/experience to improve systems.

Fatigue on tons of False Positives. One of the most critical challenges of developing practical DL-based anomaly detection systems in security domains is to save operators from overwhelming FPs. There are many reasons for FPs such as insufficient learning of distribution of normal training data, and the appearance of *concept drift* (i.e., distribution of test data is changing and deviates from training data).

We attribute the above drawbacks to the lack of interpretability in DNNs. Therefore, we develop DeepAID to interpret DL-based security systems and further solve the above drawbacks.

3.2 Why Not Existing Interpretations?

Although DL interpretability has been extensively studied [19, 37], we argue that existing interpretations are unadaptable for unsupervised DL-based security systems for the following two reasons.

Ill-suited for Unsupervised Learning. Most existing interpretation methods serve supervised models. However, there are clear differences between supervised and unsupervised models with respect to the mechanism of training and detection. For supervised DNNs, the interpretation goal is to answer *why a certain input is classified as an anomaly.* However, this is not the case for unsupervised DL models since they are not supposed to learn any

Table 2: Notations

Notation	Description
$\mathbf{x}^\circ; (\boldsymbol{x}^\circ, \mathrm{X}^\circ, \mathcal{X}^\circ)$	Interpreted anomaly; (tabular, time-series, graph form)
$\mathbf{x}^*; (\boldsymbol{x}^*, \mathrm{X}^*, \mathcal{X}^*)$	Reference for interpretation; (tabular, time-series, graph form)
N, K	# dimension in features and interpretation vectors (non-zero)
$f_R(\cdot), \mathcal{E}_R(\cdot,\cdot), t_R$	Reconstruction-based DNN, error function and threshold
$f_P(\cdot), t_P$	Prediction-based DNN and threshold
$\Vert \cdot \Vert_p, (\cdot)_i, (\cdot)_{(t)}$	L_p-norm, i-th element, t-th iteration
$\mathsf{E}_G(\cdot)$	Graph embedding function
$\mathcal{D}_{tab}(\cdot), \mathcal{D}_{ts}(\cdot), \mathcal{D}_{gra}(\cdot)$	Objective function (for tabular, time-series, graph form)
σ_n	Neighborhood scale for initialization of tabular Interpreter

knowledge from abnormal data. Here we introduce a more reasonable goal, which is to answer *why a certain anomaly is not considered as normal*. That is to say, for unsupervised DL models, what we need to interpret is *deviation*, not *classification*. We need to seek the reason why anomalies *deviate* from normal data.

Failed to Deploy in Security Domains. Firstly, the intention of interpreting security applications differs: Security practitioners are more concerned about how to design more reliable systems based on interpretations and solve the aforementioned domain-specific problems, rather than excessive pursuit of understanding the details of DNN. Consequently, Some interpretations used to understand the operating mechanism of DNN are not applicable to security applications [13, 39, 62]. Secondly, different from other domains, security applications have a low tolerance for errors, which requires the interpretations to be high-quality and robust. However, recent studies have shown that existing interpretations failed to be adopted in security domains due to their poor performance [14, 54]. For one thing, approximation-based interpretation approaches are inaccurate, unstable and time-consuming due to the stochastic sampling for training *surrogate* simple models. For another, existing interpretations have shown to be vulnerable to adversarial attacks [18, 47, 60] or even random noise [14, 54].

3.3 Why DeepAID Needs Distiller?

Why Distiller? With interpretations from DeepAID Interpreter, model decisions become interpretable, but it is still difficult for security participants to get involved in the decision-making process. In this case, Distiller provides a bridge to make the entire system become human-in-the-loop.

Why Not Existing Distillation Approaches? Knowledge distillation has been proposed to provide *global* interpretations by replacing the entire DL models with interpretable models [33, 55] (Note that they differ from approximation-based methods in §2.1 since the latter uses surrogate models for *local* approximation). However, global substitute model will inevitably induce performance reduction. By contrast, Distiller in DeepAID is just a back-end extension to DL-based systems, instead of replacing them.

4 INTERPRETING ANOMALIES

In this section, we first give high-level ideas of DeepAID Interpreter and provide a unified problem formulation for interpreting unsupervised DL models, followed by detailing its instantiations on three types of unsupervised DL-based security systems. Table 2 lists some important notations used in this paper.

4.1 Unified Formulation of Interpretations

High-level Ideas for Interpreting Anomalies. In this study, we primarily focus on interpreting *anomalies* since they are more concerned than normal data. As mentioned in §3.2, motivated by the different learning mechanism of unsupervised DNNs from supervised ones, we redefine interpretations of unsupervised models as seeking the reason why anomalies deviate from normal data in the DL model. From a high level, the proposed Interpreter uses "*deference* from *reference*" to interpret the *deviation* of anomalies. Specifically, we formulate the interpretation of an anomaly as solving an optimization problem which searches a most suitable reference which is considered as normal by the DL model. Then, the interpretation of this anomaly is derived by pinpointing the *deference* between this anomaly and its reference.

Special Concerns for Security Domains. As mentioned before, there are special security requirements for interpreters in security domains [14, 54].

(1) **Fidelity**: should be high-fidelity for imitating original DNNs;
(2) **Conciseness**: results should be concise and human-readable;
(3) **Stability**: results on the same sample should be consistent;
(4) **Robustness**: robust against adversarial attacks or noises;
(5) **Efficiency**: interpretations should be available without delaying the workflow of high-speed online systems.

Unified Problem Formulation. Given an anomaly $\mathbf{x}^\circ$, our goal is to find a reference $\mathbf{x}^*$, then the interpretation of $\mathbf{x}^\circ$ is provided by the difference between $\mathbf{x}^\circ$ and $\mathbf{x}^*$. In light of the above requirements, we formulate interpretations of $\mathbf{x}^\circ$ in unsupervised DNNs (denoted with f) for security domains as the following optimization problem:

$$\operatorname{argmin}_{\mathbf{x}^*} \ \mathcal{L}_{fid}(\mathbf{x}^*; f) + \lambda_1 \mathcal{L}_{sta}(\mathbf{x}^\circ; f, \mathbf{x}^*) + \lambda_2 \mathcal{L}_{con}(\mathbf{x}^\circ, \mathbf{x}^*) \quad (1)$$

$$\text{s.t.} \ \ \mathbf{x}^* \in \mathcal{R}(\mathbf{x}^\circ), \quad (2)$$

where $\mathcal{L}_{fid}$, $\mathcal{L}_{sta}$ and $\mathcal{L}_{con}$ respectively measures the loss of fidelity, stability, and conciseness, balanced by weight coefficients λ_1 and λ_2. The constraint (2) means that $\mathbf{x}^*$ must be searched in the same feature space as the anomaly $\mathbf{x}^\circ$ (measured by $\mathcal{R}$). Below, we will detail instantiations of the formulation and technical solution for three types of systems (according to different data types of $\mathbf{x}^\circ$).

4.2 Tabular Data Interpreter

As mentioned in §2, tabular data needs reconstruction-based learning models. Following the unified framework in §4.1, the fidelity loss of tabular anomaly data $\boldsymbol{x}^\circ$ can be transformed by adding a constraint to ensure searched reference $\boldsymbol{x}^*$ is decided to be normal. To define stability loss, we ensure $\boldsymbol{x}^*$ to be as close to $\boldsymbol{x}^\circ$ as possible while satisfying other constraints. We use Euclidean distance for measuring this term. To define conciseness loss, we leverage L_0-norm to measure the difference between $\boldsymbol{x}^*$ and $\boldsymbol{x}^\circ$, which measures total number of non-zero elements in $\boldsymbol{x}^* - \boldsymbol{x}^\circ$. Consequently, the interpretation of tabular anomaly can be formulated as:

$$\operatorname{argmin}_{\boldsymbol{x}^*} \ \|\boldsymbol{x}^* - \boldsymbol{x}^\circ\|_2 + \lambda \|\boldsymbol{x}^* - \boldsymbol{x}^\circ\|_0 \quad (3)$$

$$\text{s.t.} \ \ \mathcal{E}_R(\boldsymbol{x}^*, f_R(\boldsymbol{x}^*)) < t_R \quad (4)$$

$$\boldsymbol{x}^* \in [0,1]^N, \quad (5)$$

where the first term in objective function (3) is the stability loss, and the second term is the conciseness loss (weighted by λ). The fidelity loss is constrained by (4), which means $\boldsymbol{x}^*$ is decided to be normal. Constraint (5) limits $\boldsymbol{x}^*$ within the valid range. Note that, $\boldsymbol{x}^*$ here is after normalization to $[0, 1]$. Normalizations to other ranges are also supported, which will be explained later.

Challenges. The above formulation is difficult for solving directly due to three challenges. Firstly, L_0-norm minimization problem is proven to be NP-hard [23]. Secondly, constraint (4) is highly non-linear. Thirdly, constraint (5) is a box constraint which cannot be directly solved by most algorithms. Below, several techniques are introduced to address these challenges.

Iteratively Optimizing. To address the first challenge, we transform the L_0-norm term in the objective function into iteratively optimizing another terms. That is, in each iteration, we identify some dimensions in $\boldsymbol{x}^*$ that don not have much effect on minimizing the objective function, then their value will be replaced by corresponding value in $\boldsymbol{x}^\circ$. How to select *ineffective dimensions* will be discussed later. In this way, $\|\boldsymbol{x}^* - \boldsymbol{x}^\circ\|_0$ can be effectively limited through changing a small number of influential dimensions.

Bounding Loss. To address the second challenge, we transform the highly non-linear constraint (4) by adding a term into the objective function to minimize $\mathcal{E}_R(\boldsymbol{x}^*, f_R(\boldsymbol{x}^*))$. However, this term should not be minimized indefinitely since the goal of $\boldsymbol{x}^*$ is to probe the decision boundary of f_R. Thus, we limited $\mathcal{E}_R(\boldsymbol{x}^*, f_R(\boldsymbol{x}^*))$ to be close to t_R by employing ReLU function ($\mathsf{ReLU}(x) = \max(0, x)$). To ensure $\boldsymbol{x}^*$ is "*within*" the side of normal data with respect to the decision boundary, t_R is subtracted by a small ϵ. Therefore, constraint (4) is replaced by the following term:

$$\mathsf{ReLU}\big(\mathcal{E}_R(\boldsymbol{x}^*, f_R(\boldsymbol{x}^*)) - (t_R - \epsilon)\big). \tag{6}$$

Variable Changing. We eliminate the box constraint (5) through the idea of change-of-variables [8] with tanh function. We introduce a new vector $\boldsymbol{u}$ with the same shape of $\boldsymbol{x}^*$. Since the range of $\tanh(\boldsymbol{u})$ is $(-1, 1)$, $\boldsymbol{x}^*$ can be replaced by simple linear transformations of $\tanh(\boldsymbol{u})$. For $\boldsymbol{x}^* \in [0, 1]^N$, $\boldsymbol{x}^* = \frac{1}{2}(\tanh(\boldsymbol{u}) + 1)$. For $\boldsymbol{x}^* \in [a, b]^N$($a, b$ is arbitrary as long as $a < b$), $\boldsymbol{x}^* = \frac{b-a}{2}\tanh(\boldsymbol{u}) + \frac{b+a}{2}$. Thus, the constraint (5) is eliminated. As for categorical features, we simply relax their encoding (used in original DL models) to continuous variables and convert them to discrete values after solving the reference $\boldsymbol{x}^*$.

With above techniques, the optimization problem is converted into the following unconstrained one **in each iteration**:

$$\begin{aligned} &\operatorname{argmin}_{\boldsymbol{u}}\ \mathsf{ReLU}\big(\mathcal{E}_R(\boldsymbol{x}^*, f_R(\boldsymbol{x}^*)) - (t_R - \epsilon)\big) + \lambda\|\boldsymbol{x}^* - \boldsymbol{x}^\circ\|_2 \\ &\quad \text{where}\ \ \boldsymbol{x}^* = \begin{cases} \frac{1}{2}(\tanh(\boldsymbol{u}) + 1), & \boldsymbol{x}^* \in [0, 1]^n, \\ \frac{b-a}{2}\tanh(\boldsymbol{u}) + \frac{b+a}{2}, & \boldsymbol{x}^* \in [a, b]^n. \end{cases} \end{aligned} \tag{7}$$

For illustration purposes, we abbreviate the objective function in (7) as $\mathcal{D}_{tab}(\boldsymbol{x}^*; \mathcal{E}_R, f_R, t_R, \epsilon, \lambda, \boldsymbol{x}^\circ)$, more concisely $\mathcal{D}_{tab}(\boldsymbol{x}^*; \boldsymbol{x}^\circ)$ henceforth. This problem can be solved by gradient-based optimization approaches (we use Adam optimizer [28] in this study) since the objective function is fully differentiable.

We now introduce how to select *ineffective dimensions* during iterative optimizations. In each iteration, the best $\boldsymbol{x}^*$ is computed by solving (7), denoted with $\boldsymbol{x}^*_{(t)}$ for the t-th iteration. We first compute the gradient of $\mathcal{D}_{tab}$ evaluated at $\boldsymbol{x}^*_{(t)}$, denoted by $\nabla\mathcal{D}_{tab}(\boldsymbol{x}^*_{(t)}; \boldsymbol{x}^\circ)$. Motivated by the prior work [3], we use *gradient* × *input* together instead of only gradients for accurately measuring the effectiveness of $(\boldsymbol{x}^*_{(t)})_i$. Thus, we can select *effective* dimension(s) by solving:

$$i^* = \operatorname{argmax}_i \big(\nabla\mathcal{D}_{tab}(\boldsymbol{x}^*_{(t)}; \boldsymbol{x}^\circ)\big)_i \cdot (\boldsymbol{x}^*_{(t)})_i \text{ where } i \in \{1, 2, ..., N\}$$

by calculating and sorting N objective function values. Then, for each $i \neq i^*$, $(\boldsymbol{x}^*_{(t)})_i$ is replaced by $(\boldsymbol{x}^\circ)_i$, ending this iteration.

Algorithm 1: Interpreting tabular anomalies

```
   Input: Interpreted anomaly $x^\circ$; $max_{iter}$, $\alpha$ (learning rate); $\sigma_n$
   Output: Interpretation result $x^\circ - x^*$ with the reference $x^*$
1  $x^*_{(1)} \leftarrow x^\circ + \mathcal{N}(0, \sigma_n^2)$;   $t \leftarrow 1$ ;        ▹ initialization of reference
2  while $t \leq max_{iter}$ do                                   ▹ iterative optimization
3      $x^*_{(t)} \leftarrow$Adam($x^*_{(t)}$; $\mathcal{D}_{tab}, \alpha$) ; ▹ update $x^*_{(t)}$ by Adam optimizer
4      $i^* \leftarrow \operatorname{argmax}_i \big(\nabla\mathcal{D}_{tab}(x^*_{(t)}; x^\circ)\big)_i \cdot (x^*_{(t)})_i$;
5      for $i = 1$ to $N$ do                           ▹ replacing ineffective dimensions
6          if $i \neq i^*$ then  $(x^*_{(t)})_i \leftarrow (x^\circ)_i$ ;
7      end
8      $x^*_{(t+1)} \leftarrow x^*_{(t)}$;   $t \leftarrow t + 1$;
9  end
10 return $x^\circ - x^*_{(max_{iter})}$
```

Adversarial Robustness. Gradient-based interpretations are vulnerable against adversarial attacks [18, 60] since attackers can also use the gradients for adversarial optimization. However, our method has two natural advantages to defend against adversarial attacks compared with previous interpretations. First, our method is indifferentiable from overall perspective as we use iterative optimization to cope with L_0-norm. Therefore, the attacker cannot directly obtain the overall gradients. Second, our method is *search*-based. DeepAID interprets anomalies through searching reference $\boldsymbol{x}^*$, rather than directly based on the gradient of interpreted anomalies $\boldsymbol{x}^\circ$. Therefore, it is more difficult for attackers to affect the entire search process by only perturbing the searching entrance.

Initialization of $\boldsymbol{x}^*$. A straightforward initialization method of $\boldsymbol{x}^*$ is directly searching from anomaly $\boldsymbol{x}^\circ$. To further improve robustness, we initialize $\boldsymbol{x}^*$ from the neighborhood of anomaly $\boldsymbol{x}^\circ$. Formally, $\boldsymbol{x}^*_{(1)} = \boldsymbol{x}^\circ + \mathcal{N}(0, \sigma_n^2)$ where $\mathcal{N}$ is Gaussian distribution and σ_n reflects the scale of neighborhood. Such initialization can mitigate the attack effect on the entire search process for reference by mitigating the effect on the searching entrance $\boldsymbol{x}^*_{(1)}$.

With aforementioned techniques, the whole procedure for interpreting tabular anomalies is available at Algorithm 1.

Improving Efficiency. There are several skills to improve efficiency of our interpretation in Algorithm 1. First, we can use the loss change of two iterations to "early stop" the iterative optimization (instead of a fixed number of iterations max_{iter} on line 7). Second, increasing the learning rate α gracefully can also speed up the solution (on line 8). Third, we can computer multiple i^* (on line 9) to speed up the solution. Fourth, We can update only a few effective dimension(s) i^* instead of replacing the majority of ineffective dimensions (on lines 10-13).

Improving Conciseness. We can fix the *non-zero* dimension of interpretations to K by slightly modifying Algorithm 1: We accumulate the importance of each dimension (measured on line 9) over iterations, then only preserve top-K dimensions in $\boldsymbol{x}^\circ - \boldsymbol{x}^*$ and clip others to 0 at the end of algorithm.

4.3 Time-Series Interpreter

For interpretations of the remaining two data structures, we primarily introduce techniques that are different from the tabular case. As introduced in §2.2, anomaly detection for time-series data prefers prediction-based learning with RNN/LSTM. We denote the interpreted time-series anomaly as $\mathbf{X}^{\circ}$ consists of $\boldsymbol{x}_1^{\circ}, \boldsymbol{x}_2^{\circ}, ..., \boldsymbol{x}_t^{\circ}$. Time-series data can be generally divided into *univariate* and *multivariate*. Each $\boldsymbol{x}_i^{\circ}$ ($i \in \{1, 2, ..., t\}$) in $\mathbf{X}^{\circ}$ is a one-hot vector for univariate time series, while is a tabular feature vector for multivariate time series. Below, we primarily introduce how to interpret univariate time-series since the demonstrative system (DeepLog [12]) in our study uses univariate data, and then briefly introduce how to extend it to multivariate time series.

As before, our interpretation goal for univariate time-series anomaly is to find a normal time-series reference $\mathbf{X}^*$. Intuitively, this is like to *correct* a small number of influential abnormal points in the time-series. Unlike tabular feature vectors, the changes of one-hot vectors in univariate time-series are discrete (i.e., 0 or 1). Thus, the L_2-norm stability loss and L_0-norm conciseness loss (previously in (3)) become equally indifferentiable. The fidelity constraint (4) becomes $f_P(\mathbf{X}^*) > t_P$. The validity constraint (5) turns into constraining each $\boldsymbol{x}_i^*$ in $\mathbf{X}^*$ is still one-hot ($i \in \{1, 2, ..., t\}$). We again use the idea of *iteratively optimizing* and *bounding loss* to solve similar challenges as tabular data. Thus, we have the objective function **in each iteration** denoted with $\mathcal{D}_{ts}(\mathbf{X}^*; \boldsymbol{x}_t^*)$ as follows:

$$\begin{aligned} \mathcal{D}_{ts}(\mathbf{X}^*; \boldsymbol{x}_t^*) &= \mathsf{ReLU}\big((t_P + \epsilon) - f_P(\mathbf{X}^*)\big) \\ &= \mathsf{ReLU}\big((t_P + \epsilon) - \Pr(\boldsymbol{x}_t^* | \boldsymbol{x}_1^* \boldsymbol{x}_2^* ... \boldsymbol{x}_{t-1}^*)\big). \end{aligned} \tag{8}$$

Locating Anomaly. Before solving $\mathcal{D}_{ts}(\mathbf{X}^*)$, there is a special challenge for time-series interpretation. That is, we need to figure out whether an anomaly $\mathbf{X}^{\circ}$ is caused by $\boldsymbol{x}_t^{\circ}$ or $\boldsymbol{x}_1^{\circ} \boldsymbol{x}_2^{\circ} ... \boldsymbol{x}_{t-1}^{\circ}$. This is because $\mathbf{x}_t$ serves as the "label" of $\mathbf{x}_1 \mathbf{x}_2 ... \mathbf{x}_{t-1}$ (recall the prediction-based learning introduced in §2.2). If the label itself is abnormal, modifying other parts will be useless. If the anomaly is caused by $\boldsymbol{x}_t^{\circ}$, then we can simply replace $\boldsymbol{x}_t^{\circ}$ with the following $\boldsymbol{x}_t^*$ to turn the time-series into normal:

$$\boldsymbol{x}_t^* = \boldsymbol{x}^c = \mathrm{argmax}_{\boldsymbol{x}^c} \ \Pr(\boldsymbol{x}^c | \boldsymbol{x}_1^{\circ} \boldsymbol{x}_2^{\circ} ... \boldsymbol{x}_{t-1}^{\circ}). \tag{9}$$

Such $\boldsymbol{x}_t^*$ can be obtained directly by observing the index of the maximum probability in the last layer (usually softmax) of the RNN/LSTM. Once $\boldsymbol{x}_t^{\circ}$ is replaced, the whole process of solving $\mathbf{X}^*$ ends immediately. One the other hand, if the anomaly is not caused by $\boldsymbol{x}_t^{\circ}$, we need to solve $\boldsymbol{x}_1^* \boldsymbol{x}_2^* ... \boldsymbol{x}_{t-1}^*$ with $\boldsymbol{x}_t^{\circ}$ through (8). Formally,

$$\boldsymbol{x}_1^* \boldsymbol{x}_2^* ... \boldsymbol{x}_{t-1}^* = \big(\mathrm{argmax}_{\mathbf{X}^*} \ \mathcal{D}_{ts}(\mathbf{X}^*; \boldsymbol{x}_t^{\circ})\big)_{1,2,...,t-1}. \tag{10}$$

To locate anomaly, we introduce **Saliency Testing**. The intuition is that, if (1) it is hard to make $\mathbf{X}^{\circ}$ normal by changing $\boldsymbol{x}_1^{\circ} \boldsymbol{x}_2^{\circ} ... \boldsymbol{x}_{t-1}^{\circ}$, and (2) RNN/LSTM originally has great confidence of $\boldsymbol{x}^c$ ($\boldsymbol{x}^c \neq \boldsymbol{x}_t^{\circ}$), then we decide the anomaly is caused by $\boldsymbol{x}_t^{\circ}$. Formally, saliency testing denoted with $ST(\mathbf{X}^{\circ}; \mu_1, \mu_2)$ is conducted by:

$$ST(\mathbf{X}^{\circ}; \mu_1, \mu_2) = (\max(\nabla \mathcal{D}_{ts}(\mathbf{X}^{\circ}; \boldsymbol{x}_t^{\circ})) < \mu_1) \wedge (\boldsymbol{x}^c > \mu_2), \tag{11}$$

which respectively represent the above two conditions. The detailed configuration method of μ_1 and μ_2 is in Appendix F. To conclude, time-series reference $\mathbf{X}^*$ is solved as:

$$\mathbf{X}^* = \begin{cases} \boldsymbol{x}_1^{\circ} \boldsymbol{x}_2^{\circ} ... \boldsymbol{x}_{t-1}^{\circ} \underline{\boldsymbol{x}_t^* \leftrightarrow \text{Eq.}(9)}, & ST(\mathbf{X}^{\circ}; \mu_1, \mu_2) = \textbf{True}, \\ \underline{\boldsymbol{x}_1^* \boldsymbol{x}_2^* ... \boldsymbol{x}_{t-1}^* \leftrightarrow \text{Eq.}(10)} \boldsymbol{x}_t^{\circ}, & \text{otherwise (iteratively).} \end{cases}$$

The whole procedure for interpreting univariate time-series anomalies is available at Algorithm 2 in Appendix A. We omit the discussion of robustness for this kind of interpreters since there is no study on adversarial attacks against such kind of interpreter to the best of our knowledge. Besides, the robustness is naturally stronger due to the discreteness of changes in time-series data.

Interpreting Multivariate Time-Series. We briefly introduce how to extend our interpretation to multivariate time-series, which can be viewed as a combination of tabular and univariate time-series interpretation methods. We first locate a small number of influential data points in an abnormal time-series. Then, for each data point (essentially a feature vector), we can use the tabular interpretation method (in §4.2) to search its reference.

4.4 Graph Data Interpreter

There are many types of graphs (such as *attributed* and *unattributed*) and learning tasks (such as *node classification* and *link prediction*). For attributed graph (i.e., each node or link has a feature vector), interpretations can be obtained by the same idea as the interpretation of multivariate time series. That is, first locating a small number of abnormal nodes/links in the graph, and then using tabular interpretation method to modify important feature dimensions to find the reference. Therefore, below we primarily introduce how to interpret unattributed graph anomalies. Since the learning task in our demonstrative system (GLGV [6]) is link prediction, we will introduce how to interpret abnormal links in graph anomaly.

The common method of processing graph data is to embed it into a continuous feature space, or directly leveraging end-to-end graph neural networks (GNN). Unattributed graphs usually require the first method, namely graph embedding (GE), such as DeepWalk [41]. Let E_G denotes GE, then the general workflow of unsupervised DL with graph $\mathcal{G}$ is to first compute the embedding vector $\boldsymbol{e} = \mathsf{E}_G(\mathcal{G})$. For link-level interpretation, an abnormal link $\mathcal{X}^{\circ} = (x_a^{\circ}, x_b^{\circ})$ (x_a° and x_b° are two nodes identify this link) is embedded as $\boldsymbol{e}^{\circ} = \mathsf{E}_G(\mathcal{X}^{\circ})$. Therefore, the interpretation of graph link anomaly is derived through two steps. (1) **Step 1**: Interpreting $\boldsymbol{e}^{\circ}$ by finding its reference $\boldsymbol{e}^*$. (2) **Step 2**: Finding the original representation of $\boldsymbol{e}^*$ in the graph.

Details of step 1 are omitted since it is the same as the tabular interpretation. As for step 2, we find the original representation $\mathcal{X}^* = (x_a^*, x_b^*)$ of $\boldsymbol{e}^*$ by solving the following optimization problem:

$$\mathrm{argmin}_{\mathcal{X}^* = (x_a^*, x_b^*)} \ \mathcal{F}_1(\mathcal{X}^*) + \lambda \mathcal{F}_2(\mathcal{X}^*, \boldsymbol{e}^*) \quad \text{s.t. } x_1^*, x_2^* \in \mathcal{V}, \tag{12}$$

$$\begin{aligned} &\text{where} \ \ \mathcal{F}_1(\mathcal{X}^*) = \mathsf{ReLU}\big(\mathcal{E}_R\big(\mathsf{E}_G(\mathcal{X}^*), f_R(\mathsf{E}_G(\mathcal{X}^*))\big) - (t_R - \epsilon)\big) \\ &\text{and} \ \ \mathcal{F}_2(\mathcal{X}^*, \boldsymbol{e}^*) = \|\mathsf{E}_G(\mathcal{X}^*) - \boldsymbol{e}^*\|_2. \end{aligned} \tag{13}$$

Here the first term in objective function (12) means $\mathsf{E}_G(\mathcal{X}^*)$ is decided to be normal by the interpreted system, and the second term measures the difference of $\mathsf{E}_G(\mathcal{X}^*)$ and $\boldsymbol{e}^*$ obtained in step 1. The intuition of (12) is to force $\mathsf{E}_G(\mathcal{X}^*)$ to close to $\boldsymbol{e}^*$, and ensure $\mathsf{E}_G(\mathcal{X}^*)$ is a normal embedding vector. The constraint in (12) means x_a° and x_b° are still in the node set denoted with $\mathcal{V}$.

Problem (12) can be simply solved by gradient-based optimizer if E_G is differentiable. However, embeddings of inputs are derived by an indifferentiable *look-up* operation [34] in many cases. For example, most DL frameworks (e.g., PyTorch [40]) set the embedding layer to be indifferentiable. A simple method is to use a fully connected network (FCN) to achieve the same embedding function, but this requires additional work to modify the interpreted model. In Appendix A.2, we propose an alternative greedy solution to (12) when E_G is indifferentiable, and also provide the whole algorithm.

5 DISTILLING INTERPRETATIONS

As introduced in §3.3, to facilitate human interaction with DL-based anomaly detection systems, we propose a model-based extension based on our interpretations, referred to as *Distiller*. In this study, we primarily design Distiller for tabular data and will extend it to other data types in future work. Below, we introduce overview, details, superiority, and application of Distiller.

Distiller Overview. In a nutshell, Distiller allows security analysts to give feedback on interpretation results thus integrating expert knowledge into the security system. Specifically, analysts can give "*feedback*" (denoted with r_i) on a set of anomalies after checking their interpretations ($\boldsymbol{x}^\circ - \boldsymbol{x}^*$) from Interpreter. The number and content of feedback is highly free for analysts. We refer to "feedback on interpretations" as "*rule*" denoted as $(\boldsymbol{x}^\circ - \boldsymbol{x}^*) \rightarrow r_i$. The intuition is that, after adding rules based on expert knowledge, Distiller is able to automatically recognize similar anomalies in the future to output more reliable results. The design goal of Distiller is to store rules when analysts need to add new ones (called *Update* mode), and to match "ruled" anomalies and report previous expert feedback (called *Test* mode). To this end, Distiller essentially consists of two finite-state machines (FSM). The first FSM is used to model K-dimension interpretation ($\boldsymbol{x}^\circ - \boldsymbol{x}^*$), and the second FSM is used to model the transition from interpretation to feedback r_i. Below, we will introduce the design of two FSMs and two modes.

State and Transition Abstraction of FSMs (Figure 3). In the first FSM, we split the value range of each dimension in the interpretation vector $\boldsymbol{x}^\circ - \boldsymbol{x}^*$ into M equal-length intervals, then we have totally $M \times N$ states. Each of K dimensions in $\boldsymbol{x}^\circ - \boldsymbol{x}^*$ is mapped into one of states $\hat{s}_i$ according to its dimension and value. Subsequently, transitions of these K states are arranged according to the order of decreasing effectiveness. That is, the state with the largest effectiveness becomes the initial state, and the smallest one is the final state. The second FSM contains all MN states in the first FSM and variable states indicating feedback r_i. Transitions in the second FSM are all single-step from an interpretation $\hat{s}_i$ (initial state) to a feedback state r_i (final state). The intuition of two FSMs is that all dimensions of interpretations "contribute" to the feedback (the second FSM), and their order matters (the first FSM).

Update and Test Mode. Distiller works on two modes: (1) *Update* mode indicates adding new rules proposed by analysts. This can be implemented by separately updating two *transition matrices* of two FSMs according to the aforementioned transitions of the new rule in two FSMs. (2) *Test* mode matches anomaly interpretations to existing feedback. Interpretation $\boldsymbol{x}^\circ - \boldsymbol{x}^*$ raised by our Interpreter is first mapped into a sequence of states $\hat{s}_1\hat{s}_2 ... \hat{s}_K$. Then, the matching probability of the interpretation to each feedback $\hat{r}$ is calculated as:

$$\mathcal{P}\,(\hat{r} \mid \hat{s}_1\hat{s}_2...\hat{s}_K) = \frac{1}{K}\sum_{i=1}^{K}\big(\mathcal{P}_2(\hat{r} \mid \hat{s}_i)\prod_{j=1}^{i-1}\mathcal{P}_1(\hat{s}_{j+1} \mid \hat{s}_j)\big), \qquad (14)$$

where $\mathcal{P}_1$ and $\mathcal{P}_2$ is transition probability in two FSMs/transition matrices. The intuition is to calculate the mean of the product of transition probability from $\hat{s}_1$ to $\hat{s}_i$ and from $\hat{s}_i$ to $\hat{r}$ ($i \in \{1, 2, ..., K\}$).

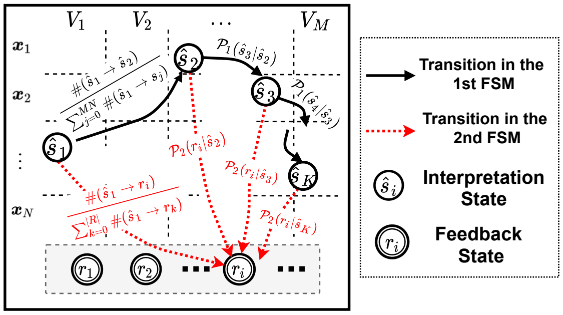

V_1
V_2
$\cdots$
V_M
$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\vdots$
$\boldsymbol{x}_N$
$\frac{\#(\hat{s}_1 \rightarrow \hat{s}_2)}{\sum_{j=0}^{MN}\#(\hat{s}_1 \rightarrow s_j)}$
$\hat{s}_1$
$\hat{s}_2$
$\hat{s}_3$
$\hat{s}_K$
$\mathcal{P}_1(\hat{s}_3|\hat{s}_2)$
$\mathcal{P}_1(\hat{s}_4|\hat{s}_3)$
$\mathcal{P}_2(r_i|\hat{s}_2)$
$\mathcal{P}_2(r_i|\hat{s}_3)$
$\mathcal{P}_2(r_i|\hat{s}_K)$
$\frac{\#(\hat{s}_1 \rightarrow r_i)}{\sum_{k=0}^{|R|}\#(\hat{s}_1 \rightarrow r_k)}$
r_1
r_2
r_i
Transition in the 1st FSM
Transition in the 2nd FSM
$\hat{s}_i$ Interpretation State
r_i Feedback State

Figure 3: States and transitions of DeepAID Distiller.

Toy Example. In Figure 4, we provide a toy example of updating and testing two anomalies (called **A1**/**A2**) from an empty Distiller. The ground truth (unknown to analysts) of **A1** and **A2** is IP and Port Scan. Suppose the analysts mark **A1** as "Scanning" after reading the interpretation ($K = 3$ here) and then updates Distiller with the new rule (❷), which essentially adds a new feedback state r_1 indicating "Scanning" and updates the corresponding transitions in the two FSMs. At this time, if we test **A1** and **A2** (❸), Distiller will return strong confidence of matching **A1** to "Scanning" ($\mathcal{P}(r_1|\hat{s}_1\hat{s}_2\hat{s}_3) = 1$) and weak confidence of matching **A2** to "Scanning" ($\mathcal{P}(r_1|\hat{s}_4\hat{s}_5\hat{s}_3) = 0.33$). Then after updating the second rule indicating **A2** to feedback "Port Scan" (❹), Distiller (❺) will return both strong confidences of matching **A1** to "Scanning" and matching **A2** to "Port Scan" ($\mathcal{P}(r_1|\hat{s}_1\hat{s}_2\hat{s}_3) = \mathcal{P}(r_2|\hat{s}_4\hat{s}_5\hat{s}_3) = 0.83$).

Distiller vs. ML/DL Classifier. Distiller shares some similarities with ML/DL classifiers, e.g., update/test mode is similar to training/prediction on multi-class tasks, and feedback can be viewed as labels. However, Distiller is more competent for human-in-the-loop detection for three reasons. First and most important, Distiller is more transparent and easy to modify. Security analysts can clearly and simply modify transitions and states in Distiller. Second, the number of classes/feedback in Distiller is adaptable. Third, Distiller can preserve the ability of anomaly detection systems to detect unknown threats by initially transiting all states in the first FSM to a feedback state representing unforeseen threats. We conduct experiments in §6.5 to demonstrate the superiority of Distiller.

Distiller vs. Rule Matching. Another potential baseline of Distiller is rule-matching methods. Compared with them, Distiller has the generalization ability to report feedback for *similar* anomalies, e.g., at ❸ (Figure 4), **A2** is also reported as "Scanning" due to its similarity to the first rule (i.e., $\mathcal{P}(r_1|\hat{s}_4\hat{s}_5\hat{s}_3) = 0.33$ but not 0).

Reducing FPs via Distiller. We showcase how Distiller may help to reduce two types of FPs in original anomaly detection models: (1) Low confidence positives. Such FPs are caused by normal data nearly decision boundary/threshold (e.g., t_R, t_P). To address them,

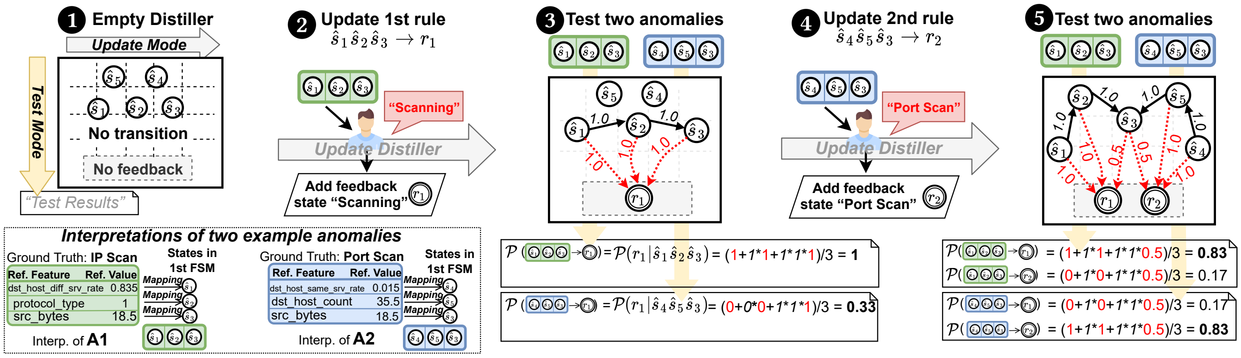


Figure 4: Toy example of updating and testing two anomalies A1 (green) and A2 (blue) in DeepAID Distiller.

we can modify Distiller by forcing some selected FP interpretations to transit to feedback states indicating normal (or FP). (2) Concept drift of normal data. We can locate and retrain FPs with very low transition probability in the first FSM, which indicates such FPs are neither normal (judged by DL model) nor known anomalies.

Theoretical Analysis. We provide proofs of the correctness, accuracy bound, and complexity of Distiller in Appendix B.

6 EXPERIMENTS

In this section, we conduct several experiments with baseline interpreters, as well as provide several case studies to introduce how to interpret and improve security-related anomaly detection systems with DeepAID. Due to space limit, we primarily evaluate tabular and time-series data based systems (as none of baseline supports graph data). Experiments on graph data based systems as well as several details and supplements of our experiments are left in Appendix C and E. Overall, we provide a roadmap of this section:

- **Experimental Setup.** We introduce implementation and setup of security systems, DeepAID, and baselines in this study (§6.1).
- **Interpreter Performance.** We demonstrate the performance of DeepAID Interpreter with respect to fidelity, stability, efficiency, and robustness, compared with existing works (§6.2).
- **Understanding Model Decisions.** We provide case studies on leveraging DeepAID interpretations to establish trusts in model decisions and discovering implicit model knowledge (§6.3).
- **Enabling Debuggability.** We provide a case on diagnosing and debugging system mistakes based on DeepAID (§6.4).
- **Human-in-the-Loop Detection.** For tabular security systems, we conduct reliable detection with expert feedbacks and knowledge based on DeepAID Interpreter and Distiller (§6.5).
- **Reducing FPs.** For tabular security systems, we showcase that DeepAID can reduce FPs based on Distiller (§6.6).

6.1 Experimental Setup

Here we briefly introduce the implementation and experimental setup, and we refer readers to Appendix C for details, as they involve many domain-specific backgrounds.

Security Systems. We use three demonstrative DL-based anomaly detection systems in security domains using three data types: `Kitsune`[35], `DeepLog`[12], and `GLGV`[6], which are introduced in §2.3. The data used in this work is primarily from their released datasets (see Appendix C for more details).

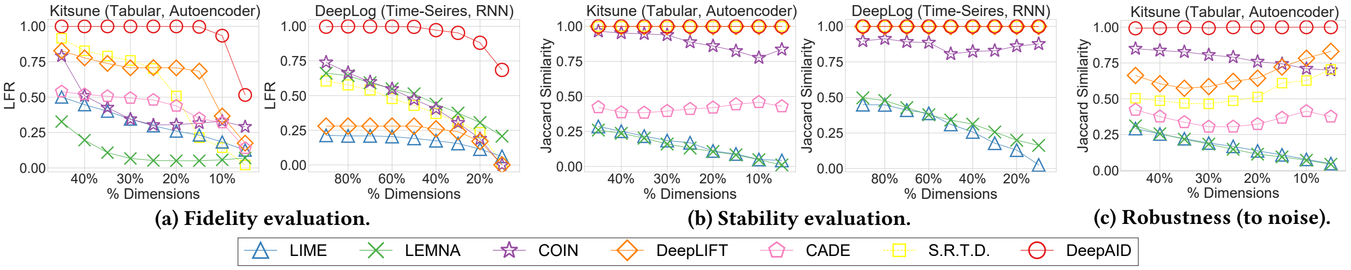
DeepAID Implementation. For Interpreter, we fix the interpretation results into K-dimension. By default, we set $\lambda = 0.001$ and $\epsilon = 0.01$ in objective functions (7), (8), (12), learning rate $\alpha = 0.5$ in Adam optimizer, and $max_{iter} = 20$. For tabular Interpreter, we set neighborhood scale $\sigma_n = 0$ by default for initialization and evaluate its effect in §6.2. For time-series Interpreter, we set $\mu_1 = 0.01$ and $\mu_2 = 0.3$ by default. For Distiller, we set $M = 20$ (number of intervals) by default. The sensitivity evaluation and configuration guideline of these hyper-parameters are in Appendix F.

Baseline Interpreters and Implementation. In §6.2, we evaluate DeepAID Interpreter with several baseline interpreters listed in Table 1 and introduced in §2.1. (1) For supervised interpretations (`LIME`, `LEMNA`, and `DeepLIFT`), we approximate the decision boundary of the anomaly detection with a supervised DNN trained with additionally sampled anomalies. (2) `COIN` and `CADE` are unsupervised baselines. `CADE` needs a supervised encoding model before interpreting [58] (See Appendix C for details). (3) We also directly **S**elect the **R**eference from **T**raining **D**ata (abbreviated as **S.R.T.D.**).

6.2 Interpreter Performance

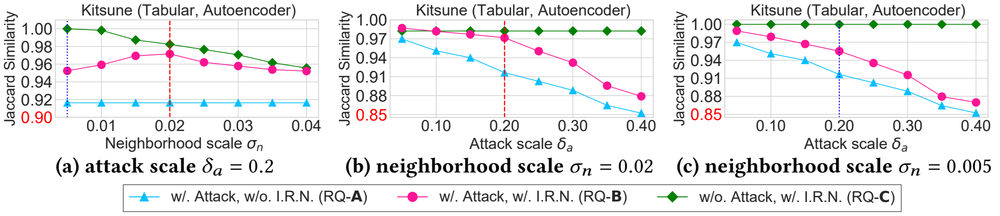
We evaluate the performance of DeepAID and baseline interpreters from four aspects: fidelity (with conciseness), stability, robustness, and efficiency, which are of particular concern when deployed in security domains. Their definitions are introduced in §4.1.

Fidelity-Conciseness Evaluation. To evaluate the fidelity of interpretations, we define an indicator similar to [20] called *Label Flipping Rate* (**LFR**) as the ratio of abnormal data that becomes normal after being replaced by interpretation results. The intuition is that LFR of high-fidelity interpreters will be higher since they accurately pinpoint important dimensions inducing the abnormality. Obviously, LFR tends to be higher when bringing more dimensions in interpretations, which destroys conciseness on the other hand. We evaluate this fidelity-conciseness trade-off in Figure 5a under tabular and time-series based scenarios. The x-axis is in decreasing order of used dimensions (increasing order of conciseness). From the tabular result, `CADE` generally outperforms approximation-based interpreters w.r.t. fidelity, while is not good as back propagation based ones. We can observe that directly selecting "reference" from



(a) Fidelity evaluation. **(b) Stability evaluation.** **(c) Robustness (to noise).**

Figure 4: Fidelity, stability, and robustness (to noise) evaluation of interpreters (*higher is better*).



(a) attack scale $\delta_a = 0.2$ **(b) neighborhood scale $\sigma_n = 0.02$** **(c) neighborhood scale $\sigma_n = 0.005$**

Figure 5: Adversarial robustness (to optimization-based attack) evaluation of DeepAID (*higher is better*).

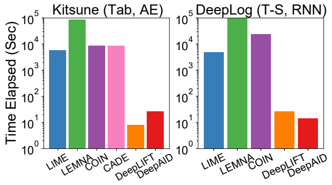


Figure 6: Efficiency evaluation of interpreters(*lower is better*).

existing in-distribution samples (S.R.T.D.) performs poorly especially with fewer dimensions, which demonstrates the necessity of *searching* reference in DeepAID. Results also demonstrate that supervised interpreters are not suitable for unsupervised learning due to their low fidelity. Particularly, DeepAID significantly outperforms other methods, especially when using fewer dimensions (e.g., DeepAID with 10% dimensions exceeds others by >50%).

Stability Evaluation. Stability of interpreters refers to the similarity of interpretations of the same samples during multiple runs. Like [14, 54], we ignore the value in results and keep the index of important feature dimensions. Given two interpretation vectors v_1 and v_2 of important dimension indexes, we leverage *Jaccard Similarity* (**JS**) to measure the similarity of two results, which is defined as $|\text{set}(v1) \cap \text{set}(v2)|/|\text{set}(v1) \cup \text{set}(v2)|$. We repeatedly measure the similarity of two interpretation results (from two runs with the same setting) for each anomaly and calculate the average. The results are shown in Figure 5b. We can observe that approximation/perturbation based methods perform poorly due to their random sampling/perturbation, while back propagation based methods (DeepAID and DeepLIFT) have strong stability.

Below, we evaluate two kinds of robustness of interpreters: robustness to *random noise* and *adversarial attacks*. Only tabular Kitsune is evaluated since time series in DeepLog is discrete.

Robustness Evaluation (to Noise). We measure **JS** of results interpreting tabular features before and after adding noise sampling from Gaussian $\mathcal{N}(0, \sigma^2)$ with $\sigma = 0.01$. As shown in Figure 5c, S.R.T.D. shows a high sensitivity to noise when selecting reference. COIN and DeepLIFT are relatively robust compare with other baselines (note that those unstable interpreters perform poorly even without noise), but are still lower than our DeepAID.

Robustness Evaluation (to Attacks). DeepAID falls into the category of back propagation (B.P.) based method. Thus, we borrow the ideas from existing adversarial attacks against B.P. based interpreters [18, 60] to develop an adaptive attack on DeepAID, called **optimization-based** attack, which can be defined as adding small perturbation on anomaly $\boldsymbol{x}^\circ$ (denoted with $\tilde{\boldsymbol{x}}^\circ$) that can induce large changes of finally searched reference $\boldsymbol{x}^*$. Formally,

$$\text{argmax}_{\tilde{\boldsymbol{x}}^\circ} \|\mathcal{D}_{tab}(\tilde{\boldsymbol{x}}^\circ; \tilde{\boldsymbol{x}}^\circ) - \mathcal{D}_{tab}(\boldsymbol{x}^\circ; \boldsymbol{x}^\circ)\|_p \ \text{ s.t. } \|\tilde{\boldsymbol{x}}^\circ - \boldsymbol{x}^\circ\|_p < \delta_a,$$

where δ_a limits the perturbation scale. Details of solving $\tilde{\boldsymbol{x}}^\circ$ are in Appendix D.1. As mentioned in §4.2, DeepAID can **I**nitialize **R**eference from the **N**eighborhood of anomaly (called **I.R.N.** henceforth) to improve the adversarial robustness. Below, we conduct three parts of experiments. (1) **RQ-A**: Robustness of DeepAID against the proposed attack without I.R.N. (2) **RQ-B**: Impact of I.R.N. against the attack. (3) **RQ-C**: Impact of I.R.N. on original stability. RQ-A or RQ-B is measured by **JS** of interpretations before and after the attack without (RQ-A) or with (RQ-B) I.R.N., and RQ-C is measured by **JS** with and without I.R.N. in the absence of attack.

We extensively evaluate the impact of perturbation scale δ_a of optimization-based attack and neighborhood scale σ_n of I.R.N.. Results are shown in Figure 5. We first fix δ_a in 5a and observe the impact of σ_n, and vice versa in 5b/5c. We find DeepAID without I.R.N. is naturally robust against such attack (**JS** > 0.91 when $\delta_a = 0.2$ and *linearly* decreases as δ_a increases in 5b/5c), thanks to the search-based idea analyzed in §4.2 (**RQ-A**). Results also demonstrate the effectiveness of I.R.N. In 5a, I.R.N. mitigates most attack effect when $\sigma_n \geq 0.02$ (**RQ-B**). We also find I.R.N. has a very small impact on the original stability (**JS** > 0.95 even when $\sigma_n = 0.04$), which can be viewed as a trade-off. Hence, a simple and safe way is to choose a small σ_n, which can also mitigate the attack effect without loss of the original stability, as demonstrated in 5c (**RQ-C**).

We also evaluate another type of adversarial attacks that misleads the distance calculations in DeepAID (called **distance-based** attacks). The results demonstrate the strong robustness of DeepAID against such attacks. For reasons of space, detailed definitions, results, and analysis against such attacks are in Appendix D.2.

Efficiency Evaluation. We evaluate the efficiency by recording runtime for interpreting 2, 000 anomalies for each interpreter. For approximation-based interpreters (LIME, LEMNA, and COIN), the time

Table 3: Example interpretations ($K = 5$) for `Kitsune`.

(a) Ground truth: Remote command execution

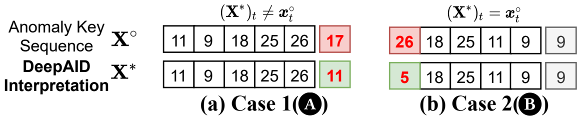
Feature Name	$\mathbf{x}^\circ$ <> $\mathbf{x}^*$ [1]	Feature Meaning	Expert’s Understanding
HpHp_1.0_mean	90 > 60	The average packet length of the current connection within 1 sec time window	Transport layer communication within a single connection
HpHp_1.0_std	30 > 10	The standard deviation of pkt length of the current connection within 1 sec	The length of the packet payload is not fixed (possibly some commands)
HH_1.0_mean	90 > 60	The average length of the all pkts with the same src-dst IPs within 1 sec	No other port is used between src-dst IPs (since 90 is the same as the first feature)
HpHp_5.0_std	20 > 10	The standard deviation of pkt length of the current connection within 5 sec	This is a rather longer communication (since 5 sec)
HpHp_3.0_mean	90 < 1,000	The average packet length of the current connection within 1 sec time window	Not a lot of content transferred (possibly short commands)

(b) Ground truth: ARP scan

Feature Name	$\mathbf{x}^\circ$ <> $\mathbf{x}^*$ [1]	Feature Meaning	Expert’s understanding
MI_dir_5.0_weight	$10^5 > 10^4$	# packets send from the same MAC&IP within 5 sec time window	A large number of packets sent from the same host and long-term (5 sec)
MI_dir_0.1_weight	300 > 100	# packets send from the same MAC&IP within 0.1 sec time window	A large number of packets sent from the same host (possibly Scan)
MI_dir_5.0_std	0.1 < 10	std. of packet length of the same MAC&IP within 5 sec	The length of packets sent from the same source are almost the same (std. is small)
MI_dir_0.1_std	1 < 10	std. of packet length of the same MAC&IP within 0.1 sec	The length of packets sent from the same source are almost the same (std. is small)
HH_1.0_weight	10 < 200	The average length of pkts with the same src-dest IPs within 1 sec	The communication is not from the same connection (10 is small, more likely Scan)

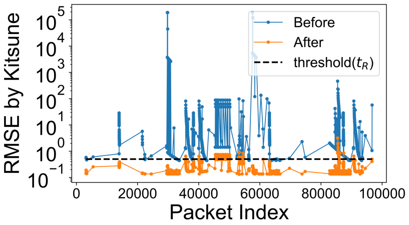
(c) False positive example

Feature Name	$\mathbf{x}^\circ$ <> $\mathbf{x}^*$ [1]	Feature Meaning	Expert’s understanding
HpHp_1.0_covariance	$10^{30} > 10$	The src-dest covariance of packet length of the current connection within 1 sec time window	There may be bugs or bias in the implementation of feature extraction
HpHp_1.0_pcc	$10^{28} > 1$	The src-dest correlation coefficient of packet length of the current connection within 1 sec time window	
HH_0.1_covariance	$10^{28} > 10$	The src-dest covariance of packet length with the same src-dest IPs within 0.1 sec time window	
HH_0.1_pcc	$10^{28} > 1$	The src-dest correlation coefficien of packet length with the same src-dest IPs within 0.1 sec time window	
HpHp_3.0_covariance	$10^{29} > 10$	The src-dest covariance of packet length of the current connection within 3 sec time window	


$(\mathbf{X}^*)_t \neq \boldsymbol{x}_t^\circ$
$(\mathbf{X}^*)_t = \boldsymbol{x}_t^\circ$
Anomaly Key Sequence $\mathbf{X}^\circ$
11 9 18 25 26 17
26 18 25 11 9 9
DeepAID Interpretation $\mathbf{X}^*$
11 9 18 25 26 11
5 18 25 11 9 9
(a) Case 1(A)
(b) Case 2(B)

Log Key	Meaning	Log Key	Meaning
5	Receiving block	18	Starting thread to transfer block
9	Receiving block with size	22	NameSystem(NS).allocateBlock
11	Responder for block terminating	25	Ask to replicate block
17	Failed to transfer block	26	NS.addStoredBlock: blockMap updated

Figure 7: Example interpretations ($K = 1, t = 6$) for `DeepLog`.



	TP	FP
Before	66,597	1,556
After	66,476	79
Change	-0.17%↓	-94.92%↓

* *The points in the left figure are all normal data (true label).*

Figure 8: Case for debugging `Kitsune`.

to train the surrogate model will also be counted, and the same for time to train the encoding model in `CADE`. The results are shown in Figure 6 under tabular and time-series based scenarios. We observe that DeepAID and `DeepLIFT` are at least two orders of magnitude faster than the other methods.

Conclusions. We conclude experiments of §6.2 here (and also in Table 1) that only our DeepAID can effectively meet all special requirements of security domains and produce high-quality interpretations for unsupervised DL-based anomaly detection.

Below, we will showcase how DeepAID can help to improving the practicability of security systems from several aspects. Figure 2 shows the flowchart of DeepAID usage with the following use cases.

6.3 Understanding Model Decisions

In this section, we use several cases to demonstrate that DeepAID can capture well-known rules (A in Figure 2) as well as discover new knowledge (B).

Tabular Data Interpretations. We use cases of network intrusion detector `Kitsune` for illustration. Two representative anomalies in Mirai botnet traffic are interpreted, where the first one is to remotely control the compromised bot to execute malware “mirai.exe” and execute a series of commands to collect information, and the second one is to ARP scan of open hosts in its LAN for lateral movement. Suppose that the operator already knows the ground truth (mentioned above) of these two anomalies by analyzing the raw traffic, their goal is to see whether the DL model has captured well-known rules with the help of DeepAID Interpreter. Here we use $K = 5$ important feature dimensions, and their names and meanings are listed in Table 3(a)(b), where the second column is the value of anomaly $\mathbf{x}^\circ$ and reference vector $\mathbf{x}^*$ in corresponding dimensions. Expert’s understandings based on the interpretations are also listed. Since (a) and (b) in Table 3 are both interpretable and reasonable for experts, we can draw the conclusion that DL model has learned the expected well-known rules.

Time-Series Interpretations. As shown in Figure 7, we use two simple cases of HDFS log anomaly detector `DeepLog` for illustration. We set sequence length $t = 6$ and use only $K = 1$ dimension for interpretation. Recall that time-series Interpreter needs to determine the location of anomalies before solving reference vector $\mathbf{X}^*$ through saliency testing (§4.3). In case 1, our Interpreter determines that the abnormality occurs at $\boldsymbol{x}_t^\circ$ and replaces 17 with 11. Since 17 is originally an abnormal log, we conclude that our Interpreter captures well-known rules. In case 2, Interpreter determines that abnormality occurs at $\boldsymbol{x}_1^\circ \boldsymbol{x}_2^\circ \ldots \boldsymbol{x}_{t-1}^\circ$ and replaces $\boldsymbol{x}_1^\circ$ with 5. The anomaly is probably because blockmap is updated without any preceding block operations. This is not an explicit rule and needs to be analyzed by domain experts, which demonstrates that DeepAID can discover new heuristics beyond explicit knowledge.

6.4 Enabling Debuggability

Below, we primarily use cases of `Kitsune` to illustrate how to use DeepAID interpretations to diagnose *explicit* errors in the system

[1] For ease of illustration, the feature values have been de-normalized and approximated.

Table 4: Performance of reliable detection based on Distiller.

Method		Number of Classes = 5					Number of Classes = 10				
		f1-micro	f1-macro	f1-micro*	f1-macro*	UACC	f1-micro	f1-macro	f1-micro*	f1-macro*	UACC
RF		0.9980	0.9983	0.6876	0.7351	N/A(0.)	0.9836	0.9827	0.7629	0.7848	N/A(0.)
MLP		0.8207	0.8491	0.8637	0.8399	N/A(0.)	0.8732	0.8791	0.8196	0.8101	N/A(0.)
Kitsune+ DeepAID	K=5	0.9782	0.9821	0.8904	0.8444	0.7330	0.9415	0.9408	0.8204	0.7780	0.5217
	K=10	0.9891	0.9911	0.9690	0.9641	0.7944	0.9797	0.9790	0.8822	0.8441	0.9357
	K=15	**0.9990**	**0.9991**	**0.9779**	**0.9736**	**1.0**	**0.9975**	**0.9975**	**0.9001**	**0.8728**	**0.9801**

f1-micro *and* ***f1-macro*** *are evaluated under* ***training data****, and* ***f1-micro**** *and* ***f1-macro**** *under* ***test data****;*
Kitsune (w/o. DeepAID): **UACC=0.28** *(not evaluated for f1-micro(*) and f1-macro(*) since it cannot classify multi-class data).*
(**Higher is better** for all five metrics.)

Table 5: Reducing FPs by Distiller.

Method		# classes = 5		# classes = 10	
		TPR	FPR	TPR	FPR
RF		0.9989	0.9719	0.9990	0.7504
MLP		1.0	0.0026	0.9998	0.3319
Kitsune		0.9755	0.0816	0.9717	0.0845
Kitsune+ DeepAID	K=5	**1.0**	**0.0**	0.9999	**0.0002**
	K=10	1.0	0.0	**1.0**	0.0046
	K=15	1.0	0.0	1.0	0.0220

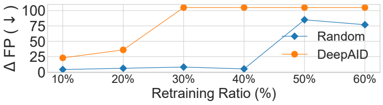


Figure 9: Reducing FPs by retraining.

(i.e., Ⓒ in Figure 2). Here *explicit* means human-factor errors that are easy to repair. (In contrast to the *implicit* errors caused by deviant model learning, which will be discussed in §6.6).

In Kitsune, we find some false positives (FP) with extremely high reconstruction error (RMSE). From interpretations (a representative one is listed in Table 3(c)), we find that these FPs are caused by morbidly large features related to the covariance (cov.) or Pearson correlation coefficient (pcc.). After analyzing the original traffic, we find the relevant features should not be so large. Therefore, we infer that there may be bugs or bias in the implementation of feature extraction algorithm in Kitsune. By examining the open-source implementation of Kitsune [35], we note that they claimed that they used approximation algorithms to compute related features instead of counting real values. Therefore, we modified the implementation of extracting cov./pcc. related features to calculate their real value. As shown in Figure 8 (with Mirai botnet traffic), by fixing this error, we reduce the FPs (94.92%↓) in Kitsune without much affecting the detection of anomalies (only 0.17%↓).

6.5 Human-in-the-Loop Reliable Detection

Next we evaluate the performance of Distiller as an extension to Interpreter to perform human-in-the-loop detection (§5). Specifically, we design experiments to answer the following questions:

- **RQ1:** Can Distiller accurately match the existing rules?
- **RQ2:** Whether Distiller has generalization ability to detect similar threats that do not appear in existing rules?
- **RQ3:** Can Distiller preserve the ability to detect unknown threats?

Experimental Setting. To evaluate Distiller, expert feedback to anomaly interpretations is required as rules for updating Distiller. However, human feedback is experience-dependent and very subjective. For fairness and simplicity, we select two well-labeled multiclass datasets, Kitsune dataset [35] and CIC-IDS2017 [43]. As mentioned in §5, we use two ML/DL classifiers as the baseline: Random Forest (RF) and Multi-Layer Perceptron (MLP). We also evaluate the impact of K for Distiller with $K = 5, 10, 15$. We respectively use 5 and 10 different classes/feedback states of attack traffic and 200 typical data(/rules) in each class for training/updating (thus totally 1000 and 2000). Note that, we *update* Distiller with the same training set for **RQ1**/**RQ2**/**RQ3** while *test* on training set (**RQ1**) and test set (**RQ2**). Particularly for **RQ3**, we test Distiller on another class of anomalies not in the training set. As for metrics, we evaluate f1-micro and f1-macro (f1-score in multi-classification) for **RQ1**/**RQ2**, and accuracy rate of unknown attack detection (UACC) for **RQ3**. Details about the datasets/metrics are in Appendix C.

Results and Conclusions. Results are shown in Table 5. From f1-micro/f1-macro, we find DeepAID can match existing rules very accurately (**RQ1**). From f1-micro*/f1-macro*, we find DeepAID significantly outperforms baselines on generalization ability (**RQ2**). Especially for K=15, f1-micro*/f1-macro* of DeepAID exceeds RF and MLP by 10% and 20%. We find MLP is even worse than RF, which may be because the training set is relatively small, while Distiller performs very well with fewer rules. The choice of K can be viewed as a trade-off. That is, a larger K ensures that the rules are more detailed (hence more accurate) but will lose the conciseness. From the results of UACC, we find that DeepAID not only retains the ability to detect unknown threats, but also significantly improves the original 28% to >98% (for K=15). This is because DL models failed to report unknown threats that are similar to normal data, while Distiller has explicit rules to match known anomalies, thus can judge unknown threats more accurately (**RQ3**). This also shows that DeepAID can also reduce false negatives (FN) incidentally. This experiment confirms DeepAID can help security systems to handle thousands of rules with superior performance on detecting stored (**RQ1**), similar (**RQ2**), and unknown (**RQ3**) anomalies.

6.6 Reducing FPs

We evaluate two methods mentioned in §5 to reduce FPs in security systems with Distiller (also Ⓓ and Ⓔ in Figure 2).

Modifying Distiller. We select only 10 rules from 10 FPs to a new feedback state representing normal data, and use the same baselines and datasets in §6.5. Here we consider all abnormal classes as one class and use two well-known metrics for evaluation: TPR and FPR (True/False Positive Rate). The results are shown in Table 5. DeepAID demonstrates extremely superior performance with nearly no FP and 100% TPR. It significantly reduces FPR of Kitsune (from 8% to 0%) while improving its TPR. The performance of baselines are poor with respect to FPR. Here we find a small number of FPs appears when K=15. This is because higher K is more likely to induce more "*conflicts*", which means that one state in the first FSM transits to multiple feedback states. We provide the detailed theoretical analysis of conflicts in Appendix B.

Retraining the DL model. Another method to reduce FPs mentioned in §5 is to retrain DL models with concept drift FPs found by Distiller. Here we use a simple baseline by randomly selecting FPs for retraining, in order to evaluate whether DeepAID can find the more optimal retraining set. As shown in Figure 9, we conclude that DeepAID can help to save the cost of retraining and achieve high performance (e.g., with DeepAID, retraining 30% samples reduces more FPs than 60% in the random method).

7 DISCUSSIONS

Below, we discuss design choices, limitations, and future works.

Why only Interpret Anomalies? In practical security applications, operators are more concerned about anomalies. Actually, they cannot even handle all anomalies, let alone the normal data.

Why White-box Interpretation? Besides the taxonomy in §2.1, local DL interpretations can also be separated as *white-box* and *black-box*. White-box methods require interpreters to have full access to the DL model, while black-box methods treat the DL model as a black box without detailed knowledge. DeepAID and DeepLIFT are white-box for using model gradients, while other baseline interpreters are black-box. Indeed, black-box methods are more convenient for operators. However, as justified in related works and our experiments (§6.2), they are difficult to use in practice due to the relatively poor performance. On the other hand, the white-box condition makes sense when the one seeking interpretations is the model owner. [20] thinks white-box condition is unreasonable when using pre-trained models, but this is not the case for unsupervised DL models since they do not need pre-training. In fact, DeepAID only requires the back-propagated gradient information with no need for fully understanding or modifying the original model, which avoids burdening the operators.

Limitations and Future Work. First, the adversarial robustness evaluation and claim of DeepAID are mainly against optimization-based and distance-based attacks (in §6.2 and Appendix D). There are other target attacks that may fail DeepAID, such as poisoning the original models (as DeepAID is highly faithful to model decisions), hijacking the search process to generate false references, and crafting anomalies *extremely* or *equally* far from the decision boundary in many dimensions to force interpretations to use more features (spoiling the conciseness). Future work can investigate the robustness against more attacks. Second, hyper-parameters of DeepAID are configured empirically. We have evaluated the sensitivity of hyper-parameters in Appendix F, and provided brief guidelines about how to configure them in the absence of anomalies. Future work can develop more systematic strategies to configure hyper-parameters. Third, we implement Distiller only for tabular data. Future work will focus on extending Distiller to other data types. Fourth, the practical effectiveness of DeepAID may depend on the expert knowledge of operators, since they do not have prior knowledge of anomalies in the unsupervised setting. Future work can investigate the impact of knowledge level and look into approaches to relax the requirement of expert knowledge for operators.

8 RELATED WORK

Most related works have been introduced in §2. Below, we briefly discuss other related works from two aspects.

Interpreting Unsupervised Learning & Anomaly Detection. Recently, there are a few studies discussing interpretation methods on unsupervised learning or anomaly detection. First, some studies develop global interpretation through model distillation [49] and rule extraction [5] as well as interpretation for clustering models [25], which are beyond the scope of this study. As for local interpretation of anomaly detection, in addition to COIN [31] and CADE [58] which are used as baselines, most of other works simply modify the existing supervised methods to unsupervised learning. For example, SHAP interpretation [32] is adapted to (single) Autoencoder in [2], which cannot be applied to Kitsune (ensemble Autoencoders) or other DL models. Deep Taylor Decomposition [36] is leveraged for interpreting one-class SVMs in [26], which cannot be applied to DNNs. In [7], a dedicated interpretation for Isolation Forest is developed which also cannot be applied to DNNs. An interpretation method for convolutional models and image analysis is proposed in [29] by visualizing abnormal region, which is unadaptable for other DL models in security domains. There are also works converting anomaly detection to supervised tasks and leverage existing supervised interpretations [1, 38]. Other works [4, 24] for interpreting anomaly in big-data streaming systems use the knowledge of many other anomalies (time-series intervals), which is hard to be generalized into the unsupervised setting in security domains.

Improving DL-based security systems. Recently, several works have been proposed for improving the practicality of (unsupervised) DL-based security systems, such as improving the robustness of DL-based systems [17, 21], performing lifelong learning [11], detecting concept drift [58], learning from low-quality labeled data [30]. They are orthogonal to our work and could be adopted together for developing more practical security systems.

9 CONCLUSIONS

In this paper, we propose DeepAID, a general framework to interpret and improve DL-based anomaly detection in security applications. We design DeepAID Interpreter by optimizing well-designed objective functions with special constraints for security domains to provide high-fidelity, human-readable, stable, robust, and efficient interpretations. Several applications based on our interpretations and the model-based extension Distiller are proposed to improve the practicality of security systems, including understanding model decisions, discovering new knowledge, diagnosing mistakes, and reducing false positives. By applying and evaluating DeepAID over three types of security-related anomaly detection systems, we show that DeepAID can provide high-quality interpretations for DL-based anomaly detection in security applications.

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their insightful comments. We also thank for the suggestions from Shize Zhang, Bin Xiong, Kai Wang, as well as all other members from NMGroup and CNPT-Lab in Tsinghua University. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1800200. Zhiliang Wang is the corresponding author of this paper.

REFERENCES

[1] Kasun Amarasinghe, Kevin Kenney, and Milos Manic. 2018. Toward explainable deep neural network based anomaly detection. In *2018 11th International Conference on Human System Interaction (HSI)*. IEEE, 311–317.

[2] Liat Antwarg, Bracha Shapira, and Lior Rokach. 2019. Explaining anomalies detected by autoencoders using SHAP. *arXiv preprint arXiv:1903.02407* (2019).

[3] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one* 10, 7 (2015), e0130140.

[4] Peter Bailis, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. 2017. MacroBase: Prioritizing Attention in Fast Data. In *SIGMOD Conference*. ACM, 541–556.

[5] Alberto Barbado and Óscar Corcho. 2019. Rule Extraction in Unsupervised Anomaly Detection for Model Explainability: Application to OneClass SVM. *CoRR* abs/1911.09315 (2019).

[6] Benjamin Bowman, Craig Laprade, Yuede Ji, and H Howie Huang. 2020. Detecting Lateral Movement in Enterprise Computer Networks with Unsupervised Graph AI. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*. 257–268.

[7] Mattia Carletti, Chiara Masiero, Alessandro Beghi, and Gian Antonio Susto. 2019. Explainable machine learning in industry 4.0: evaluating feature importance in anomaly detection to enable root cause analysis. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, 21–26.

[8] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (S&P)*. IEEE, 39–57.

[9] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 1–58.

[10] Chun-Hao Chang, Elliot Creager, Anna Goldenberg, and David Duvenaud. 2019. Explaining Image Classifiers by Counterfactual Generation. In *ICLR*. OpenReview.net.

[11] Min Du, Zhi Chen, Chang Liu, Rajvardhan Oak, and Dawn Song. 2019. Lifelong anomaly detection through unlearning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 1283–1297.

[12] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 1285–1298.

[13] Mengnan Du, Ninghao Liu, Qingquan Song, and Xia Hu. 2018. Towards explanation of dnn-based prediction with guided feature inversion. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. 1358–1367.

[14] Ming Fan, Wenying Wei, Xiaofei Xie, Yang Liu, Xiaohong Guan, and Ting Liu. 2020. Can We Trust Your Explanations? Sanity Checks for Interpreters in Android Malware Analysis. *IEEE Transactions on Information Forensics and Security* 16 (2020), 838–853.

[15] Ruth Fong, Mandela Patrick, and Andrea Vedaldi. 2019. Understanding Deep Networks via Extremal Perturbations and Smooth Masks. In *ICCV*. IEEE, 2950–2958.

[16] Ruth C Fong and Andrea Vedaldi. 2017. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 3429–3437.

[17] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 3–18.

[18] Amirata Ghorbani, Abubakar Abid, and James Zou. 2019. Interpretation of neural networks is fragile. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 33. 3681–3688.

[19] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2018. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)* 51, 5 (2018), 1–42.

[20] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. 2018. Lemna: Explaining deep learning based security applications. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 364–379.

[21] Dongqi Han, Zhiliang Wang, Ying Zhong, Wenqi Chen, Jiahai Yang, Shuqiang Lu, Xingang Shi, and Xia Yin. 2021. Evaluating and Improving Adversarial Robustness of Machine Learning-Based Network Intrusion Detectors. *IEEE Journal on Selected Areas in Communications* (2021).

[22] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. 2020. UNICORN: Runtime Provenance-Based Detector for Advanced Persistent Threats. In *Network and Distributed System Security Symposium (NDSS)*.

[23] Mashud Hyder and Kaushik Mahata. 2009. An approximate l0 norm minimization algorithm for compressed sensing. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 3365–3368.

[24] Vincent Jacob, Fei Song, Arnaud Stiegler, Yanlei Diao, and Nesime Tatbul. 2020. AnomalyBench: An Open Benchmark for Explainable Anomaly Detection. *CoRR* abs/2010.05073 (2020).

[25] Jacob Kauffmann, Malte Esders, Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. 2019. From Clustering to Cluster Explanations via Neural Networks. *CoRR* abs/1906.07633 (2019).

[26] Jacob Kauffmann, Klaus-Robert Müller, and Grégoire Montavon. 2020. Towards explaining anomalies: A deep Taylor decomposition of one-class models. *Pattern Recognition* 101 (2020), 107198.

[27] Alexander D. Kent. 2015. Cybersecurity Data Sources for Dynamic Network Research. In *Dynamic Networks in Cybersecurity*. Imperial College Press.

[28] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR (Poster)*.

[29] Shogo Kitamura and Yuichi Nonaka. 2019. Explainable Anomaly Detection via Feature-Based Localization. In *ICANN (Workshop) (Lecture Notes in Computer Science, Vol. 11731)*. Springer, 408–419.

[30] Junjie Liang, Wenbo Guo, Tongbo Luo, Vasant Honavar, Gang Wang, and Xinyu Xing. 2021. FARE: Enabling Fine-grained Attack Categorization under Low-quality Labeled Data. In *The Network and Distributed System Security Symposium 2021*.

[31] Ninghao Liu, Donghwa Shin, and Xia Hu. 2018. Contextual outlier interpretation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*. 2461–2467.

[32] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems (NIPS)*. 4765–4774.

[33] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. 2020. Interpreting Deep Learning-Based Networking Systems. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM)*. 154–171.

[34] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013), 3111–3119.

[35] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: an ensemble of autoencoders for online network intrusion detection. *25th Annual Network and Distributed System Security Symposium (NDSS)*.

[36] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. 2017. Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recognition* 65 (2017), 211–222.

[37] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. 2018. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing* 73 (2018), 1–15.

[38] Andrea Morichetta, Pedro Casas, and Marco Mellia. 2019. EXPLAIN-IT: Towards Explainable AI for Unsupervised Network Traffic Analysis. In *Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks*. 22–28.

[39] Anh Nguyen, Jason Yosinski, and Jeff Clune. 2016. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. *arXiv preprint arXiv:1602.03616* (2016).

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*. 8026–8037.

[41] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*. 701–710.

[42] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (KDD)*. 1135–1144.

[43] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization.. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*. 108–116.

[44] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning important features through propagating activation differences. In *International Conference on Machine Learning(ICML)*. 3145–3153.

[45] Lior Sidi, Yisroel Mirsky, Asaf Nadler, Yuval Elovici, and Asaf Shabtai. 2020. Helix: DGA Domain Embeddings for Tracking and Exploring Botnets. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management (CIKM)*. 2741–2748.

[46] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR (Workshop Poster)*.

[47] Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. 2020. Fooling lime and shap: Adversarial attacks on post hoc explanation methods. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society (AAAI)*. 180–186.

[48] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. 2017. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825* (2017).

[49] Fei Song, Yanlei Diao, Jesse Read, Arnaud Stiegler, and Albert Bifet. 2018. EXAD: A system for explainable anomaly detection on big data traces. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 1435–1440.

[50] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *ICML (Proceedings of Machine Learning Research, Vol. 70)*. PMLR, 3319–3328.

[51] Ruming Tang, Zheng Yang, Zeyan Li, Weibin Meng, Haixin Wang, Qi Li, Yongqian Sun, Dan Pei, Tao Wei, Yanfei Xu, et al. 2020. Zerowall: Detecting zero-day web attacks through encoder-decoder recurrent neural networks. In *39th IEEE*

Conference on Computer Communications (INFOCOM). IEEE, 2479–2488.

[52] Muhammad Fahad Umer, Muhammad Sher, and Yaxin Bi. 2017. Flow-based intrusion detection: Techniques and challenges. *Computers & Security* 70 (2017), 238–254.

[53] Ruoying Wang, Kexin Nie, Tie Wang, Yang Yang, and Bo Long. 2020. Deep Learning for Anomaly Detection. In *Proceedings of the 13th International Conference on Web Search and Data Mining (WSDM)*. 894–896.

[54] Alexander Warnecke, Daniel Arp, Christian Wressnegger, and Konrad Rieck. 2020. Evaluating explanation methods for deep learning in security. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 158–174.

[55] Mike Wu, Michael C Hughes, Sonali Parbhoo, Maurizio Zazzi, Volker Roth, and Finale Doshi-Velez. 2018. Beyond Sparsity: Tree Regularization of Deep Models for Interpretability. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

[56] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. 2018. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference (WWW)*. 187–196.

[57] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP)*. 117–132.

[58] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. 2021. CADE: Detecting and Explaining Concept Drift Samples for Security Applications. In *30th USENIX Security Symposium (USENIX Security)*.

[59] Houssam Zenati, Manon Romain, Chuan-Sheng Foo, Bruno Lecouat, and Vijay Chandrasekhar. 2018. Adversarially learned anomaly detection. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 727–736.

[60] Xinyang Zhang, Ningfei Wang, Hua Shen, Shouling Ji, Xiapu Luo, and Ting Wang. 2020. Interpretable deep learning under fire. In *29th USENIX Security Symposium (USENIX Security)*.

[61] Ying Zhong, Wenqi Chen, Zhiliang Wang, Yifan Chen, Kai Wang, Yahui Li, Xia Yin, Xingang Shi, Jiahai Yang, and Keqin Li. 2020. HELAD: A novel network anomaly detection model based on heterogeneous ensemble learning. *Computer Networks* 169 (2020), 107049.

[62] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. 2016. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. 2921–2929.

[63] Chong Zhou and Randy C Paffenroth. 2017. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 665–674.

[64] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).

[65] Luisa M. Zintgraf, Taco S. Cohen, Tameem Adel, and Max Welling. 2017. Visualizing Deep Neural Network Decisions: Prediction Difference Analysis. In *ICLR*. OpenReview.net.

APPENDICES

A ALGORITHMS OF INTERPRETERS

We provide the procedure of time-series and graph-data Interpreter.

A.1 Procedure of Time-Series Interpretation

The procedure of DeepAID Interpreter for time-series based systems is in Algorithm 2.

Algorithm 2: Interpreting time-series anomalies (univariate)

```
Input: Interpreted anomaly $X^\circ$; $max\_iter$, learning rate $\alpha$; $\mu_1, \mu_2$
Output: Interpretation result $X^\circ - X^*$ with the reference $X^*$
$\boldsymbol{x}^c \leftarrow \mathrm{argmax}_{\boldsymbol{x}^c}\ \Pr(\boldsymbol{x}^c | \boldsymbol{x}_1^\circ \boldsymbol{x}_2^\circ ... \boldsymbol{x}_{t-1}^\circ)$;
if $ST(X^\circ; \mu_1, \mu_2) = \boldsymbol{True}$ then                ▹ saliency testing
    $X^* \leftarrow \boldsymbol{x}_1^\circ \boldsymbol{x}_2^\circ ... \boldsymbol{x}_{t-1}^\circ \boldsymbol{x}^c$;
else
    Initialize $X^*$ with $X^\circ$; $t \leftarrow 0$;
    while $t < max\_iter$ do                                  ▹ iterative optimization
        $X^* \leftarrow$Adam($X^*; \mathcal{D}_{ts}, \alpha$) ;   ▹ update $X^*$ by Adam optimizer
        $i^* \leftarrow \mathrm{argmax}_i \left(\nabla \mathcal{D}_{ts}(X^*; \boldsymbol{x}_t^\circ)\right)_i$; ;        ▹ effectness
          measurement
        for $i = 1$ to $N$ do                  ▹ replacing ineffective dimensions
            if $i \neq i^*$ then  $(X^*)_i \leftarrow (X^\circ)_i$ ;
        end
        Discretize $X^*$ into one-hot vectors with only 0 or 1;
        $t \leftarrow t + 1$;
    end
end
return $X^\circ - X^*$
```

Algorithm 3: Interpreting graph link anomalies (unattributed)

```
Input: Link anomaly $\mathcal{X}^\circ = (x_a^\circ, x_b^\circ)$; $max\_iter$, learning rate $\alpha$,
       vertex set $\mathcal{V}$
Output: The reference $\mathcal{X}^* = (x_a^*, x_b^*)$ for interpretation
$\boldsymbol{e}^\circ \leftarrow \mathsf{E}_G(\mathcal{X}^\circ)$;
Solve $\boldsymbol{e}^*$ through Algorithm (1) with the input $\boldsymbol{e}^\circ$;
if $\mathsf{E}_G$ is differentiable then
    $\mathcal{X}^* \leftarrow$Adam($\mathcal{X}^*; \mathcal{D}_{gra}(\mathcal{X}^*; \boldsymbol{e}^*), \alpha$) ;   ▹ gradient-based method
    Discretize $\mathcal{X}^*$ into two one-hot vectors with only 0 or 1;
else                                                            ▹ greedy search
    for each $x \in \{x_a^\circ, x_b^\circ\}$ do                            ▹ Initialize PQ
        for each $y \in (\mathcal{N}_G(x) \cap \mathcal{V})$ do
            $\mathbf{v}.a \leftarrow x$; $\mathbf{v}.b \leftarrow y$;
            $\mathbf{v}.w \leftarrow \mathcal{D}_{gra}((\mathbf{v}.a, \mathbf{v}.b); \boldsymbol{e}^*)$;
            PUSH($\mathbf{v}$,PQ);
        end
        $\mathcal{V} \leftarrow \mathcal{V} - \{x\}$ ;               ▹ delete visited vertex from $\mathcal{V}$
    end
    Initialize $\mathcal{X}^*$ with $\mathcal{X}^\circ$; $t \leftarrow 0$;
    while $t < max\_iter$ and PQ $\neq \emptyset$ do                 ▹ prioritized BFS
        $\mathbf{u} \leftarrow$ POP(PQ);
        if $\mathbf{u}.w < \mathcal{D}_{gra}(\mathcal{X}^*; \boldsymbol{e}^*)$ then
            $\mathcal{X}^* \leftarrow (\mathbf{u}.a, \mathbf{u}.b)$ ;                       ▹ update reference
        for each $y \in (\mathcal{N}_G(\mathbf{u}.b) \cap \mathcal{V})$ do
            $\mathbf{v}.a \leftarrow \mathbf{u}.b$; $\mathbf{v}.b \leftarrow y$;
            $\mathbf{v}.w \leftarrow \mathcal{D}_{gra}((\mathbf{v}.a, \mathbf{v}.b); \boldsymbol{e}^*)$;
            PUSH($\mathbf{v}$,PQ);
        end
        $\mathcal{V} \leftarrow \mathcal{V} - \{\mathbf{u}.b\}$ ;            ▹ delete visited vertex from $\mathcal{V}$
        $t \leftarrow t + 1$;
    end
end
return $\mathcal{X}^*$
```

A.2 Procedure of Graph Data Interpretation

Recall in §4.4, we claim that problem (12) cannot be directly solved by gradient-based optimizer if E_G is indifferentiable. To address this problem, we propose an alternative greedy solution as follows: **Greedy Search.** In a nutshell, we start searching two reference nodes x_a^*, x_b^* from x_a°, x_b°, and gradually search *outward*. In this way, searched reference link will not be too far from the abnormal link, which increases the interpretability. Specifically, we denote the objective function in (12) with $\mathcal{D}_{gra}(\mathcal{X}^*; \boldsymbol{e}^*)$, which is used to measure the *priority* of link $\mathcal{X}^*$. Then, the greedy search is conducted by breadth-first search (BFS) with a *priority queue* denoted with PQ. Let $\mathcal{N}_G(x)$ represents the set of neighbor nodes of x. Initially, push links consisting of x_a°, x_b° and their neighbors (formally, $\{(x, y) | x \in \{x_a^\circ, x_b^\circ\}, y \in \mathcal{N}_G(x)\}$) into the queue PQ. By measuring the priority with $\mathcal{D}_{gra}$, the node popped from PQ and its (unvisited) neighbor nodes are then pushed into PQ. After a fixed number of iterations, the current optimal link will be used as the reference. The whole procedure of graph Interpreter is in Algorithm 3.

B THEORETICAL ANALYSIS OF DISTILLER

We provide theoretical analysis about the correctness of Distiller (in B.1) and its complexity (in B.2).

B.1 Theoretical Proofs

Preliminaries (rule, query, result). Recall that Distiller introduced in §5 consists of two FSMs, where the first one is used to store representative interpretation vectors and the second one is used to store feedback by experts. In update mode of Distiller, we call an interpretation vector containing K dimensions together with a *feedback* state as a "*rule*". In test mode of Distiller, we call the given interpretation vector a "*query*". The feedback state in the second FSM with the highest probability transferred from this query is called the "*result*" of this query, denoted with r_q. Interpretation vectors in *rule* or *query* $\boldsymbol{x}^{\circ} - \boldsymbol{x}^{*}$ need to firstly be mapped into states $\hat{S} = \hat{s}_1\hat{s}_2...\hat{s}_K$ in the first FSM. Then, for query $\hat{S}$, the result is derived by $r_q = \text{argmax}_{\hat{r}}\, \mathcal{P}\left(\hat{r} \mid \hat{S}\right)$ where $\mathcal{P}$ is computed by (14).

We first provide a theorem of the correctness of Distiller (also the correctness of (14)) as follows:

Theorem 1 (Correctness of Distiller). The queried probability is always equal to the probability expectation of all states transferring to the query result according to the queried states. Formally, as (14):

$$\mathcal{P}\left(\hat{r} \mid \hat{s}_1\hat{s}_2...\hat{s}_K\right) = \frac{1}{K}\sum_{i=1}^{K}\left(\mathcal{P}_2(\hat{r} \mid \hat{s}_i)\prod_{j=1}^{i-1}\mathcal{P}_1(\hat{s}_{j+1} \mid \hat{s}_j)\right).$$

Proof **of Theorem 1**. Obviously, $\mathcal{P}_1$ and $\mathcal{P}_2$ are two Markov process, or we say they have Markov property since probability distribution of next states of the process $\mathcal{P}_1$ or $\mathcal{P}_2$ depends only upon the present state. Meanwhile, $\mathcal{P}_1$ and $\mathcal{P}_2$ are statistically independent since they are updated separately. $\mathcal{P}_1$ captures the probability of $\hat{s}_i$ transferring to $\hat{s}_K$, while $\mathcal{P}_2$ captures the probability of $\hat{s}_i$ transferring to $\hat{r}$. Thus, we multiply $\mathcal{P}_1$ and $\mathcal{P}_2$ to compute the probability of co-occurrence of the two processes (due to the independence of $\mathcal{P}_1$ and $\mathcal{P}_2$). Finally, we calculate the mean for each state in $\hat{S}$ as the expectation of co-occurrence of the two processes. ■

Next, we further introduce some definitions of terms for sake of analyzing the accuracy bound of Distiller.

Definition 1 (State conflict). We say that ***state conflict*** happens when two rules indicating different feedback states have the same state(s) in $\hat{S}$. We say that the number of state conflicts in two rules is i if there are i identical states between two rules. For example, there are 2 state conflicts in $\hat{s}_1\hat{s}_2\hat{s}_3 \longrightarrow \hat{r}_1$ and $\hat{s}_2\hat{s}_4\hat{s}_1 \longrightarrow \hat{r}_2$, namely $\hat{s}_1$ and $\hat{s}_2$. Note that conflicts happen only when $\hat{r}_1 \neq \hat{r}_2$.

Definition 2 (Transition conflict). Similar to state conflict, we say that ***transition conflict*** happens when two rules indicating different feedback states have the same transition(s) in $\hat{S}$. For example, there are 2 state conflicts in $\hat{s}_1\hat{s}_2\hat{s}_3\hat{s}_4 \longrightarrow \hat{r}_1$ and $\hat{s}_3\hat{s}_4\hat{s}_1\hat{s}_2 \longrightarrow \hat{r}_2$, namely $\hat{s}_1\hat{s}_2$ and $\hat{s}_3\hat{s}_4$.

Definition 3 (Complete conflict). We say that ***complete conflict*** happens when there are K state conflicts and $K-1$ transition conflicts between two rules (K is the number of states in $\hat{S}$). For example, complete conflict happens in $\hat{s}_1\hat{s}_2\hat{s}_3\hat{s}_4 \longrightarrow \hat{r}_1$ and $\hat{s}_1\hat{s}_2\hat{s}_3\hat{s}_4 \longrightarrow \hat{r}_2$, namely only feedback states are different in two rules.

Definition 4 (Query accuracy). Given a Distiller in test mode and a validation query set with true-label feedback states, ***query accuracy*** can be computed as the ratio of the query result that equals the true label.

Intuitively, the conflicts, especially complete conflicts between rules are the root cause of the decrease in query accuracy. Next, we analyze the relationship between conflicts and query accuracy. Here we use R and F to respectively represent the set of all rules and feedback states. Therefore, the number of different rules and feedback are $|R|$ and $|F|$. Obviously, more rules represented in one Distiller will more likely to induce (more) conflicts. Below, we analyze the relationship between the number of rules, conflicts and query accuracy.

Lemma 1. Suppose that rules are randomly sampled (with uniform probability), the expectation of the number of different rules represented in one state is $\frac{K|R|}{MN}$, where M is the number of feature intervals in Distiller, and N is the total number of dimensions in tabular data, and K is the number of used dimensions in the interpretation vector.

Proof **of Lemma 1**. Recall that there are MN states totally in the first FSM. Because one rule will sample K states, and each state is sampled with equal probability, the probability of each state being selected by a rule is $\frac{K}{MN}$. Suppose that different rules are sampled independent, then for $|R|$ rules, we get the expectation of the number of different rules represented in one state $\frac{K|R|}{MN}$. ■

Lemma 2. Suppose that rules are randomly sampled (with uniform probability), the expectation of the number of states transferring to a certain feedback state in the second FSM is $\frac{K|R|}{MN|F|}$, where $|F|$ is the number of feedback states in the second FSM.

Proof **of Lemma 2**. Because rules are randomly sampled with uniform probability, the probability of a certain state to different feedback states is also equal. Then based on Lemma 1, the $\frac{K|R|}{MN}$ rules that represented by a certain state have equal probability transferring to $|F|$ feedback states, thus we get $\frac{K|R|}{MN|F|}$. ■

Theorem 2 (Query accuracy bound). The expectation of query accuracy rate is 100% if the number of rules $|R|$ represented by Distiller satisfies that $|R| < \frac{MN|F|}{K}$.

Proof **of Theorem 2**. If $|R| < \frac{MN|F|}{K}$, then $\frac{K|R|}{MN|F|} < 1$. Then according to Lemma 2, this means that the expectation of the number of a certain state transferring to a feedback state is less than 1. For a given rule $\hat{s}_1\hat{s}_2...\hat{s}_K \longrightarrow \hat{r}_{true}$, we consider the worst case that only states $\hat{s}_1\hat{s}_2...\hat{s}_K$ in this rule transfer to $\hat{r}_{true}$. This is to say, the number of rules where $\hat{s}_i (i \in \{1, 2, ..., K\})$ transferring to $\hat{r}_{true}$ is 1. According to Lemma 2, the number of rules where $\hat{s}_i (i \in \{1, 2, ..., K\})$ transferring to other feedback denoted with $\hat{r}_{false}$ are expected to be $\frac{K|R|}{MN|F|} < 1$. In this case, if we have a query $\hat{s}_1\hat{s}_2...\hat{s}_K$, we will have the result $\hat{r}_{true}$, according to the probability transition equation proven in Theorem 1. because:

$$\begin{aligned}\mathcal{P}\left(\hat{r}_{true} \mid \hat{s}_1\hat{s}_2...\hat{s}_K\right) &= \frac{1}{K}\sum_{i=1}^{K}\Big(\frac{1}{1+\varkappa(|F|-1)}\prod_{j=1}^{i-1}\mathcal{P}_1(\hat{s}_{j+1} \mid \hat{s}_j)\Big)\\ &> \frac{1}{K}\sum_{i=1}^{K}\Big(\frac{\varkappa}{1+\varkappa(|F|-1)}\prod_{j=1}^{i-1}\mathcal{P}_1(\hat{s}_{j+1} \mid \hat{s}_j)\Big)\\ &= \mathcal{P}\left(\hat{r}_{false} \mid \hat{s}_1\hat{s}_2...\hat{s}_K\right), \qquad (15)\end{aligned}$$

where $\varkappa = \frac{K|R|}{MN|F|} < 1$. We also depict Figure 10 for ease of understanding. Since for any query $\hat{S}$, we have $\mathcal{P}\left(\hat{r}_{true} \mid \hat{S}\right) > \mathcal{P}\left(\hat{r}_{false} \mid \hat{S}\right)$, this confirms the expectation of query accuracy rate is 100% (Note that this is under the worst case). ■

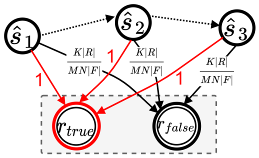


Figure 10: Illustration of Proof of Theorem 2.

Corollary 1. Distiller can store $\frac{MN}{K}$ different rules (in the first FSM) at most for the same feedback state (in the second FSM), to ensure that the expectation of query accuracy rate is 100%.

***Proof* of Corollary 1.** According to Theorem 2, the expectation of query accuracy rate is 100% if $|R| < \frac{MN|F|}{K}$, then $\frac{|R|}{|F|} < \frac{MN}{K}$. ■

Corollary 2. The expectation number of complete conflicts is less than 1 if $|R| < \frac{MN|F|}{K}$.

***Proof* of Corollary 2.** Assume for the purpose of contradiction that there has complete conflict (namely the number is equal or more than 1) when $|R| < \frac{MN|F|}{K}$, according to Theorem 2, query results related to the conflict will be wrong in the worst case. This indicates the query accuracy is not 100%, which contradicts Theorem 2. Thus, we conclude the expectation of the number of complete conflicts is less than 1 if $|R| < \frac{MN|F|}{K}$. ■

B.2 Complexity Analysis

Distiller supports online fashion and can be updated in linear time since only related states are updated. In test mode, the time complexity of Distiller is also linear according to (14). As for the space complexity, the primary cost is two FSMs maintained in Distiller. We use *probability transition matrix* to model two FSMs, which also referred to as stochastic matrix is commonly used to describe the transitions of a Markov process. Since it is a square matrix, and there are MN states totally, the space complexity of Distiller is $O((MN)^2)$. This shows that Distiller can be deployed in security systems with a low memory overhead with thousand-scale MN.

C DETAILED EXPERIMENTAL SETTINGS

Below, we introduce details of security systems and settings in our experiments which are omitted in the main body of our paper.

C.1 Datasets

For sake of fairness and reproducibility, we primarily evaluate anomaly detection based security systems with datasets released in their original work. The information of used datasets are listed in Table 6. For Kitsune, we use the dataset collected from a IoT (Internet-of-Things) network in its work. We additionally use another open dataset for network intrusion detection called CIC-IDS2017 [43], in order to collect various types of attack traffic for the evaluation of reliable detection by Distiller in §6.5. CIC-IDS2017 collected traffic for various common attacks in a large-scale testbed, which covers many common devices and middleboxes in the network. Anomalies indicate various types of network attacks. For DeepLog, we use HDFS dataset [57] which is the same dataset evaluated in its work. This dataset consists of key number sequence of Hadoop file system logs, and anomalies indicate malfunction or malicious behavior in the file system. For GLGV, we also use a dataset evaluated in its original work called LANL-CMSCSE[27], which collects tons of multi-source events of a real internal computer network in 58 consecutive days. Here we mainly use authentication events collected from individual devices. Anomalies here indicate lateral movement in the APT campaign.

Table 6: Datasets used in this study.

Security Systems	Datasets	# Samples	# Selected Anomaly
Kitsune[35] (Tabular Data based System)	Kitsune-Mirai[35]	764,137	78,739
	Kitsune-Fuzzing[35]	2,244,139	15,000
	Kitsune-SSDP Flood[35]	4,077,266	15,000
	Kitsune-ARP MitM[35]	2,504,267	15,000
	Kitsune-SYN DoS[35]	4,077,266	15,000
	IDS17-DDoS[43]	44,547,764	335,181
	IDS17-DoS Hulk[43]		10,000
	IDS17-Infiltration[43]		10,000
	IDS17-PortScan[43]		10,000
	IDS17-Botnet[43]		10,000
	IDS17-Slowhttptest[43]		5,000
DeepLog[12] (Time Series based System)	HDFS[12, 57]	553,366	16,838
GLGV[6] (Graph Data based System)	LANL-CMSCSE[27]	1,051,430,459	747

Table 7: Detection performance of interpreted systems.

Systems	Datasets	TPR	FPR
Kitsune*	Kitsune-Mirai	0.8549	0.0127
	IDS17-DDoS	0.9120	0.0012
DeepLog	HDFS	0.9144	0.0124
GLGV	LANL-CMSCSE	0.8099	0.0005

* *Original implementation instead of the modified version in §6.4.*

C.2 Implementations

Security Systems. For Kitsune, we primarily use its open-source implementation[1] (in §6.4, we modified this implementation to improve the system). By default, we use settings claimed in their paper and code. We use 50,000 normal samples for training its unsupervised DL models. For DeepLog, we primarily use an open-source implementation[2] and achieve similar performance to the original work. For GLGV, we implement it according to the details provided in the paper, and achieved a similar performance to the original paper (TP is slightly lower, but FP is also lower). The results of original performance of security systems are shown in Table 7.

[1] Kitsune: https://github.com/ymirsky/Kitsune-py

[2] DeepLog: https://github.com/wuyifan18/DeepLog

Baseline Interpreters. Similarly, we primarily use open-source implementations [3][4][5][6][7] and default parameters and settings for several baseline interpreters. For supervised baselines (LIME, LEMNA, and DeepLIFT), we need to first convert the unsupervised anomaly detection models to supervised learning models since the unsupervised DNNs used in security systems are threshold-based but not parametric. Thus, we use supervised DNNs trained with 5,000 normal data and additionally sampled 5,000 anomalies to approximate the detection boundary, and then evaluate the supervised interpreters on approximated models. For DeepLIFT, we randomly select normal training data as the "reference points" [44]. For CADE, since the interpreter is originally designed for explaining concept drift samples, we simulate interpreted anomalies in Kitsune Dataset as concept drift samples by training a supervised classifier with 5,000 normal data and 5,000 malicious data from CIC-IDS2017 Dataset. After training the concept drift detector (essentially Contrastive Autoencoder [58]) with two-class data, we interpret anomalies in Kitsune Dataset with CADE interpreter.

C.3 Experimental Settings

Below, we provide supplementary statements on the settings of experiments in our study.

Hardware. Our experiments are evaluated on a Dell PowerEdge R720 server using Ubuntu 14.04 LTS with 6 cores Intel Xeon(R) CPU E5-2630 @ 2.60GHz and 94G RAM. All deep learning models are using GPU mode with NVIDIA GeForce GTX 1080ti.

Software. The software implementation of DeepAID is primarily based on Python 3 with several packages, including scikit-learn for machine learning models and pytorch for deep learning models.

Settings of §6.2. Here we evaluate Kitsune with Kitsune-Mirai dataset. We use the first 50,000 normal data for training and use 10,000 anomalies for evaluating the performance of interpreters. For DeepLog, we use 5,000 normal data for training and use 5,000 anomalies for evaluating the performance of interpreters. For GLGV, we use 50M normal event logs for training and use 747 anomalies for evaluating Interpreter. Note that in the *efficiency* evaluation, we select 2,000 anomalies for tabular and time-series data and 500 anomalies for graph data, and record the runtime of interpreters.

Settings of §6.3. Note that the values in Table 3 are de-normalized since only the original value of the feature is interpretable. For ease of illustration, these values have been approximated.

Settings of §6.4. Here we evaluate Kitsune with Kitsune-Mirai dataset. And used the same method (with the first 50,000 normal data) for training. Then we use all 78,739 anomalies and 10,000 normal data for evaluating and comparing the performance before and after fixing the problem of feature extraction, which is discovered with the help of DeepAID interpretations.

Settings of §6.5. Here Kitsune is trained with the same method again. In the evaluation of 5 classes, we use all five Kitsune datasets, while we together use the first five CIC-IDS2017 datasets for the evaluation of 10 classes. The ratio of training set to test set is 1:9. Namely, we have totally 9,000 and 18,000 test data for 5 classes and 10 classes. This ratio is reasonable, because in practice it is impossible for the operators to update the rules to the Distiller frequently, indicating that the ratio of training data should be relatively small. For **RQ3**, we choose the last class of CIC-IDS2017 dataset as the "unknown" attack.

Settings of §6.6. Here we have two parts of experiments. The first part (i.e., modifying Distiller) uses the same settings as §6.5. In the second part (i.e., retraining the DL model), we train Kitsune with normal traffic from IDS2017, and evaluate it with Kitsune-Mirai traffic. Since these two datasets are collected from different testbeds, concept drift may occur during testing. Indeed, we find there are totally >2,000 FPs when using the modified version of Kitsune. In contrast to 79 FPs (recall in Figure 8), we confirm the occurrence of concept drift. In Figure 9, we choose 1,000 FPs to put into Distiller for concept drift detection, and use another 1,000 FPs for testing.

C.4 Evaluation Metrics

We introduce the definition of metrics for the evaluation of Distiller omitted in the main body, including f1-micro/f1-macro/UACC in §6.5 and TPR/FPR in §6.6. The detection performance of binary classification tasks can be measured by firstly counting its true positives (*TP*), true negatives (*TN*), false positives (*FP*), and false negatives (*FN*). In this context, we can define the evaluation metrics:

$$Precision = \frac{TP}{TP+FP}, \ \textbf{TPR} = Recall = \frac{TP}{TP+FN},$$

$$\textbf{FPR} = \frac{FP}{FP+TN}, \ \textbf{ACC} = \frac{TP+TN}{TP+FN+TN+FP},$$

$$\textbf{F1-Score} = \frac{2\times Precision \times Recall}{Precision + Recall}.$$

UACC is just ACC of treating the unknown class as positive. As for f1-micro/f1-macro, they are f1-score for multi-class evaluation. Specifically, f1-micro calculates metrics globally by counting the total TP, FN, and FP, while f1-macro calculates metrics for each label, and finds their unweighted mean.

D ADVERSARIAL ROBUSTNESS OF INTERPRETER

Below, we provide details of adaptive attacks for adversarial robustness evaluation of the Interpreter in §6.2. We show the strong adversarial robustness of DeepAID and analyze the reasons. As mentioned in §6.2, we evaluate two types of adversarial attacks, including optimization-based attack and distance-based attacks.

D.1 Robustness against optimization-based attack

Attack Methodology. As mentioned in §6.2, we borrow the ideas from existing adversarial attacks against B.P. based interpreters [18, 60] to develop the following adaptive attack on DeepAID:

$$\text{argmax}_{\tilde{x}^\circ} \|\mathcal{D}_{tab}(\tilde{x}^\circ;\tilde{x}^\circ) - \mathcal{D}_{tab}(x^\circ;x^\circ)\|_p \ \text{ s.t. } \|\tilde{x}^\circ - x^\circ\|_p < \delta_a, \tag{16}$$

where $\mathcal{D}_{tab}$ is the objective function of optimization (7). As mentioned in §4.2, our Interpreter iteratively optimizes (7) to find a better reference. However, such iterative optimization is hard for attackers to gain the overall gradients. From the perspective of

[3] LIME: https://github.com/marcotcr/lime

[4] LEMNA: https://github.com/nitishabharathi/LEMNA(DNN), https://github.com/Henrygwb/Explaining-DL(RNN)

[5] COIN: https://github.com/ninghaohello/Contextual-Outlier-Interpreter

[6] DeepLIFT: https://github.com/pytorch/captum

[7] CADE: https://github.com/whyisyoung/CADE

attackers, we simplify Interpreter into solving (7) only once. Hence, the optimization for interpretation is transformed into single-step gradient descent for $\mathcal{D}_{tab}$ with the input of anomaly $\boldsymbol{x}^{\circ}$. Formally, $\boldsymbol{x}^{*} \approx \boldsymbol{x}^{\circ} - \alpha\nabla_{\boldsymbol{x}^{\circ}}\mathcal{D}_{tab}(\boldsymbol{x}^{\circ};\boldsymbol{x}^{\circ})$. Then, the interpretation result is $\boldsymbol{x}^{\circ} - \boldsymbol{x}^{*} \approx \alpha\nabla_{\boldsymbol{x}^{\circ}}\mathcal{D}_{tab}(\boldsymbol{x}^{\circ};\boldsymbol{x}^{\circ})$. In this context, the objective function of optimization-based attack in (16) is transformed into:

$$\mathrm{argmax}_{\tilde{\boldsymbol{x}}^{\circ}} \|\alpha\nabla_{\tilde{\boldsymbol{x}}^{\circ}}\mathcal{D}_{tab}(\tilde{\boldsymbol{x}}^{\circ};\tilde{\boldsymbol{x}}^{\circ}) - \alpha\nabla_{\boldsymbol{x}^{\circ}}\mathcal{D}_{tab}(\boldsymbol{x}^{\circ};\boldsymbol{x}^{\circ})\|_{p}. \qquad (17)$$

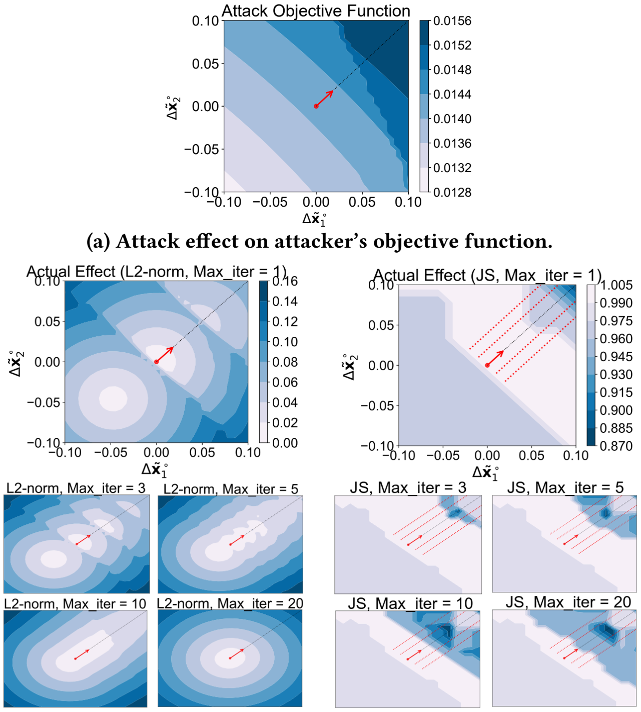
Here we set $p = 2$ for simplicity ($p = 1, 2$ in [18]). This problem can be easily solved by gradient-based methods. Like [18], to limit $\tilde{\boldsymbol{x}}^{\circ}$ and $\boldsymbol{x}^{\circ}$ to be close, we only modify top-K dimensions of $\tilde{\boldsymbol{x}}^{\circ}$ in each iteration and clip $\tilde{\boldsymbol{x}}^{\circ}$ to make $\|\tilde{\boldsymbol{x}}^{\circ} - \boldsymbol{x}^{\circ}\|_2 < \delta_a$.

Robustness Evaluation and Analysis. The results of DeepAID Interpreter against optimization-based attack have been shown in the main body (Figure 5 in §6.2). We conclude that DeepAID Interpreter is naturally robust against such attacks and becomes more robust with I.R.N.. In §4.2, we have analyzed the reason why DeepAID is naturally robust is due to iterative optimization and search-based idea. And I.R.N. can further improve the robustness by mitigating attack effect on the searching entrance. Specifically, with I.R.N., attackers can only use the perturbation solved from $\boldsymbol{x}^{\circ}$ to attack the searching process with the *actual* entrance $\boldsymbol{x}^{*}_{(1)} = \boldsymbol{x}^{\circ} + \mathcal{N}(0, \sigma_n^2)$. However, there is a gap between the effect of perturbation over $\boldsymbol{x}^{\circ}$ and actual attack effect on $\boldsymbol{x}^{*}_{(1)}$. We demonstrate the gap with an intuitive example in Figure 11.

Here we select an anomaly $\boldsymbol{x}^{\circ}$ and depict the influence of perturbations in two most important dimensions of $\tilde{\boldsymbol{x}}^{\circ}$ on the attacker's objective function (17) and the actual attack effect (i.e., the interpretation result). In Figure 11a, attackers choose the direction of perturbation according to the gradient of objective function (17), which is the red arrow in the figure. In Figure 11b, we evaluate two metrics for actual attack effect: the first one is the L2-distance between reference searching from $\tilde{\boldsymbol{x}}^{\circ}$ and $\boldsymbol{x}^{\circ}$ (left side of the figure), and the second one is **JS** between the two references (right side of figure). We evaluate the actual attack effect under different iterations (max_{iter}) of Interpreter. First, the results of L2-distance demonstrate that our Interpreter is naturally robust against such optimization-based attack since attackers cannot accurately simulate the overall gradients in our iterative optimization. As max_{iter} increases, the perturbation effect tends to be smooth, and the attack effect completely disappears when $max_{iter} = 20$. Second, the results of **JS** demonstrate the large gap between attacker's optimization function and actual effect. The area that improves the actual attack effect can be a very small and transient area in the gradient direction of the attacker's optimizer (See the results of JS when $max_{iter} = 3, 5$). In this context, I.R.N. with a small noise on the searching entrance can eliminate the attack effect (see red dotted lines). Third, we find that the value in the neighborhood of $\boldsymbol{x}^{\circ}$ usually remains the same (i.e., light-colored near the origin in Figure 11b), which demonstrates that I.R.N. with a small σ_N has little impact on the original stability.

D.2 Robustness against distance-based attacks

Attack Methodology. As mentioned in §6.2, we evaluate another type of attacks called distance-based attacks specifically designed against the DeepAID by mislead the distance calculations in DeepAID. The high-level idea of such attacks is to trick the Interpreter to choose *irrelevant* features (i.e., zero dimensions in interpretation $\boldsymbol{x}^{\circ} - \boldsymbol{x}^{*}$) by subtly perturbing them. Since there are two distance metrics in our Interpreter (L_0 and L_2, see Eq.(3)), we evaluate two distance-based attacks: L_0 attack adds small perturbations on groups of *irrelevant* features to trick the L_0 distance into selecting them, while L_2 attack adds a few *irrelevant* features with high values to spoil the computation of the L_2. Specifically, for L_0 attack, we choose 50% irrelevant features and add noise sampling from Gaussian $\mathcal{N}(0, \sigma^2)$ with default $\sigma = 0.01$. This evaluation is similar to robustness against noise in §6.2, while the difference is that L_0 attack only perturb 50% dimensions. For L_2 attack, we randomly choose one irrelevant dimension and change it into A times the maximum value in anomaly with default $A = 0.8$. Note that, the huge change to the selected dimension will change it from *irrelevant* to *relevant*. Thus we ignore the selected dimension when calculating JS, and only observe whether it has an impact on *other* irrelevant dimensions.

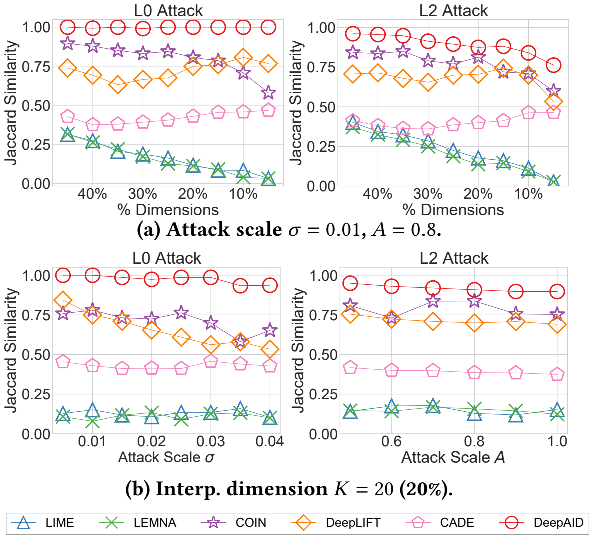


(a) Attack effect on attacker's objective function.

(b) Attack effect on actual Interpreter's results.

Figure 11: Example of demonstrating robustness against optimization-based attack.

Robustness Evaluation and Analysis. We use the same dataset in §6.2 and evaluate two distance-based attacks on DeepAID and other baseline Interpreters. We first evaluate with default σ or A and also evaluate the impact of two attack scale parameters. The results are shown in Figure 12. We can observe that DeepAID Interpreter is very robust against L_0 attack. This is because we use *gradient × input* together instead of only gradients when evaluating the effectiveness of dimensions (as introduced in §4.2). This effectively limits the impact of small perturbations on anomalies. As for L_2 attack, DeepAID also outperforms other baselines. This is because



(a) Attack scale $\sigma = 0.01$, $A = 0.8$.

(b) Interp. dimension $K = 20$ (20%).

Figure 12: Robustness evaluation against distance-based attacks (*higher is better*).

we only update a small number of features in each iteration to satisfy conciseness (as introduced in §4.2). Therefore, DeepAID can update selected features under L_2 attack without affecting many other features. This is also why in Figure 12a DeepAID becomes more robust against L_2 attack when the number of dimensions increases. The results in Figure 12b also demonstrate the strong robustness of DeepAID. Particularly, DeepAID is not sensitive to the degree of attack.

E GRAPH DATA BASED EXPERIMENTS

In the main body, we primarily use tabular and time-series based systems for evaluation and illustration due to space limit. Another reason is that we have not found any interpreter that can be used as a baseline for unsupervised graph learning tasks (Recall Table 1). Although there are a few methods suitable for graph data such as [33], they are difficult to migrate to unsupervised scenarios. Hence, we only evaluate DeepAID itself for graph data based systems. Below, we use GLGV to briefly illustrate the adoption of DeepAID on graph data based systems.

Backgrounds. The backgrounds of GLGV have been introduced in §2.3. GLGV [6] detects lateral movement in APT campaigns from authentication logs within enterprise networks in an unsupervised (specifically, reconstruction-based) fashion. It first builds authentication logs into *authentication graph*, where the nodes represent IP/device/users, and the edges represent authentication or access. Figure 13(a) shows an example of a benign authentication graph without any attack (i.e., normal data), which can be divided into three domains According to different permissions or authentication relationships. Only the administrator "admin" has the permission to access the Domain Controller (DC).

Settings. As mentioned in §4.4, we implement two versions of graph data Interpreter according to Algorithm 3, distinguished by whether E_G is differentiable, denoted with DeepAID and DeepAID′. DeepAID means E_G is differentiable, thus we use gradient-based optimization method in Interpreter, while DeepAID′ uses BFS-based method to address the indifferentiable problem. For DeepAID′, we use three max_{iter} (Abbreviated as m): $m = 5, 10, 20$. m directly reflects how many neighborhoods are viewed in the search process. We use 10% normal data with over 10M logs to construct the authentication graph, which consists of 47,319 nodes and 59,105,124 edges. We use 747 anomalies in the dataset for evaluation.

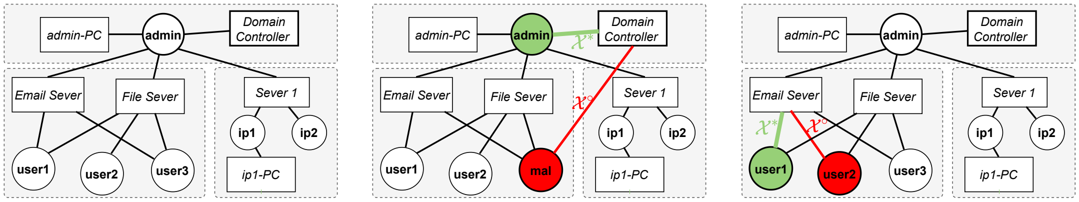
Performance of Interpreter. We primarily evaluate the fidelity and efficiency of DeepAID Interpreter. The same evaluation methods and indicators as §6.2 are used here. Note that runtime in efficiency evaluation is per 100 anomalies. The results are shown in Figure 14. We can find that DeepAID (with differentiable E_G) are more accurate (i.e., with high fidelity) but less efficient as the number of parameters of the neural network are too large and the back propagation is very slow. For search-based methods, DeepAID with larger m is more accurate while also more time-consuming (but is acceptable). In summary, for security systems whose E_G is indifferentiable, we recommend the search-based Interpreter, as DeepAID′ needs to re-implement E_G, which will increase the workload. For systems with differentiable E_G, both methods can be used depending on how the operator treats the fidelity-efficiency trade-off.

Usage of DeepAID. We provide two simple cases to illustrate how to use DeepAID on GLGV for interpreting anomalies and explaining FPs. We still use the scenario shown in Figure 13(a). As shown in Figure 13(b), suppose that an anomaly is reported by GLGV, which is the link from "user3" to DC (denoted by the red line). Now operators would like to find the basis of this decision-making by GLGV. After using our Interpreter, the reference link denoted by the green line from "admin" to DC is provided. This means that "user3" to DC is decided as an anomaly because GLGV thinks (only) "admin" can access to (or, authenticate to) DC. Furthermore, we can also reconstruct possible attack scenarios with the help of DeepAID interpretations: "user3" were compromised by the attacker (e.g., maybe after downloading malware from a phishing email). The attacker gaining the initial foothold of user3's devices would like to move laterally in this enterprise network for more harmful attacks. Several technicals such as vulnerability exploitation were used by the attackers to pursue a higher permission. Finally, the attacker successfully escalated her privilege to access the DC [6].

In Figure 13(c), we provide another case of using DeepAID to analyze FPs in GLGV. Here GLGV considers the link from "user2" to email server is an anomaly, which is actually an FP. From the interpretation result provided by our DeepAID, we can find that GLGV thinks (only) "user1" has the privilege to access the email server. In other words, GLGV thinks "user1" and "user2" are under different levels of privileges, which is not the case. Therefore, with the help of DeepAID, we can confidently conclude that GLGV make mistakes in its prediction of "user2". Through further analysis, we find that this is caused by insufficient representation ability of embedding vectors. Therefore, we solve this FP problem by increasing the length of neighbors traversed by random walk in GLGV and increasing the number of iterations of the embedding process.

F HYPER-PARAMETERS SENSITIVITY

DeepAID has some hyper-parameters in Interpreter and Distiller. Below, we first test the sensitivity of these hyper-parameters and then discuss how to configure them in the absence of anomalies.



(a) Example of normal authentication graph (b) Example of interpretation results (c) Example of interpreting FPs

Figure 13: Illustrations of interpreting graph data based system (GLGV).

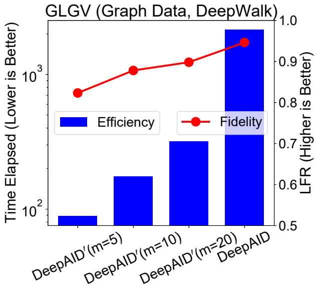


Figure 14: Performance of DeepAID Interpreter for graph data based system.

Hyper-parameters in DeepAID Interpreter. We evaluate six hyper-parameters in Interpreter, they are learning rate α in Adam optimizer, number of maximum iteration max_{iter} for optimization, λ in the stability term and ϵ in the fidelity term of objective functions (7), (8), (12). For time-series Interpreter, another two parameters μ_1 and μ_2 (for saliency testing) are tested. We set $K = 10$ (10%) for testing parameters in tabular Interpreter and set $K = 3$ (30%) for time-series Interpreter, and use the same datasets in 6.1. We primarily evaluate the sensitivity of fidelity (LFR). The stability and robustness are insensitive to parameters (except for optimization-based attack in D.1, which is more robust when max_{iter} increases). For efficiency, it is common sense that increasing the learning rate and reducing the number of iterations can reduce runtime, while other parameters are insensitive to efficiency. The testing method is to change one parameter with a reasonable range while fixing other parameters for each time. Table 8 lists their value range for testing, as well as default values when testing other parameters. The results of the change and standard deviation of LFR are also listed in Table 8.

From the results in Table 8, we can find the change of LFR is small (e.g., std. for all parameters is <1%). Thus, the performance of DeepAID Interpreter is basically insensitive to hyper-parameters.

Hyper-parameters in DeepAID Distiller. Distiller introduce only one new parameter M, the number of value intervals. Here we use the same settings in Table 4 and 5, and we set $K = 10$ and use 5-class datasets. We evaluate f1-macro, f1-macro*, TPR, FPR, and UACC under different M. The results are shown in Figure 15. We can find that the change of all metrics is very small when $M \geq 20$.

Hyper-parameters Configuration. We introduce the guideline of configuring hyper-parameters of DeepAID for operators. Especially in the unsupervised setting, hyper-parameters should be configured in the absence of anomalies (i.e., using normal data only). Note that operators have a high fault tolerance rate of choosing parameters since we have demonstrated their insensitivity. First, α and max_{iter} are easy to configure since the learning rate of Adam optimizer has been well studied in machine learning community, and max_{iter} should be increased appropriately if α is small. Also considering the adversarial robustness (D.1), we suggest that max_{iter} should not be too small, such as more than 5. λ balances the weight between fidelity and stability term. This is a free choice for operators. However, we should ensure that the values of the two terms are at the same order of magnitude. Thus, we can estimate the approximate value of the two terms, and can fine-tune the value after division. Note that, this does not need knowledge of anomalies since we only estimate the magnitude of two terms (i.e., only need to know the value range of normalized features). For configuring ϵ, we are more concerned about the upper bound, i.e., ϵ cannot be too large. A feasible method to measure the bound is to measure the value difference between the 99th quantile in normal data and the anomaly threshold (t_R, t_P). On the contrary, we are more concerned about the lower bound of μ_1. A feasible method is to measure the gradient term in (11) with all normal data, and choose the maximum (or quantile) as the lower bound of μ_1. For μ_2, it can be configured through computing the mean of $\Pr(x_t|x_1x_2...x_{t-1})$ in normal data. Note that, μ_2 must be greater than $1/N_{ts}$ where N_{ts} is number of "classes" of x_t, otherwise μ_2 cannot capture the class with the largest probability.

Table 8: Sensitivity test of hyper-parameters in Interpreter.

Parameters	Default	Testing Range	LFR (range)	LFR (std.)
α	0.5	[0.1, 5.0]	[0.9150, 0.9215]	0.0031
max_{iter}	20	[10, 100]	[0.9150, 0.9150]	0.0000
λ	0.001	[0.0005,0.005]	[0.8954, 0.9150]	0.0078
ϵ	0.01	[0.005, 0.05]	[0.9019, 0.9281]	0.0098
μ_1	0.01	[0.001, 0.1]	[0.9525, 0.9525]	0.0000
μ_2	0.3	[0.2, 0.4]	[0.9452, 0.9643]	0.0061

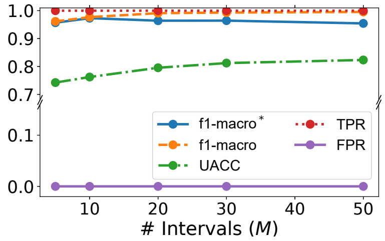


Figure 15: Sensitivity test of hyper-parameter M in Distiller.